Victor Khomenko
Olivier H. Roux (Eds.)

LNCS 10877

Application and Theory of Petri Nets and Concurrency

39th International Conference, PETRI NETS 2018
Bratislava, Slovakia, June 24–29, 2018
Proceedings

Springer

Lecture Notes in Computer Science 10877

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

More information about this series at http://www.springer.com/series/7407

Victor Khomenko · Olivier H. Roux (Eds.)

Application and Theory of Petri Nets and Concurrency

39th International Conference, PETRI NETS 2018
Bratislava, Slovakia, June 24–29, 2018
Proceedings

Springer

Editors
Victor Khomenko
University of Newcastle
Newcastle upon Tyne
UK

Olivier H. Roux
École Centrale de Nantes
Nantes Cedex 3
France

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-91267-7 ISBN 978-3-319-91268-4 (eBook)
https://doi.org/10.1007/978-3-319-91268-4

Library of Congress Control Number: 2018942335

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG
part of Springer Nature
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume constitutes the proceedings of the 39th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2018). This series of conferences serves as an annual meeting place to discuss progress in the field of Petri nets and related models of concurrency. These conferences provide a forum for researchers to present and discuss both applications and theoretical developments in this area. Novel tools and substantial enhancements to existing tools can also be presented.

Petri Nets 2018 was colocated with the Application of Concurrency to System Design Conference (ACSD 2018). Both were organized by the Interes Institute and Faculty of Electrical Engineering and Information Technology, Slovak University of Technology. The conference took place at the Austria Trend Hotel Bratislava, during June 24–29, 2018. We would like to express our deepest thanks to the Organizing Committee chaired by Gabriel Juhás for the time and effort invested in the local organization of this event. This year, 33 papers were submitted to Petri Nets 2018 by authors from 19 different countries. Each paper was reviewed by three reviewers. The discussion phase and final selection process by the Program Committee (PC) were supported by the EasyChair conference system. From 23 regular papers and ten tool papers, the PC selected 23 papers for presentation: 15 regular papers and eight tool papers. The number of submissions was a bit lower than expected. However, we were pleased that several highly innovative and very strong papers were submitted. After the conference, some of these authors were invited to submit an extended version of their contribution for consideration in a special issue of a journal.

We thank the PC members and other reviewers for their careful and timely evaluation of the submissions and the fruitful constructive discussions that resulted in the final selection of papers. The Springer LNCS team (notably Anna Kramer and Alfred Hofmann) provided excellent and welcome support in the preparation of this volume. We are also grateful to the invited speakers for their contributions:

- Orna Grumberg, Israel Institute of Technology, Haifa, Israel, who delivered the Distinguished Carl Adam Petri Lecture "Semantic Difference for Program Versions"
- Fabrice Kordon, Université P & M. Curie, Paris, France, "Self-Adaptive Model Checking, the Next Step?"
- Matthias Függer, Max-Planck-Institut für Informatik, Germany, "Challenges of Circuit Design: Circuits as Robust Distributed Algorithms"

Alongside ACSD 2018, the following workshops were colocated: the Workshop on Petri Nets and Software Engineering (PNSE 2018) and the Workshop on Algorithms and Theories for the Analysis of Event Data (ATAED 2018). Other colocated events included: the Model Checking Contest, the Petri Net Course, an Advanced Tutorial

on Verification, and an Advanced Tutorial on Process Mining (A Tour In Process Mining: From Practice to Algorithmic Challenges).

We hope you enjoy reading the contributions in this LNCS volume.

June 2018

Victor Khomenko
Olivier H. Roux

Organization

Steering Committee

W. van der Aalst	RWTH Aachen University, Germany
J. Kleijn	Leiden University, The Netherlands
L. Pomello	Università degli Studi di Milano-Bicocca, Italy
G. Ciardo	Iowa State University, USA
F. Kordon	Université P & M. Curie, France
W. Reisig	Humboldt-Universität zu Berlin, Germany
J. Desel	University of Hagen, Germany
M. Koutny (Chair)	Newcastle University, UK
G. Rozenberg	Leiden University, The Netherlands
S. Donatelli	Universita' di Torino, Italy
L. M. Kristensen	Western Norway University of Applied Sciences, Norway
M. Silva	Universidad de Zaragoza, Spain
S. Haddad	ENS Cachan, France
C. Lin	Tsinghua University, China
A. Valmari	University of Jyväskylä, Finland
K. Hiraishi	Japan Advanced Institute of Science and Technology, Japan
W. Penczek	Institute of Computer Science PAS, Poland
A. Yakovlev	Newcastle University, UK

Program Committee

Didier Buchs	CUI, University of Geneva, Switzerland
Lawrence Cabac	University of Hamburg, Germany
Maximilien Colange	Laboratoire de Recherche et Développement de l'EPITA, France
José-Manuel Colom	University of Zaragoza, Spain
Isabel Demongodin	LSIS - UMR CNRS 7296, France
Dirk Fahland	Eindhoven University of Technology, The Netherlands
Gilles Geeraerts	Université libre de Bruxelles, Belgium
Henri Hansen	Tampere University of Technology, Finland
Petr Jancar	Palacky University Olomouc, Czech Republic
Ryszard Janicki	McMaster University, Canada
Gabriel Juhas	Slovak University of Technology Bratislava, Slovakia
Victor Khomenko (Chair)	Newcastle University, UK
Jetty Kleijn	LIACS, Leiden University, The Netherlands
Fabrice Kordon	LIP6/Sorbonne Université and CNRS, France
Lukasz Mikulski	Nicolaus Copernicus University, Torun, Poland
Andrew Miner	Iowa State University, USA

Giovanni Michele Pinna	Università di Cagliari, Italy
Pascal Poizat	Université Paris Nanterre and LIP6, France
Olivier H. Roux (Chair)	LS2N/Ecole Centrale de Nantes, France
Pawel Sobocinski	University of Southampton, UK
Wil van der Aalst	RWTH Aachen University, Germany
Irina Virbitskaite	A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Russia
Matthias Weidlich	Humboldt-Universität zu Berlin, Germany
Karsten Wolf	Universität Rostock, Germany

Additional Reviewers

Barylska, Kamila
Brenner, Leonardo
Cerna, Ivana
Erofeev, Evgeny
Frutos Escrig, David
Gogolinska, Anna
Haustermann, Michael
Hillah, Lom Messan
Hoogeboom, Hendrik Jan
Jezequel, Loig
Klikovits, Stefan
Korovina, Margarita
Leroux, Jérôme
Linard, Alban
Mhaskar, Neerja
Moldt, Daniel
Mosteller, David
Piatkowski, Marcin
Racordon, Dimitri
Rietveld, Kristian
Sawa, Zdenek
Tarasyuk, Igor
Thierry-Mieg, Yann
Zuberek, Wlodek

Contents

Languages

Semantics and Expressiveness

Tools

Invited Talk

Self-adaptive Model Checking, the Next Step?

Fabrice Kordon(✉) and Yann Thierry-Mieg

Sorbonne Université, CNRS LIP6, 75005 Paris, France
{Fabrice.Kordon,Yann.Thierry-Mieg}@lip6.fr

Abstract. Model checking is becoming a popular verification method that still suffers from combinatorial explosion when used on large industrial systems. Currently, experts can, in some cases, overcome this complexity by selecting appropriate modeling and verification techniques, as well as an adapted representation of the system. Unfortunately, this cannot yet be done automatically, thus hindering the use of model checking in industry.

The objective of this paper is to sketch a way to tackle this problem by introducing *self-adaptive model checking*. This is a long term goal that could lead the community to elaborate a new generation of model checkers able to successfully push forwards the scale of the systems they can deal with.

Keywords: Verification · Model checking
Formal methods and methodology · Benchmark for verification

1 Introduction

Model checking is becoming a popular verification method, even in large companies such as Intel, Motorola and IBM [18]. There are already many success stories involving this technique. PolyORB, an open source middleware now used in aerospace applications, was formally verified to prove that no deadlocks nor livelocks could occur on one execution node [24]. Similarly, some aspects of the bluetooth interaction protocols have been studied formally [12]. Finally, NASA development of critical code also involved model checking to ensure its safety [23].

The main advantage of model checking is to be quite easy to automate, and so, it can be operated by non experts. Unfortunately, it suffers from the so-called combinatorial state explosion, that is difficult to tackle, especially for non experts. This can be called the "model checking dilemma": on the one hand, this approach is easy to use but as soon as you deal with complex problems, an expert aware of the various appropriate techniques and algorithms is required to complete the verification task.

Today, numerous techniques have been defined by the various communities working on the topic. They may rely on several types of automata like Büchi [6], Rabin [30], Streett [34], testing automata [17] and variants [2], etc. They may

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 3–15, 2018.
https://doi.org/10.1007/978-3-319-91268-4_1

also involve several techniques to describe the system such as symmetry reductions [8], various types of decision diagrams [7,20,33], partial order reductions [10], on-the-fly automata reductions [15], etc. Moreover, sometimes, several techniques are combined like Binary decision diagrams and symmetry reductions in [36], on-the-fly reductions and hierarchical decision diagrams in [13], and hierarchical decision diagrams and symmetry reductions in [11]. Finally, the analysis of properties is also a part of the optimization problem since there may exist some particular cases where some adapted algorithm performs better [31], automata derived from formulas can also be optimized [1] and observed elements in the system taken into consideration to reduce complexity [26].

However, when analyzing the behavior of model checkers on large benchmarks, such as the one of the Model Checking Contest[1] [28], we can notice that several combinations of techniques can be successfully operated to solve some classes of problems (e.g. LTL, CTL, reachability, bound computation, etc.). Identifying such combinations of techniques is thus of great help.

But so far, only experts can estimate which combination of techniques will be the more likely to solve a complex verification problem. The idea of *"self-adaptive"* model checking is to define the bases of an infrastructure embedding the capability to select and use the best data-representations and algorithms so that it can transparently tackle the complexity when performing verification.

This paper is structured as follows. Section 2 presents a simple analysis of the techniques used by model checking tools participating to the Model Checking Contest in 2015, 2016 and 2017. In particular, we are trying to extract some information about the involved techniques and focus on determining wether symbolic (based on decision diagram) or explicit approaches are the most efficient. Then, Sect. 3 defines what we mean with "self-adaptive model checking" before Sect. 4 discusses the impact of this approach on tools architecture. Section 5 discusses some issues to be solved to enable meta-heuristics to select an appropriate combination of algorithms to solve a verification problem. Section 6 concludes the paper.

2 Information Gathered from the Model Checking Contest

For more than a decade, we observed the emergence of software contests that assess the capabilities of verification tools on complex benchmarks. It is a way to identify the theoretical approaches that are the most fruitful in practice, when applied to realistic examples. Such events motivate the involved community to improve research tools and measure the benefits gained by new improvements.

They are also interesting because they bring representative and shared benchmarks to the involved communities. Moreover, since many tools compete, it is possible to gather and analyze information from the detailed outputs produced by such events. As a typical example, the Model Checking Contest benchmark is growing every year thanks to models proposed by the community. In 2017, it was composed of 78 models from which, thanks to some scaling parameters,

[1] See http://mcc.lip6.fr.

812 instances are derived. Numerous formulas (reachability, CTL, LTL) are also available (a new set is produced every year).

Models proposed by the Model Checking Contest mainly represent concurrent systems and are expressed in PNML [21] (Petri Net markup Language, an ISO/IEC standard to describe Petri net specifications). Some are derived from higher specification languages or translated from code. Some others are natively expressed using Petri nets. Many models come from a wide range of application domains and describe hardware systems, protocols, distributed algorithms, biological systems, etc. Some models are extracted from research papers and denote interesting "theoretical" configurations to be analyzed.

In the Model Checking Contest, tools are confronted to several examinations: StateSpace, UpperBounds, Reachability, CTL and LTL. *StateSpace* requires the tool to compute the full state space of a specification and then provide informations about it. Mandatory information concerns the number of states but tools may also provide additional informations like the number of transitions, the maximum number of tokens per marking in the net and the maximum number of tokens that can be found in a place.

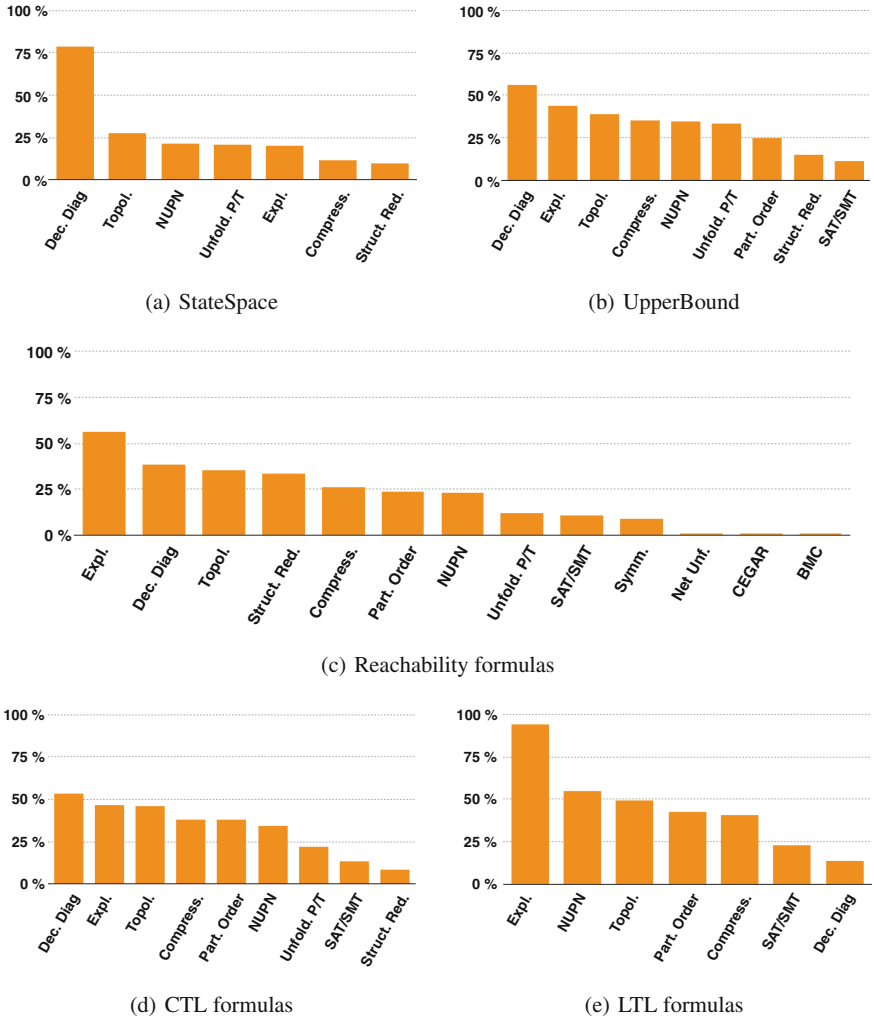
UpperBounds requires the tool to compute as a integer value, the exact upper bound of a list of places designated in a formula (there are 16 formulas per model instance).

Reachability, *CTL*, and *LTL* require the tool to evaluate if formulas are satisfied or not. For each formulas, we consider atomic propositions referring to either the marking of places or the fireability of transitions (16 formulas of each type are provided per model instance). In the reachability examination, there are extra formulas to check if there exist a deadlock.

So, the Model Checking Contest could be seen as a way to observe and evaluate the most successful techniques for a given type of model checking activity. To do so, we have analyzed the techniques participating tools reported to use over the three last years of the Model Checking Contest (2015, 2016, and 2017) where data was collected using comparable data formats. All the reported techniques are listed in Table 1.

Table 1. List of techniques reported by tools.

Technique	Explanation
BMC	The tool uses Bounded Model Checking and/or K-induction techniques
CEGAR	The tool uses a CEGAR [9] approach
Compress.	The tool uses some compression technique (other than decision diagrams)
Dec. Diag	The tool uses a kind of decision diagram
Expl.	The tool does explicit model checking
Net Unf.	The tool uses McMillan unfolding [29]
NUPN	The tool exploits the structural information provided in the NUPN [16] format
Part. Order	The tool uses some partial order technique
SAT/SMT	The tool relies on a SAT or SMT solver
Struct. Red.	The tool uses structural reductions (Berthelot, Haddad, etc.)
Symm.	The tool exploits symmetries of the system
Topol.	The tool uses structural informations on the Petri net itself (invariants, etc.)
Unfold. P/T	The tool transforms colored nets into their equivalent P/T



(a) StateSpace (b) UpperBound

(c) Reachability formulas

(d) CTL formulas (e) LTL formulas

Fig. 1. Cumulated declarations of techniques per successful examination as reported by tools during the Model Checking Contest over the 2015, 2016, and 2017 editions.

Figure 1 reports, for the valid answers, the percentage of techniques used by tools to compute examinations (they sometimes use several techniques simultaneously). Unfortunately, these techniques are so far reported per examination, even if, often, an examination contains 16 formulas to be evaluated. However, it is interesting to observe the top winning techniques for each examination categories.

It is easy to see that, based on these raw data, symbolic model checking (based on decision diagrams) is used more often than explicit techniques for the StateSpace, the UpperBound, and the CTL examinations. Explicit approaches appear more often in reachability and LTL examinations. Tools also report a large number of additional techniques like compression, partial order, or the use

Table 2. Number of tools relying on symbolic versus explicit approaches for participating tools (in the 2015, 2016 , and 2017 editions). Several participations of a given tool are cumulated.

Technique	StateSpace	UpperBound	Reachability	CTL	LTL
Symbolic tools	22 (71%)	8 (57%)	11 (44%)	10 (63%)	3 (27%)
Explicit tools	9 (29%)	6 (43%)	14 (56%)	6 (27%)	80 (73%)
Total tools	30	13	24	15	10

of structural informations to optimize model checking. Some tools also rely on Constraint solving, CEGAR [9] or bounded model checking.

However, this raw data must then be normalized because the number of tools declaring a technique may change from one examination to another as shown in Table 2. For example, the raw value declared for Decision Diagrams in Fig. 1(a) must be pondered by the fact that more tools between 2015 and 2017 rely on this technique (so it is naturally reported more often).

Symbolic approaches (based on some type of decision diagram) are usually confronted to explicit ones (often associated with other optimization techniques) in the community since they are in mutual exclusion. It is thus of interest to refine the raw data of Fig. 1 by focusing on these two approaches and check which one seems to be the most efficient and in which situations.

Figure 2 summarizes the ponderated ratio between Symbolic and Explicit approaches for the model checking contest examinations proposed in 2015, 2016 an 2017. For each examination, the first large bar shows the ratio between the tools declaring the use of Decision diagrams[2] and those declaring the use of explicit approaches[3]. The two other thinner bars present a "normalized" success rate when considering the respective number of tools using the corresponding technique. All these data are collected for successful examinations only (so the two approaches can claim together 100% of success).

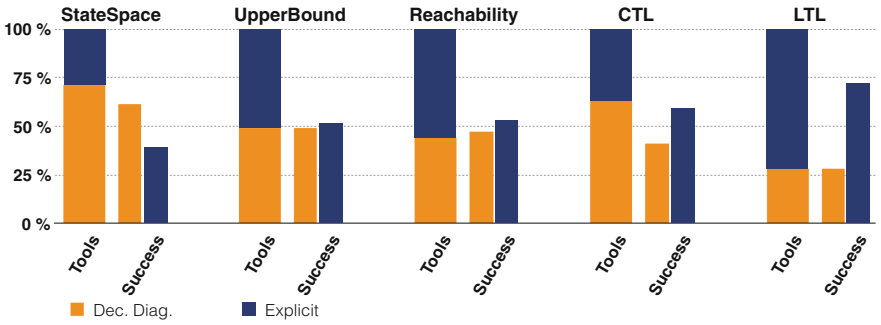


Fig. 2. Normalized measure of the success of decision diagram based techniques versus explicit ones (for the 2015, 2016, and 2017 editions) (Color figure online)

[2] Orange or gray in B&W.
[3] Dark blue or black in B&W.

Table 3. Analysis of the formulas computed by tools over the 2015, 2016 and 2017 editions of the Model Checking Contest. Formulas computed by several tools are cumulated.

	Reachability	CTL	LTL	Total
Satisfied computed formulas	205 478 (51%)	70 824 (45%)	28 773 (23%)	
Unsatisfied computed formulas	197 280 (49%)	84 990 (55%)	97 662 (77%)	
Total computed formulas	402 758	155 814	126 435	685 007

From these normalized data, it appears that symbolic and explicit techniques are quite comparable. For the StateSpace examination, decision diagram technology is a main factor of performance since the full state space must be computed. Figure 1(a) also shows that topological approaches (for example, based on the exploitation of NUPN[4] data) are quite useful to optimize the encoding of the state space using decision diagrams.

For the UpperBound examination, results are more balanced but slightly in favor of explicit approaches. Similarly to the StateSpace examination, Fig. 1(b) outlines an extensive use of topological information (including the use of NUPN data) to compute an appropriate variable order for decision diagrams. Compression mechanisms are also reported to be associated with explicit approaches.

For the evaluation of formulas (reachability, CTL, LTL), there seems to be some advantage to explicit techniques too. This is less clear for reachability formulas, more evident for CTL ones, and particularly true for LTL ones. However, for LTL, we must consider two factors that reduce the relevance of these measures. First, the LTL formula generator used in the Model Checking Contest remains quite basic compared to the one of reachability and CTL formulas (the team is working on this). Second, while reachability and CTL computed formulas are quite balanced between the satisfied and unsatisfied ones (see Table 3), this is not the case for LTL ones (more than $\frac{3}{4}$ are unsatisfied, so that some counter-example might be found rapidly). These two factors probably hinder any conclusion for LTL at this stage.

This simple study, based on he output of a single verification competition, may help to understand how and when one could operate a set of techniques to perform model checking. Unfortunately, even if several involved tools do not come from the Petri net community, the inputs of the Model Checking Contest all represent concurrent systems expressed using Petri nets. So, there might be some bias in the way such models are processed. It would be great if a similar analysis could be done on other verification contests. This is a complex task requiring at least some common glossary and notions to be defined.

3 What is Self-adaptive Model Checking?

The term "adaptive model checking" [19] was first used to denote a way to learn a model from a component in the context of black-box testing. It was also called later "black box checking" [5].

[4] NUPN means "Nested-Unit Petri Nets" and is additional information providing some structure to the specification [16]. Some models in the benchmark embed such information.

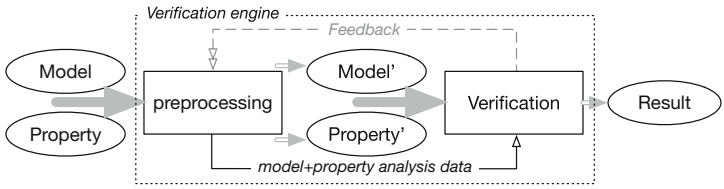


Fig. 3. The self-adaptive model checking process inside a verification engine.

The context here is totally different. The objective is to integrate some "intelligence" in tools so that they can self-adapt to the *most appropriate combination and configuration* of techniques when verifying a property on a model. In this situation, "appropriate" may have several meanings. Among them, let us consider the combinatorial state explosion problem that prevents a non-expert user from verifying a property on a system where some expert would apply a combination of modeling abstractions and the choice of the best model checking engine to solve the problem.

This combinatorial explosion problem is itself difficult to solve because it has to be tackled at different levels. It does not only involve a toolset and must be considered already at the modeling level. This is why it raises methodological issues in the way specifications and properties can be tuned to reduce complexity. Typically, relevant abstractions in the system model with regards to the properties to be verified, may dramatically reduce the verification complexity and should not be ignored.

Self-adaptive model checking must be operated at several stages inside a *verification engine* (seen as a "black box" by users) where various techniques are coordinated to build a verification process.

Figure 3 sketches what could be such a process. First, we consider as input a model and a property (both can be produced automatically or manually). A preprocessing step analyzes the model and the associated property to produce a simplified model and a simplified property, as well as some analysis data. The simplified model must be equivalent to the original one with regards to the property to be checked. Similarly, the simplified property must be equivalent to the original one. Analysis data about the model and its related property can be deduced from structural analysis (*e.g.* hierarchical design of the system, invariants or some properties when the input specification is a Petri nets, or any other information that can be derived from the specification, and possibly the property). In the worst case, the simplified model and/or property are equal to the input model and property. Let us note that, in the Model Checking Contest, such a situation is rare when models are complex (for example, those coming from an industrial case study) and tools implement such type of optimizations.

Once produced, the simplified model and property are then processed by a verification engine which can be adapted using the analysis data.

One can even imagine that a feedback from the way model and property are processed may provide useful information for some later preprocessing. Typically, this could lead to the integration of a CEGAR [9] like loop inside the verification engine itself.

4 Impact on Model Checkers Architecture

This two step process suggests an architecture that is similar to the one of modern compilers (front-end, middle-end, back-end). In fact, it is a trend to adopt this type of architecture in modern model checkers [27].

This trend is illustrated in Fig. 4. The idea is to separate the verification engine from the input formalism. Then, the notion of "pivot representation" naturally arises as an intermediate representation between an "upper level", and a a set of "verification engines" able to process this pivot representation. This software architecture brings three major advantages.

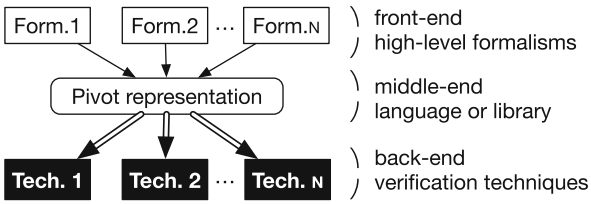


Fig. 4. Software architecture of modern model checkers (from [27]).

First, *it decouples the input (high-level) specification language from its verification*. Then, the specification language designer may work independently from the verification machinery as long as they provide a sound and formal semantics to their notation. This is of particular interest when dealing with numerous input languages, because it does not hinder the access to efficient verification engines thanks to the pivot representation.

Several tools like MC-Kit [32], LTSMin [25], Spot [14], or ITS-Tools [35] already experimented this type of approach. Some of them (like Spot or LTSMin) offer an API to encode the notion of state and the transition relation of a given model. Others (like ITS-Tools) implement an "assembly language" suitable to encode such notions. These solutions showed their efficiency in numerous situations but suffer from an important drawback when it comes to provide feedback to users : they usually lack back-translation mechanisms to show counterexamples in the terms of the input specification. This is one of the main challenge for self-adaptive model checking as soon as one wants to cope with several input formalisms.

The second advantage is that *many preprocessing optimizations can be performed*, either in the front-end (the input language specific ones) or in the middle-end (more generic ones). Optimizations implemented in the middle-end benefit to all the input formalisms supported by the model checker.

We already mentioned tools structured in a way they can easily enable analysis from various input specifications. There is unfortunately no standard pivot representation (neither at an API level nor at a language level) despite the numerous attemps to define interesting languages for verification of

industrial-like systems (such as PNML [21], FIACRE [3], GAL [35], or Promela [22]). Probably, finding a standard suitable to be associated with numerous and different techniques is an important challenge for the community.
The third advantage is that it is then possible to *exploit the technology foreseen to be the most efficient* to process the model and its related property. One can even imagine the back-end to be (automatically) produced on-the-fly from off-the-shelf libraries, assembled to build the most performant model checking engine for a given couple $\langle model, property \rangle$.

It appears from the model checking contest that most winning tools simultaneously activate several techniques to compute properties. Thus, combination of representation techniques (*i.e.* explicit, symbolic, use of different classes of automata), together with reduction algorithms (like partial order, saturation in decision diagrams, etc.) is probably needed in the future. Here, numerous challenges must be addressed to elaborate appropriate and combinable back-ends for verification.

5 The Decision Process

The decision process to select libraries to be assembled to produce an efficient model checking engine is a crucial challenge. A deeper analysis of the involved techniques and their success in identified situations (*e.g.* the use of some operator, some structure of the model or the property, etc.) is required to enable some meta-heuristic that would act as a decision process to perform the assembling of a verification engine.

However, the question of the technique to be used to implement such a meta-heuristic remains. We think that several directions should be investigated:

- An analysis of the input system and property, associated with a dictionary of techniques usable in each situation could be considered. It requires to be aware of the correlations between some characteristics of the couple $\langle model, property \rangle$ and the most efficient technique to solve problems having such characteristics. As an illustration, we can cite the definition of adapted efficient algorithms dedicated to the verification of subcategories of LTL formulas [4]. Unfortunately only a few such situations are identified so far.
- Recent learning techniques have proven their efficiency to take decisions based on the analysis of a large set of data. Unfortunately, we have no evidence that we already have a sufficient amount of unbiased data to operate such a technique.
- The increasing parallelism of modern computers allows us to imagine a portfolio-like implementation of a model checker where several algorithms would concurrently be operated. Unfortunately, such a solution requires massively parallel architectures since the number of possible combinations when mixing algorithms, representations, and implementation matters, grows rapidly.

Of course, even if the complexity of the decision algorithm is more likely to be related to the size of the system instead of the size of its associated state space, it can take a while, thus being of little interest for unsatisfied properties for which a counterexample is found rapidly. However, it is easy to imagine that such an analysis could be performed in parallel of a first search using some default configuration of the model checking engine. However, for satisfied properties, for the analysis of CTL formulas, or any other situation where the full state space has to be explored to take a decision, the cost of building on-the-fly a dedicated model checking engine would be probably rapidly balanced by the gain on the verification operation itself.

Anyway, at this stage, it is difficult to state which of these approaches will help and produce a self-adaptive model checking tool. We trust there is an interesting problem for the community to deal with.

6 Conclusion

In this paper, we depict *self-adaptive model checking* as a way to increase the efficiency of model checking. It is a mix of methodological, theoretical, and technical visions. Methodological aspects reside in the definition of a typical process including preprocessing (a way to rewrite and simplify the problem) and the use of optimized verification engines, possibly elaborated and compiled on-the-fly for a given couple $\langle model, property \rangle$. Theoretical aspects reside in the fact that theory needs to be extended, for example to enable the combination of several algorithms when it is possible. Technical aspects reside in the definition of some standards (a common pivot representation, a common software architecture, etc.) to enable the sharing of off-the-shelf efficient libraries.

It would also be of interest to share and increase typical benchmarks so that the effect of some algorithm combinations could be explored deeply. Then, more lessons could be gathered from larger experiments on these benchmarks. At this stage, the Model Checking Contest [28] can provide interesting data based on the analysis of the results available for 2015, 2016 and 2017. A first and raw analysis of these data is discussed in the paper. Getting similar information from other similar events should be a goal for the communities involved.

Of course, much work is still needed to complete fully automated self-adaptive model checking. However, the community of model checking already handles many building blocks for this purpose: numerous algorithms, numerous internal representations of the state space (symbolic or explicit, based on a variety of automata), various logics, etc.

So, one can expects that, sooner or later, a new generation of model checking tools will emerge, benefitting from all these expertise, implementation experience, and experimentation.

Self-adaptive Model Checking is a long-term goal the community should take as an important challenge to deal with.

References

1. Baarir, S., Duret-Lutz, A.: Sat-based minimization of deterministic ω-automata. In: 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR, pp. 79–87 (2015)
2. Ben Salem, A.E., Duret-Lutz, A., Kordon, F., Thierry-Mieg, Y.: Symbolic model checking of stutter-invariant properties using generalized testing automata. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 440–454. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_38
3. Berthomieux, B., Bodeveix, J.P., Filali, M., Lang, F., Le Botland, D., Vernadat, F.: The syntax and semantic of fiacre. Technical report 7264, CNRS-LAAS (2007)
4. Bloem, R., Ravi, K., Somenzi, F.: Efficient decision procedures for model checking of linear time logic properties. In: Halbwachs, N., Peled, D. (eds.) CAV 1999. LNCS, vol. 1633, pp. 222–235. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48683-6_21
5. Broy, M., Jonsson, B., Katoen, J., Leucker, M., Pretschner, A. (eds.): Model-Based Testing of Reactive Systems. LNCS, vol. 3472. Springer, Heidelberg (2005). https://doi.org/10.1007/b137241
6. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Congress on Logic, Method, and Philosophy of Science, pp. 1–12. Stanford University (1960, 1962)
7. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: $10\hat{2}0$ states and beyond. Inf. Comput. **98**(2), 142–170 (1992)
8. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: Stochastic well-formed colored nets and symmetric modeling applications. IEEE Trans. Comput. **42**(11), 1343–1360 (1993)
9. Clarke, E.M., Fehnker, A., Han, Z., Krogh, B.H., Ouaknine, J., Stursberg, O., Theobald, M.: Abstraction and counterexample-guided refinement in model checking of hybrid systems. Int. J. Found. Comput. Sci. **14**(4), 583–604 (2003)
10. Clarke, E.M., Jha, S., Marrero, W.R.: Efficient verification of security protocols using partial-order reductions. STTT **4**(2), 173–188 (2003)
11. Colange, M., Baarir, S., Kordon, F., Thierry-Mieg, Y.: Towards distributed software model-checking using decision diagrams. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 830–845. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_58
12. Duflot, M., Kwiatkowska, M.Z., Norman, G., Parker, D.: A formal analysis of bluetooth device discovery. STTT **8**(6), 621–632 (2006)
13. Duret-Lutz, A., Klai, K., Poitrenaud, D., Thierry-Mieg, Y.: Self-loop aggregation product—a new hybrid approach to on-the-fly LTL model checking. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 336–350. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24372-1_24
14. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, É., Xu, L.: Spot 2.0—a framework for LTL and ω-automata manipulation. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 122–129. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_8
15. Evangelista, S., Haddad, S., Pradat-Peyre, J.: Syntactical colored petri nets reductions. In: Automated Technology for Verification and Analysis, Third International Symposium, ATVA. pp. 202–216 (2005)

16. Garavel, H.: Nested-unit petri nets: a structural means to increase efficiency and scalability of verification on elementary nets. In: Devillers, R., Valmari, A. (eds.) PETRI NETS 2015. LNCS, vol. 9115, pp. 179–199. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19488-2_9
17. Geldenhuys, J., Hansen, H.: Larger automata and less work for LTL model checking. In: Valmari, A. (ed.) SPIN 2006. LNCS, vol. 3925, pp. 53–70. Springer, Heidelberg (2006). https://doi.org/10.1007/11691617_4
18. Gerth, R.: Model checking if your life depends on it: a view from intel's trenches. In: Dwyer, M. (ed.) SPIN 2001. LNCS, vol. 2057, p. 15. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45139-0_2
19. Groce, A., Peled, D., Yannakakis, M.: Adaptive model checking. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 357–370. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46002-0_25
20. Hamez, A., Thierry-Mieg, Y., Kordon, F.: Building efficient model checkers using hierarchical set decision diagrams and automatic saturation. Fundam. Inf. **94**(3–4), 413–437 (2009)
21. Hillah, L., Kindler, E., Kordon, F., Petrucci, L., Trèves, N.: A primer on the Petri Net Markup Language and ISO/IEC 15909–2. In: Petri Net Newsletter (originally presented at the 10th International workshop on Practical Use of Colored Petri Nets and the CPN Tools - CPN 2009), vol. 76, pp. 9–28 (2009)
22. Holzmann, G.: The Spin Model Checker: Primer and Reference Manual, 1st edn. Addison-Wesley Professional, Boston (2003)
23. Holzmann, G.J.: Mars code. Commun. ACM **57**(2), 64–73 (2014)
24. Hugues, J., Thierry-Mieg, Y., Kordon, F., Pautet, L., Baarir, S., Vergnaud, T.: On the formal verification of middleware behavioral properties. In: 9th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2004), pp. 139–157. Elsevier (2004)
25. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: high-performance language-independent model checking. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 692–707. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_61
26. Klai, K., Poitrenaud, D.: MC-SOG: an LTL model checker based on symbolic observation graphs. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS, vol. 5062, pp. 288–306. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68746-7_20
27. Kordon, F., Leuschel, M., van de Pol, J., Thierry-Mieg, Y.: Software architecture of modern model checkers. In: High Assurance System: Methods, Languages, and Tools. LNCS 10000 (2018, to appear)
28. Kordon, F., Garavel, H., Hillah, L.M., Paviot-Adet, E., Jezequel, L., Rodríguez, C., Hulin-Hubard, F.: MCC'2015 – the fifth model checking contest. In: Koutny, M., Desel, J., Kleijn, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency XI. LNCS, vol. 9930, pp. 262–273. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53401-4_12
29. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: von Bochmann, G., Probst, D.K. (eds.) CAV 1992. LNCS, vol. 663, pp. 164–177. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-56496-9_14
30. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. AMS **141**, 1–35 (1969)

31. Renault, E., Duret-Lutz, A., Kordon, F., Poitrenaud, D.: Strength-based decomposition of the property Büchi automaton for faster model checking. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 580–593. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36742-7_42
32. Schröter, C., Schwoon, S., Esparza, J.: The model-checking kit. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 463–472. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44919-1_29
33. Schwarick, M., Heiner, M.: CSL model checking of biochemical networks with interval decision diagrams. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 296–312. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03845-7_20
34. Streett, R.S.: Propositional dynamic logic of looping and converse is elementarily decidable. Inf. Control **54**(1/2), 121–141 (1982)
35. Thierry-Mieg, Y.: Symbolic model-checking using ITS-tools. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 231–237. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_20
36. Wang, F., Schmidt, K., Yu, F., Huang, G., Wang, B.: BDD-based safety-analysis of concurrent software with pointer data structures using graph automorphism symmetry reduction. IEEE Trans. Softw. Eng. **30**(6), 403–417 (2004)

Petri Net Synthesis

Analysis and Synthesis of Weighted Marked Graph Petri Nets

Raymond Devillers[1] and Thomas Hujsa[2(✉)]

[1] Département d'Informatique, Université Libre de Bruxelles, 1050 Brussels, Belgium
rdevil@ulb.ac.be

[2] Department of Computing Science, Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany
hujsa.thomas@gmail.com

Abstract. Numerous real-world systems can be modeled with Petri nets, which allow a combination of concurrency with synchronizations and conflicts. To alleviate the difficulty of checking their behaviour, a common approach consists in studying specific subclasses. In the converse problem of Petri net synthesis, a Petri net of some subclass has to be constructed efficiently from a given specification, typically from a labelled transition system describing the behaviour of the desired net.

In this paper, we focus on a notorious subclass of persistent Petri nets, the weighted marked graphs (WMGs), also called generalised (or weighted) event (or marked) graphs or weighted T-nets. In such nets, edges have multiplicities (weights) and each place has at most one ingoing and one outgoing transition. Although extensively studied in previous works and benefiting from strong results, both their analysis and synthesis can be further investigated. To this end, we provide new conditions delineating more precisely their behaviour and give a dedicated synthesis procedure.

Keywords: Weighted Petri net · Analysis · Synthesis
Marked graph · Event graph

1 Introduction

Petri nets have proved useful to model numerous artificial and natural systems. Their weighted version allows weights on arcs, making possible the bulk consumption or production of tokens, hence a more compact representation of the systems.

Many fundamental properties of Petri nets are decidable, although often hard to check. Given a bounded Petri net, a naive analysis can be performed by constructing its finite reachability graph, whose size may be considerably larger than the net size. To avoid such a costly computation, subclasses are often considered, allowing to derive efficiently their behaviour from their structure only.

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 19–39, 2018.
https://doi.org/10.1007/978-3-319-91268-4_2

This approach has led to various polynomial-time checking methods dedicated to several subclasses, the latter being defined by structural restrictions in many cases [13,17,18,24,28].

In the domain of Petri net synthesis, a specification has to be implemented by a Petri net, meaning that the behaviour of the Petri net obtained must correspond exactly to the specification. Classical representations of such a specification encompass labelled transitions systems (lts for short), which are rooted directed graphs with labels on the arcs, and a synthesis procedure is meant to build a Petri net of a specific subclass whose reachability graph is isomorphic to a given lts.

Weighted Marked Graphs: Applications and Previous Studies. In this paper, we focus on marked graphs with weights (also called generalised event graphs and weighted T-nets), a subclass of weighted Petri nets in which each place has at most one input and one output. They can model Synchronous DataFlow graphs [21], which have been fruitfully used to design and analyse many real systems such as embedded applications, notably Digital Signal Processing (DSP) applications [20,23,25].

Various characterisations and polynomial-time sufficient conditions of structural and behavioural properties, notably of liveness, boundedness and reversibility, have been developed for this class [22,26]. These nets are a special case of persistent systems, in which no transition firing can disable another transition.

Petri Net Synthesis: Previous Studies. Given a labelled transition system, previous works have proposed algorithms synthesizing a Petri net with an isomorphic reachability graph, sometimes aiming at a Petri net subclass [6,9]. In the latter case, the objective is to delineate properties of the lts that are specific to the target subclass, so as to determine sufficient and necessary conditions for its synthesisability within the subclass. Ideally, such specific conditions should be easier to check than generic ones, for instance during a pre-synthesis phase.

Marked graphs, i.e. unit-weighted marked graphs, belong to the larger class of choice-free nets, in which each place has at most one output. Both classes benefit from dedicated synthesis algorithms that operate in polynomial time [2,4,6,8,9]. However, such methods do not yet exist for the intermediate class of marked graphs with arbitrary weights.

Contributions. In this paper, we further investigate the class of weighted marked graphs (WMGs). We delineate new properties of these nets and propose a synthesis procedure aiming at this subclass.

First, we provide new structural and behavioural properties of WMGs: we give a comparison property on the sequences starting at the same state and reaching another common state, we show that WMGs are necessarily *backward persistent*, meaning that for all reachable states s_1, s_2, s_3 such that $s_2[a\rangle s_1$ (i.e. s_1 is reached from s_2 through the action with label a) and $s_3[b\rangle s_1$, there exists a reachable state s_4 with $s_4[b\rangle s_2$ and $s_4[a\rangle s_3$. We also develop conditions allowing the existence of a feasible sequence corresponding to a given Parikh vector.

Then, we delineate necessary conditions for the WMG-solvability of an lts, such as backward persistence and the existence of particular cycles. We show, with the help of a counter-example from another subclass, that these conditions are not sufficient for a WMG solution to exist.

Finally, we devise a WMG-synthesis procedure, specialising previous methods that were designed for the larger class of choice-free nets.

Organisation of the Paper. In Sect. 2, we introduce general definitions, notations and properties. In Sect. 3, we recall some properties of persistent Petri nets and provide new structural and behavioural results on WMGs, including the proof of backward persistence. In Sect. 4, we describe a synthesis procedure for WMGs. Section 5 presents our conclusion with perspectives.

2 Classical Definitions, Notations and Properties

In the following, we define formally Petri nets, labelled transitions systems and related notions. We also recall classical properties of Petri nets in Proposition 1.

Petri Nets, Incidence Matrices, Pre- and Post-sets. A *(Petri) net* is a tuple $N = (P, T, W)$ such that P is a finite set of *places*, T is a finite set of *transitions*, with $P \cap T = \emptyset$, and W is a weight function $W : ((P \times T) \cup (T \times P)) \to \mathbb{N}$ setting the weights on the arcs. A *marking* of the net N is a mapping from P to $\mathbb{N}$, i.e. a member of $\mathbb{N}^P$, defining the number of tokens in each place of N.

A *(Petri net) system* is a tuple $\zeta = (N, M_0)$ where N is a net and M_0 is a marking, often called *initial marking*. The *incidence matrix* C of N (and ζ) is the integer place-transition matrix with components $C(p,t) = W(t,p) - W(p,t)$, for each place p and each transition t.
The *post-set* $n^\bullet$ and *pre-set* ${}^\bullet n$ of a node $n \in P \cup T$ are defined as $n^\bullet = \{n' \in P \cup T \mid W(n,n') > 0\}$ and ${}^\bullet n = \{n' \in P \cup T \mid W(n',n) > 0\}$.

Firings and Reachability in Petri Nets. Consider a system $\zeta = (N, M_0)$ with $N = (P, T, W)$. A transition t is *enabled* at M_0 (i.e. in ζ) if $\forall p \in {}^\bullet t$, $M_0(p) \geq W(p,t)$, in which case t can *occur* at or be *fired* from M_0. The firing of t from M_0 leads to the marking $M = M_0 + C[P,t]$ where $C[P,t]$ is the column of C associated to t: we note this as $M_0[t\rangle M$.
A finite *(firing) sequence* σ of length $n \geq 0$ on the set T, denoted by $\sigma = t_1 \dots t_n$ with $t_1 \dots t_n \in T$, is a mapping $\{1, \dots, n\} \to T$. Infinite sequences are defined similarly as mappings $\mathbb{N} \setminus \{0\} \to T$. A sequence σ of length n is *enabled* in ζ if the successive states obtained, $M_0[t_1\rangle M_1 \dots [t_n\rangle M_n$, satisfy $M_{k-1}[t_k\rangle M_k$, $\forall k \in \{1, \dots, n\}$, in which case M_n is said to be *reachable* from M_0: we note this as $M_0[\sigma\rangle M_n$. If $n = 0$, σ is the *empty sequence* ϵ, implying $M_0[\epsilon\rangle M_0$. The set of markings reachable from M_0 is noted $[M_0\rangle$.
The *reachability graph* of ζ, noted $RG(\zeta)$, is the rooted directed graph (V, A, ι) where V represents the set of vertices $[M_0\rangle$, A is the set of arcs labelled with transitions of T such that the arc $M \xrightarrow{t} M'$ belongs to A if and only if $M[t\rangle M'$ and $M \in [M_0\rangle$, and $\iota = M_0$ is the root.

In Fig. 1, a weighted system is pictured on the left. Its reachability graph is pictured on the right, where v^T denotes the transpose of vector v.

Petri Net Subclasses. N is *plain* if no arc weight exceeds 1; *choice-free* (CF for short) [11,27] (also called *place-output-nonbranching* in [5]) if $\forall p \in P$, $|p^\bullet| \leq 1$; *fork-attribution* (FA) [27] if it is CF and, in addition, $\forall t \in T$, $|^\bullet t| \leq 1$; a *weighted marked graph* (WMG, also called weighted T-system in [26]) if it is CF and, in addition, $\forall p \in P$, $|^\bullet p| \leq 1$. A WMG is pictured on the left of Fig. 1. Well-studied subclasses encompass *marked graphs* [10], which are plain and fulfill $|p^\bullet| = 1$ and $|^\bullet p| = 1$ for each place p, and *T-systems* [13], which are plain and fulfill $|p^\bullet| \leq 1$ and $|^\bullet p| \leq 1$ for each place p.

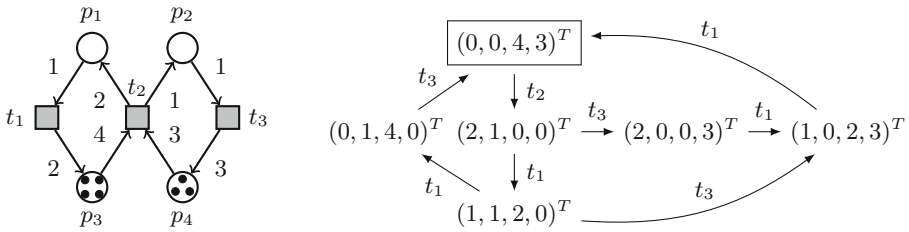


Fig. 1. A WMG system ζ and its reachability graph $RG(\zeta)$ are pictured respectively on the left and on the right. The initial marking is boxed in $RG(\zeta)$.

Lts and Their Relationship with Petri Nets. A *labelled transition system with initial state*, abbreviated *lts*, is a quadruple $TS = (S, \rightarrow, T, \iota)$ where S is the set of *states*, T is the set of *labels*, $\rightarrow \subseteq (S \times T \times S)$ is the *transition relation*, and $\iota \in S$ is the *initial state*.
A label t is *enabled* at $s \in S$ if $\exists s' \in S\colon (s,t,s') \in \rightarrow$, written $s[t\rangle$ or $s[t\rangle s'$, in which case s' is *reachable* from s through the execution of t. We denote by $s^\bullet$ the set $\{s' | \exists t \in T, s[t\rangle s'\}$.
A label t is *backward enabled* at s if $\exists s' \in S\colon (s',t,s) \in \rightarrow$, written $[t\rangle s$ or $s'[t\rangle s$.
A *(firing) sequence* σ of length $n \geq 0$ on the set of labels T, denoted by $\sigma = t_1 \dots t_n$ with $t_1 \dots t_n \in T$, is *enabled* at some state s_0 if the successive states obtained, $s_0[t_1\rangle s_1 \dots [t_n\rangle s_n$, satisfy $s_{k-1}[t_{i_k}\rangle s_k$, $\forall k \in \{1, \dots, n\}$: we note $s_0[\sigma\rangle s_n$. Similarly, other notions and notations, related to sequences and reachability in Petri nets, extend readily to labelled transition systems by replacing markings with states.
The reachability graph $RG(\zeta)$ of a system $\zeta = (N, M_0)$ can be represented by the labelled transition system $TS = (S, \rightarrow, T, \iota)$ if an isomorphism $\gamma\colon S \rightarrow [M_0\rangle$ exists such that $\gamma(\iota) = M_0$ and $(s,t,s') \in \rightarrow \Leftrightarrow \gamma(s)[t\rangle\gamma(s')$ for all $s, s' \in S$. If an lts TS is isomorphic to the reachability graph of a Petri net system ζ, we say that ζ *solves* TS, and that it WMG-solves TS if N is a WMG.
These notions are illustrated on the lts on the right of Fig. 2, which is isomorphic to the reachability graph of the WMG in Fig. 1; it is thus WMG-solvable.

Two lts $TS_1 = (S_1, \rightarrow_1, T, s_{01})$ and $TS_2 = (S_2, \rightarrow_2, T, s_{02})$ are isomorphic if there is a bijection $\beta\colon S_1 \rightarrow S_2$ with $\beta(s_{01}) = s_{02}$ and $(s,t,s') \in \rightarrow_1 \Leftrightarrow (\beta(s), t, \beta(s')) \in \rightarrow_2$, for all $s, s' \in S_1$.

Vectors, Semiflows and Cycles. The *support* of a vector is the set of the indices of its non-null components. Consider any net $N = (P, T, W)$ with its incidence matrix C. A *T-vector* is an element of $\mathbb{N}^T$; it is called *prime* if the greatest common divisor of its components is one (i.e. its components do not have a common non-unit factor). A *T-semiflow* ν of the net is a non-null T-vector whose components are only non-negative integers (i.e. $\nu \gneqq \mathbb{0}$) and such that $C \cdot \nu = \mathbb{0}$. A T-semiflow is called *minimal* when it is prime and its support is not a proper superset of the support of any other T-semiflow [27].
The *Parikh vector* $\mathbf{P}(\sigma)$ of a finite sequence σ of transitions is a T-vector counting the number of occurrences of each transition in σ, and the *support* of σ is the support of its Parikh vector, i.e. $supp(\sigma) = supp(\mathbf{P}(\sigma)) = \{t \in T \mid \mathbf{P}(\sigma)(t) > 0\}$. A (non-empty) cycle around a marking M is a non-empty sequence σ such that $M[\sigma\rangle M$; the Parikh vector of a non-empty cycle is a T-semiflow and a non-empty cycle is called prime if its Parikh vector is prime.

Further Notions. Consider a lts $TS = (S, \rightarrow, T, \iota)$. For all states $s, s' \in S$, a sequence $s[\sigma\rangle s'$ is called a *cycle*, or more precisely a *cycle at (or around) state* s, if $s = s'$. A non-empty cycle $s[\sigma\rangle s$ is called *small* if there is no non-empty cycle $s'[\sigma'\rangle s'$ in TS with $\mathbf{P}(\sigma') \lneqq \mathbf{P}(\sigma)$. A *two-way uniform chain* of TS is a couple $(\{s_i \in S | i \in \mathbb{Z}, \forall i, j \in \mathbb{Z} : i \neq j \Rightarrow s_i \neq s_j\}, \sigma \in T^+)$ such that $\forall i \in \mathbb{Z}, s_i[\sigma\rangle s_{i+1}$, where T^+ is the set of non-empty sequences on T.

In Fig. 2, a two-way uniform chain is depicted on the left; on the right, the lts is finite, hence has no two-way uniform chain. The lts TS is:

- *totally reachable* if $S = [\iota\rangle$;
- *reversible* if $\iota \in [s\rangle$ for each state $s \in [\iota\rangle$, meaning the strong connectedness of this lts when it is totally reachable;
- *weakly periodic* if for each couple $(\{s_i \in S | i \in \mathbb{N}\}, \sigma \in T^+)$ such that $\forall i \in \mathbb{N}$ $s_i[\sigma\rangle s_{i+1}$ (where σ is a non-empty sequence of labels), either $s_i = s_j\ \forall i, j \in \mathbb{N}$, or $i \neq j \Rightarrow s_i \neq s_j\ \forall i, j \in \mathbb{N}$;
- *strongly cycle consistent* if for every sequence $s[\alpha\rangle s'$, the existence of cycles $s_1[\beta_1\rangle s_1$, $s_2[\beta_2\rangle s_2$, ..., $s_n[\beta_n\rangle s_n$ and of numbers $k_1, k_2, \ldots, k_n \in \mathbb{Q}$ such that $\mathbf{P}(\alpha) = \sum_{i=1}^{n} k_i \cdot \mathbf{P}(\beta_i)$ implies that $s = s'$;
- *deterministic* if, for all states $s, s', s'' \in S$ and labels $t, t' \in T$ such that $s[t\rangle s' \wedge s[t\rangle s''$, necessarily $s' = s''$; it is *fully deterministic* if for all sequences σ and σ' such that $\mathbf{P}(\sigma) = \mathbf{P}(\sigma')$, we have, for all states $s, s', s'' \in S$: $s[\sigma\rangle s' \wedge s[\sigma'\rangle s'' \Rightarrow s' = s''$;
- *backward deterministic* if, for all states $s, s', s'' \in S$ and labels $t, t' \in T$ such that $s'[t\rangle s \wedge s''[t\rangle s$, necessarily $s' = s''$; it is *fully backward deterministic* if, for all sequences σ and σ' such that $\mathbf{P}(\sigma) = \mathbf{P}(\sigma')$, we have, for all states $s, s', s'' \in S$: $s'[\sigma\rangle s \wedge s''[\sigma'\rangle s \Rightarrow s' = s''$;
- *persistent* if for all states $s, s', s'' \in S$ and labels $t', t'' \in T$ such that $s[t'\rangle s'$ and $s[t''\rangle s''$ with $t' \neq t''$, there exists a state $s''' \in S$ such that $s'[t''\rangle s'''$

and $s''[t'\rangle s'''$; it is *backward persistent* if for all states $s, s', s'' \in S$ and labels $t', t'' \in T$ such that $s'[t'\rangle s$ and $s''[t''\rangle s$ with $t' \neq t''$, there exists a state $s''' \in S$ such that $s'''[t''\rangle s'$ and $s'''[t'\rangle s''$.

Figure 2 illustrates some of these notions. All notions defined for labelled transition systems apply to Petri nets through their reachability graphs. For example, a Petri net is reversible if its reachability graph is isomorphic to a reversible lts, meaning that the initial marking is reachable from every reachable marking.

Fig. 2. On the left, a two-way uniform chain based on σ. On the right, a labelled transition system with states $\{s_0, s_1, s_2, s_3, s_4, s_5\}$, labels $\{t_1, t_2, t_3\}$ and initial state $\iota = s_0$. It is isomorphic to the reachability graph of Fig. 1. The label t_2 is enabled at s_0 and t_3 is backward enabled at s_0. The state s_1 is reachable from s_0 through the execution of t_2. Denote by σ the sequence $t_2t_3t_1t_1$. Then, the Parikh vector of σ is $\mathbf{P}(\sigma) = (2, 1, 1)$ and its support is $supp(\sigma) = \{t_1, t_2, t_3\}$. Since $s_0[\sigma\rangle s_0$, σ is a cycle around state s_0. This lts is totally reachable, weakly periodic, fully deterministic and fully backward deterministic, strongly cycle consistent, persistent, backward persistent and reversible.

The following proposition recalls properties satisfied by every Petri net system and presented in [5].

Proposition 1 (Classical properties of Petri nets [5]). *If $\zeta = (N, M_0)$, where $N = (P, T, W)$, is a Petri net system, then $RG(\zeta)$ is totally reachable, weakly periodic, fully deterministic, fully backward deterministic, and strongly cycle consistent. Moreover it has no two-way uniform chain over the set $S = \mathbb{N}^P$ of all the possible markings for N, meaning that no couple $(\{M_i \in \mathbb{N}^P | i \in \mathbb{Z}\backslash\{0\}, \forall i, j \in \mathbb{Z} : i \neq j \Rightarrow s_i \neq s_j\}, \sigma \in T^+)$ exists such that $\forall i \in \mathbb{Z}$, $M_i[\sigma\rangle M_{i+1}$.*

3 Properties of WMGs and Larger Persistent Classes

In this section, we investigate the structure and behaviour of WMGs. For that purpose, we first recall notions and results relevant to persistent systems in Subsect. 3.1. Then, in Subsect. 3.2, for the class of WMGs, we show a property of the sequences sharing the same starting state and the same ending state, and we prove backward persistence. Finally, in Subsect. 3.3, we propose conditions for the existence of feasible sequences corresponding to a given T-vector in WMGs.

3.1 Previous Results and Notions Related to Persistence

In addition to the general properties of Petri nets mentioned in Proposition 1, we recall results and notions useful to the study of persistent systems.

The next result is dedicated to WMGs and extracted from [26,27].

Proposition 2 (Minimal T-semiflow and cycles in WMGs [26,27]). *Consider a connected WMG net N. If N has a T-semiflow ν then there exists a unique minimal (hence prime) one π, which satisfies: $supp(\pi) = T$ and $\nu = k \cdot \pi$ for some integer $k > 0$. Moreover, for any marking M_0, writing $\zeta = (N, M_0)$, if $RG(\zeta)$ contains some non-empty cycle, then the Parikh vector of each small cycle of $RG(\zeta)$ equals π.*

The next notion of residues is useful to the study of persistent systems.

Definition 1 (Residues). *Let T be a set of labels and $\tau, \sigma \in T^*$ two sequences over this set. The* (left) residue of τ with respect to σ, *denoted by $\tau \overset{\bullet}{-} \sigma$, arises from cancelling successively in τ the leftmost occurrences of all symbols from σ, read from left to right. Inductively: $\tau \overset{\bullet}{-} \varepsilon = \tau$; $\tau \overset{\bullet}{-} t = \tau$ if $t \notin supp(\tau)$; $\tau \overset{\bullet}{-} t$ is the sequence obtained by erasing the leftmost t in τ if $t \in supp(\tau)$; and $\tau \overset{\bullet}{-} (t\sigma) = (\tau \overset{\bullet}{-} t) \overset{\bullet}{-} \sigma$. For example, $acbcacbc \overset{\bullet}{-} abbcb = cacc$ and $abbcb \overset{\bullet}{-} acbcacbc = b$.*

We deduce the next property of residues.

Lemma 1 (Disjoint support of residues). *For any two sequences τ and σ, the residues $\delta_1 = \tau \overset{\bullet}{-} \sigma$ and $\delta_2 = \sigma \overset{\bullet}{-} \tau$ have disjoint supports: $supp(\delta_1) \cap supp(\delta_2) = \emptyset$. Consequently, $\delta_1 \overset{\bullet}{-} \delta_2 = \delta_1$ and $\delta_2 \overset{\bullet}{-} \delta_1 = \delta_2$.*

Proof. For any label t, $\mathbf{P}(\tau)(t) = \mathbf{P}(\sigma)(t) \Rightarrow \mathbf{P}(\delta_1)(t) = \mathbf{P}(\delta_2)(t) = 0$, $\mathbf{P}(\tau)(t) > \mathbf{P}(\sigma)(t) \Rightarrow \mathbf{P}(\delta_2)(t) = 0$ and $\mathbf{P}(\tau)(t) < \mathbf{P}(\sigma)(t) \Rightarrow \mathbf{P}(\delta_1)(t) = 0$. In all cases, $t \notin supp(\delta_1) \cap supp(\delta_2)$. □

Kellers's theorem is based on residues and applies to persistent lts.

Theorem 1 (Keller [19]). *Let $(S, \to, T, \iota)$ be a deterministic, persistent lts. Let τ and σ be two label sequences enabled at some state s. Then $\tau(\sigma \overset{\bullet}{-} \tau)$ and $\sigma(\tau \overset{\bullet}{-} \sigma)$ are both enabled at s and lead to the same state.*

Applying Theorem 1, we obtain the next result directly.

Proposition 3 (Persistence and determinism). *Let TS be a persistent lts. If TS is also deterministic, then it is fully deterministic.*

3.2 Equivalent Sequences and Backward Persistence

In the following, we provide new properties on the reachability graph of WMGs. Since non-connected nets can be studied by analysing each connected component separately, we restrict our attention to connected nets.

For the class of WMGs, we first provide in Lemma 2 a property of the sequences starting from a same state s and leading to the same state s'. Then, we prove the backward persistence of WMGs in Theorem 2. To achieve it, we need to define the reverse of a net and of a firing sequence.

Definition 2 (Reverse nets and sequences). *The* reverse *of a net N, denoted by $-N$, is obtained from N by reversing all the arcs while keeping the weights. We denote by σ^{-1} the sequence σ followed in reverse order. For example, if $\sigma = t_1t_2t_2t_3$, then $\sigma^{-1} = t_3t_2t_2t_1$.*

The set of WMGs is closed under reverse, contrarily to the set of CF nets.

Lemma 2 highlights strong similarities in the reachability graph between two sequences sharing the same starting state and the same destination state. The proof makes use of reverse sequences feasible in reverse WMGs.

Lemma 2 (Equivalent sequences in WMGs). *Let N be a connected WMG. Assume the existence of markings M, M_1 and sequences σ, σ' such that $M[\sigma\rangle M_1$ and $M[\sigma'\rangle M_1$. If N has no T-semiflow, then $\mathbf{P}(\sigma) = \mathbf{P}(\sigma')$. Otherwise, either $\mathbf{P}(\sigma) = \mathbf{P}(\sigma')$, or there exists an integer $k > 0$ such that $\mathbf{P}(\sigma) = \mathbf{P}(\sigma') + k.\pi$ or $\mathbf{P}(\sigma) + k.\pi = \mathbf{P}(\sigma')$, where π is the unique minimal T-semiflow of N.*

Proof. Let us assume that $\mathbf{P}(\sigma) \neq \mathbf{P}(\sigma')$. We show in the following that N has necessarily a T-semiflow in this case, proving the first claim by contraposition. Since N is a WMG, it is persistent. Defining $\tau = \sigma \overset{\bullet}{-} \sigma'$ and $\tau' = \sigma' \overset{\bullet}{-} \sigma$, applying Keller's theorem (Theorem 1), we have for some marking M_2 that $M_1[\tau\rangle M_2$ and $M_1[\tau'\rangle M_2$. By Lemma 1, τ and τ' have disjoint supports, $\tau \overset{\bullet}{-} \tau' = \tau$ and $\tau' \overset{\bullet}{-} \tau = \tau'$. Thus, applying Keller's theorem, a marking M_3 is reached from M_2 by firing τ or τ'. Iterating this process up to any positive integer i, some marking M_{i+1} is reached from M_i with $M_i[\tau\rangle M_{i+1}$ and $M_i[\tau'\rangle M_{i+1}$.

Now, in the reverse net $-N$, which is also a WMG, since $M_2[(\tau)^{-1}\rangle M_1$ and $M_2[(\tau')^{-1}\rangle M_1$, still with disjoint supports, we can construct markings M_0, M_{-1}, ... such that $\forall i \in \mathbb{Z}$, $M_i[(\tau)^{-1}\rangle M_{i-1}$ and $M_i[(\tau')^{-1}\rangle\ M_{i-1}$, i.e. also $M_{i-1}[\tau\rangle M_i$ and $M_{i-1}[\tau'\rangle M_i$. If all M_i's are (pairwisely) different, this leads to a two-way uniform chain for the system (N, M_1), contradicting Proposition 1. Consequently, for some $i, j \in \mathbb{Z}$ with $i \neq j$, we have $M_i = M_j$, and since σ, σ' are different, they are not both empty and τ, τ' cannot be both empty. Thus, N has a T-semiflow, proving the first claim of the lemma.

For the second claim, either $\mathbf{P}(\sigma) = \mathbf{P}(\sigma')$ or $\mathbf{P}(\sigma) \neq \mathbf{P}(\sigma')$. Consider the latter case: from the first part of the proof, taking the same notation, there is a positive integer n such that τ^n and τ'^n are cycles appearing in the reachability graph of the system (N, M). Since the supports of τ and τ' are not both empty, Proposition 2 applies: there is a unique minimal T-semiflow π, whose support is T, and integers $k, k' \geq 0$ exist such that $\mathbf{P}(\tau^n) = k \cdot \pi$ and $\mathbf{P}(\tau'^n) = k' \cdot \pi$, where $k > 0$ or $k' > 0$. Since the supports of τ and τ' are disjoint and the support of any cycle is T, then either $k' = 0$, $\tau' = \epsilon$, τ is a cycle and $\mathbf{P}(\sigma) \gneqq \mathbf{P}(\sigma')$, or $k = 0$, $\tau = \epsilon$, τ' is a cycle and $\mathbf{P}(\sigma') \gneqq \mathbf{P}(\sigma)$. Thus, either $\mathbf{P}(\sigma) = \mathbf{P}(\sigma') + \mathbf{P}(\tau) = \mathbf{P}(\sigma') + q.\pi$ for an integer $q > 0$ or $\mathbf{P}(\sigma') = \mathbf{P}(\sigma) + \mathbf{P}(\tau') = \mathbf{P}(\sigma) + q.\pi$ for an integer $q > 0$. Hence the claim. □

In [26], in the proof of Theorem 4.8, it is mentioned that each WMG is backward persistent without a proof, basing on the fact that the reverse of a WMG is still a WMG, hence a persistent net. However, this property needs to

be proved carefully: since $M_1[a\rangle M$ and $M_2[b\rangle M$, Keller's theorem implies the existence of a marking M' reachable from $(-N, M_1)$ and $(-N, M_2)$, such that $M'[a\rangle M_2$ and $M'[b\rangle M_1$ in the original system; however, the reachability of M' in the original system, under the assumption of reachability for M_1 and M_2, is not obvious. In the following, we show it is indeed the case.

Theorem 2 (Backward persistence of WMGs). *Backward persistence of WMGs In a connected WMG system $\zeta = (N, M_0)$, let us assume that markings M_1, M_2, M are reachable and that, for two different labels a and b, $M_1[a\rangle M$ and $M_2[b\rangle M$. Then, a marking M' is reachable in ζ such that $M'[a\rangle M_2$ and $M'[b\rangle M_1$.*

Proof. Let us write $N = (P, T, W)$ and introduce two sequences σ_1 and σ_2 enabled in ζ such that $M_0[\sigma_1\rangle M_1$ and $M_0[\sigma_2\rangle M_2$. From the previous remarks, we know that M' is reachable in the reverse system $(-N, M_1)$, hence belongs to $\mathbb{N}^P$. It remains to show that $M' \in [\zeta\rangle$.

From Lemma 2, either $\mathbf{P}(\sigma_1 a) = \mathbf{P}(\sigma_2 b)$ or, without loss of generality, $\mathbf{P}(\sigma_1 a) = \mathbf{P}(\sigma_2 b) + k \cdot \pi$, where π is the unique minimal T-semiflow of N, with support T. Then, in either case, b occurs at least once in σ_1. For the reverse net $-N$, we have $M_1[\sigma_1^{-1}\rangle M_0$, and from Keller's theorem, we have $M_1[b\rangle M'$ and $M_2[a\rangle M'$; we also have $M_0[b^\bullet\!-\!\sigma_1^{-1}\rangle M''$ and $M'[\sigma_1^{-1}{}^\bullet\!-\!b\rangle M''$. Since b occurs in σ_1, hence also in σ_1^{-1}, $b^\bullet\!-\!\sigma_1^{-1} = \epsilon$ and $M'' = M_0$. Going back to ζ, we deduce that $M_0[(\sigma_1^{-1}{}^\bullet\!-\!b)^{-1}\rangle M'$, thus M' is reachable in ζ (Fig. 3). □

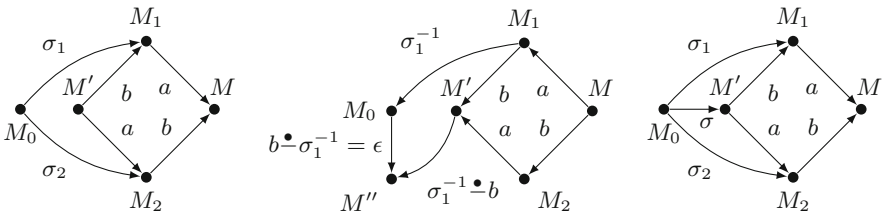


Fig. 3. Illustration of the proof of Theorem 2: the initial assumptions are depicted on the left, the sequences in the reverse system $(-N, M)$ are depicted in the middle, where the sequence leading to M'' from M_0 equals ϵ, implying that $M'' = M_0$. In the original system, we deduce the reachability of M' from M_0 on the right, with $\sigma = (\sigma_1^{-1}{}^\bullet\!-\!b)^{-1}$.

Theorem 2 becomes wrong for FA systems, thus also for CF systems. Indeed, a non-backward persistent FA system is provided in Fig. 4.

3.3 Fireability of T-Vectors in WMGs

In this subsection, we develop conditions for the existence of enabled sequences corresponding to given Parikh vectors. For that purpose, we borrow some vocabulary from [26,28] as follows: for a net with incidence matrix C, we say that a

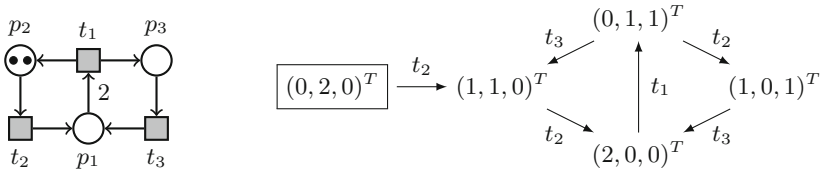


Fig. 4. A Fork-Attribution (FA) system on the left and its reachability graph on the right, where the initial marking is boxed. The FA system is not backward persistent, since the marking $(1,1,0)^T$ can be reached from two predecessors by firing t_2 and t_3 respectively from the initial marking and from $(0,1,1)^T$, but the initial marking has no predecessor.

marking M is *potentially reachable* from a marking M_0 if a T-vector ν exists such that $M = M_0 + C \cdot \nu$. If, additionally, a sequence σ is feasible in (N, M_0) such that $\mathbf{P}(\sigma) = \nu$, we say that ν is fireable (or feasible, or realisable) at M_0.

Lemma 3 (Realisable T-vectors in WMGs). *Let $N = (P, T, W)$ be a WMG with incidence matrix C. Let M be a marking and $\nu \in \mathbb{N}^T$ be a T-vector such that $M + C \cdot \nu \geq 0$. Let T_1 be the support of ν, $P_1 = {}^\bullet T_1 \cap T_1^\bullet$, σ' a transition sequence such that $\nu \leq \mathbf{P}(\sigma')$, and M' be a marking such that $\forall p \in P_1 : M'(p) = M(p)$. Then, if $M'[\sigma'\rangle$, there is a firing sequence $M[\sigma\rangle$ such that $\mathbf{P}(\sigma) = \nu$.*

Proof. By induction on the size of ν. If $\nu = \mathbb{0}$, the property is clearly true. Otherwise, let t be the first transition of T_1 occurring in σ', i.e. $\sigma' = \sigma'_1 t \sigma'_2$ with $\nu(t_i) = 0$ for each t_i in σ'_1.

Assume that $\neg M[t\rangle$, then for some $p \in {}^\bullet t$, $M(p) < W(p,t)$. Since $M + C \cdot \nu \geq 0$, there is $t' \in {}^\bullet p$, $t' \neq t$, such that $t' \in T_1$, and t' is unique in ${}^\bullet p$ since N is a WMG. This contradicts the fact that $M'[\sigma'_1 t\rangle$ since t' does not occur in σ'_1. Hence, we assume that $M[t\rangle M_1$ and $M'[\sigma'_1\rangle M''[t\rangle M'_1[\sigma'_2\rangle$.

Since the net is a WMG, the only transitions able to modify the places in P_1 are in T_1. Thus, we have $M(p) = M'(p) = M''(p)$ and $M_1(p) = M'_1(p)$ for each $p \in P_1$ (no transition of T_1 belongs to σ'_1). Let us denote by δ_t the T-vector with value 1 for t, 0 elsewhere.

Hence, the induction hypothesis applies to $\nu - \delta_t \leq \mathbf{P}(\sigma'_2)$ from the markings M_1 and M'_1, and there is a firing sequence $M_1[\sigma_1\rangle$ with $\mathbf{P}(\sigma_1) = \nu - \delta_t$. Thus, the sequence $\sigma = t\sigma_1$ with Parikh vector ν is enabled at M. The lemma results. □

Instantiating Lemma 3 with $M = M'$, we deduce the next corollary.

Corollary 1 (Potential reachability in WMGs). *Let $N = (P, T, W)$ be a WMG with incidence matrix C. Let M be any marking and $\nu \in \mathbb{N}^T$ be a T-vector such that $M' = M + C \cdot \nu \geq 0$. Let σ be a transition sequence such that $\nu \leq \mathbf{P}(\sigma)$. Then, if $M[\sigma\rangle$, there is a firing sequence $M[\sigma'\rangle M'$ such that $\mathbf{P}(\sigma') = \nu$.*

Lemma 3 and Corollary 1 are not valid in the class of FA systems. Indeed, denoting by C the incidence matrix of the system in Fig. 4, by M_0 its initial

marking $(0,2,0)^T$ and by ν the T-vector $(1,1,1)^T$, we have: $M_0 = M_0 + C \cdot \nu$, and the sequence $\sigma = t_2t_2t_1t_3$ is enabled at M_0, with $\mathbf{P}(\sigma) \geq \nu$. However, there is no initially feasible sequence whose Parikh vector equals ν.

We present next the notion of a *maximal execution vector* in order to obtain Theorem 3 on potentially reachable markings below.

Definition 3 (Maximal execution vector in WMGs). *Let ζ be a WMG system whose set of transitions is T. We denote by $maxex_\zeta : T \to \mathbb{N} \cup \{\infty\}$ the extended T-vector satisfying: $\forall t \in T$, $maxex_\zeta(t)$ is the maximal number of times t may be executed in firing sequences of ζ, allowing the case $maxex_\zeta(t) = \infty$.*

Theorem 3 (Potential reachability in WMGs, revised). *Let $\zeta = (N, M_0)$ be a WMG with incidence matrix C. Let $\nu \in \mathbb{N}^T$ be a T-vector such that $M = M_0 + C \cdot \nu \geq 0$. Let $maxex_\zeta$ be the maximal execution vector of ζ. Then, there exists a firing sequence $M_0[\sigma\rangle M$ with $\mathbf{P}(\sigma) = \nu$ if and only if $\nu \leq maxex_\zeta$.*

Proof. Suppose that $M_0[\sigma\rangle M$ with $\mathbf{P}(\sigma) = \nu$. Since σ can be fired at M_0, we deduce, from the definition of $maxex_\zeta$, that $\forall t \in T$, $maxex_\zeta(t) \geq \mathbf{P}(\sigma)(t)$, thus $\nu \leq maxex_\zeta$.

Conversely, suppose that $\nu \leq maxex_\zeta$. Then, for each $t \in T$, since $\nu(t) \leq maxex_\zeta(t)$, there is a finite firing sequence $M_0[\sigma_t\rangle$ such that $\nu(t) \leq \mathbf{P}(\sigma_t)(t)$. By persistence and Keller's theorem (applied $|T| - 1$ times), there is a finite firing sequence $M_0[\sigma'\rangle$ such that $\forall t \in T : \mathbf{P}(\sigma_t)(t) \leq \mathbf{P}(\sigma')(t)$, hence $\nu \leq \mathbf{P}(\sigma')$ and Corollary 1 applies. □

In this section, we delineated several properties on the reachability graph of WMGs. In the next section, we exploit some of these conditions, notably persistence and backward persistence, to synthesise a WMG from a given lts, when possible.

4 Synthesis of Connected, Bounded, Weakly Live WMGs

In the domain of Petri net synthesis from labelled transition systems, the aim is to build a Petri net system whose reachability graph is isomorphic to a given lts, when it exists. Usually, one has to check first some necessary structural properties of the lts. In some rare cases, such conditions have been proven sufficient for ensuring the existence of a solution (sometimes a unique minimal one) in the class considered and for driving the synthesis process [2,4]. However, in most cases, the known synthesis methods need a combination of such necessary conditions with other computational checks and constructions [1,3,5–7,9].

In this section, we focus on finite, totally reachable and *weakly live* lts, the latter property meaning that each label of T occurs at least once in the lts. We build a procedure synthesising a connected WMG solving such lts when possible.

First, in Subsect. 4.1, we highlight necessary conditions of WMG-solvability, notably persistence, backward persistence and the existence of specific cycles. We also build a counter-example showing that these conditions, when satisfied, are not sufficient to ensure WMG-solvability.

Then, in Subsect. 4.2, we highlight constraints induced by each place and we delineate two subsets of the lts states that are particularly relevant to WMG-synthesis. By focusing the analysis on these states, the number of checking steps is potentially reduced.

Finally, in Subsects. 4.3 and 4.4, we define systems of constraints for two kinds of lts shapes: the cyclic case, i.e. when the lts is strongly connected (hence reversible), and the acyclic case, i.e. when the lts does not contain any cycle. We show these two cases to contain all the lts being solvable by a connected, bounded and weakly live WMG. Also, the number of constraints is reduced by checking only the relevant states defined in Subsect. 4.2. When these systems have a solution, we obtain a WMG solving the lts. To extend this method to all the WMG-solvable lts, the decomposition technique developed in [14–16] to factorise a lts into *prime factors*, i.e. factors that cannot be further factorised and hence should correspond to connected nets, can finally be exploited.

4.1 Necessary Conditions for Solvability with Connected WMGs

For a synthesis into a connected WMG to succeed, the given lts must satisfy the conditions of Proposition 1, the properties described in Proposition 2, as well as persistence and backward persistence, as proved in Theorem 2. The boundedness of the WMG obtained stems from the finiteness of the lts. We capture part of these conditions with the next notation **b** and **c** and explain the relationship between the existence of a cycle in the lts and property **c**.

Properties b and c. For any lts $TS = (S, \rightarrow, T, \iota)$, we denote by:

- **b** the property: TS is finite, weakly periodic, deterministic and backward deterministic, persistent and backward persistent, totally reachable;
- **c** the property: TS is strongly connected, all its small cycles have the same prime Parikh vector π with support T, and $\mathbf{P}(\alpha)$ is a multiple of π for each cycle α.

Let us consider the case in which the finite lts contains a cycle. Then, the (finite) reachability graph of any connected and bounded WMG solving this lts contains a cycle. Thus, from Proposition 2, the cycle contains all transitions. By Corollary 4 in [27], the system is live, and by backward persistence, it is also reversible, implying the strong connectedness of the reachability graph. Consequently, we have to consider only two cases: the given lts is either acyclic or is strongly connected, the second case being considered in property **c**.

Without loss of generality, we assume in the sequel that the lts considered are weakly live. The next lemma presents relationships between properties relevant to the synthesis.

Lemma 4 (Determinism, reversibility, cycle consistence). *Let us consider a weakly live lts $TS = (S, \rightarrow, T, \iota)$.*

(1) If TS satisfies **b***, it also satisfies the full determinism and full backward determinism.*

(2) If TS satisfies **b** *and is acyclic, all the sequences between any two states have the same Parikh vector.*

(3) If TS satisfies **b** *and contains a small prime cycle with support* T *and Parikh vector* π*, then TS satisfies property* **c***, there is a small prime cycle around each state, each arc belongs to a small prime cycle, TS satisfies the strong cycle consistence; also, for any two states* s_1 *and* s_2*, there is a sequence from* s_1 *to* s_2 *whose Parikh vector* δ *is not greater than nor equal to* π*, and each other sequence* σ *from* s_1 *to* s_2 *satisfies* $\mathbf{P}(\sigma) = \mathbf{P}(\delta) + k \cdot \pi$ *for a non-negative integer* k*.*

Proof

(1) Full determinism and full backward determinism arise directly from determinism and backward determinism and from persistence and backward persistence, with the aid of Proposition 3 applied to TS and to its reverse version.

(2) If the lts is acyclic, satisfies **b** and, for some $s \in S$ and $s' \in [s\rangle$, we have $s[\alpha\rangle s'$ as well as $s[\beta\rangle s'$ with $\mathbf{P}(\alpha) \neq \mathbf{P}(\beta)$, and $\alpha \overset{\bullet}{-} \beta$ or $\beta \overset{\bullet}{-} \alpha$ is non-empty (both of them may be non-empty). Then, as in the proof of Lemma 2, with the aid of Keller's theorem and of Lemma 1, we can build a uniform chain $s'[\alpha \overset{\bullet}{-} \beta\rangle s_1[\alpha \overset{\bullet}{-} \beta\rangle s_2 \cdots$ and $s'[\beta \overset{\bullet}{-} \alpha\rangle s_1[\beta \overset{\bullet}{-} \alpha\rangle s_2 \cdots$. Since the lts is finite, there must exist positive integers i and j such that $i < j$ and $s_i = s_j$, forming a non-empty cycle, hence a contradiction with the acyclicity.

(3) In the rest of the proof, we suppose that the lts satisfies **b** and contains a small prime cycle α around some state $s \in S$ with support T. Determinism and persistence imply that cycles can be pushed forward Parikh-equivalently, i.e.: if $s[\alpha\rangle s \wedge s[t\rangle s'$, then $s'[\alpha'\rangle s'$ for some α' with $\mathbf{P}(\alpha') = \mathbf{P}(\alpha)$ (applying Keller's theorem). Symmetrically, backward determinism and backward persistence imply that cycles can be pushed backward Parikh-equivalently.

Now, consider any non-empty cycle β around some state s' in TS. Both cycles α and β can be pushed backward Parikh-equivalently to the initial state ι (since s and s' are reachable from ι by total reachability). Using Keller's theorem, both support-disjoint sequences $\alpha \overset{\bullet}{-} \beta$ and $\beta \overset{\bullet}{-} \alpha$ are feasible at ι and lead to some marking s_0. Since $(\alpha \overset{\bullet}{-} \beta)^n$ and $(\beta \overset{\bullet}{-} \alpha)^n$ are feasible at ι for every positive integer n while the lts is finite, there exists a positive integer m such that $(\alpha \overset{\bullet}{-} \beta)^m$ and $(\beta \overset{\bullet}{-} \alpha)^m$ are cycles. Since the lts is also weakly periodic, deterministic and backward deterministic, both $\alpha \overset{\bullet}{-} \beta$ and $\beta \overset{\bullet}{-} \alpha$ are cycles. Since $\alpha, \beta \neq \epsilon$ and $supp(\alpha) = T$, we have $\mathbf{P}(\alpha \overset{\bullet}{-} \beta) \lneq \mathbf{P}(\alpha)$. Hence, if $\alpha \overset{\bullet}{-} \beta \neq \epsilon$, it forms a smaller cycle, contradicting the fact that α is already a small cycle. Thus, necessarily, $\alpha \overset{\bullet}{-} \beta = \epsilon$, implying that $\mathbf{P}(\beta) \geq \mathbf{P}(\alpha)$. Suppose that $\mathbf{P}(\beta)$ is not a multiple of $\mathbf{P}(\alpha)$. Denote by k the largest integer such that $\mathbf{P}(\beta) \geq k \cdot \mathbf{P}(\alpha)$ and $\beta' = \beta \overset{\bullet}{-} \alpha^k \neq \epsilon$. Necessarily, $\mathbf{P}(\alpha) \not\geq \mathbf{P}(\beta')$ and $\mathbf{P}(\beta') \not\geq \mathbf{P}(\alpha)$, implying that $\mathbf{P}(\alpha) \gneqq \mathbf{P}(\alpha \overset{\bullet}{-} \beta') \gneqq \mathbb{0}$, where $\alpha \overset{\bullet}{-} \beta'$ is a cycle, contradicting the fact that α is a small cycle. We deduce that $\mathbf{P}(\beta)$, as well as each other Parikh vector of each non-empty cycle of the lts, is a multiple of $\mathbf{P}(\alpha) = \pi$.

Hence, from total reachability and persistence, there is a small prime cycle (with Parikh vector π) around the initial state, as well as around any state.

Since there is a small cycle with support T around each state, by Keller's theorem each arc can be extended into a cycle: $s[t\rangle s'$ implies there is a sequence $s'[\gamma\rangle s$ with $\mathbf{P}(t\gamma) = \pi$. As a consequence, TS is reversible, thus strongly connected.

For any cycle $s[\beta\rangle s$, from the above, we have that $\mathbf{P}(\beta) = k \cdot \pi$ for some integer $k \geq 0$. Now, if a sequence $s[\gamma\rangle s'$ is such that $k_1 \cdot \mathbf{P}(\gamma) = k_2 \cdot \pi$ for some positive integers k_1, k_2, since π is prime k_1 must divide k_2; let us denote by k' the integer k_2/k_1. We have $\mathbf{P}(\gamma) = k' \cdot \pi$ so that by full determinism $s = s'$, hence the strong cycle consistence.

Consider a sequence $s[\alpha\rangle s'$ in TS. Suppose that $\mathbf{P}(\alpha) \geq \pi$. Since there is a small cycle γ with Parikh vector π around s, we build a shorter sequence by applying Keller's theorem as follows: $\alpha \stackrel{\bullet}{-} \gamma$ is fireable at s and leads to s'. We can build such shorter sequences until we get a sequence $s[\alpha'\rangle s'$ with $\mathbf{P}(\alpha') \not\geq \pi$ and $\mathbf{P}(\alpha) = \mathbf{P}(\alpha') + k \cdot \pi$ for some integer $k > 0$. If we start from another sequence $s[\beta\rangle s'$, we get similarly $s[\beta'\rangle s'$ with $\mathbf{P}(\beta') \not\geq \pi$, and $\mathbf{P}(\beta) = \mathbf{P}(\beta') + h \cdot \pi$ for a non-negative integer h. If $\mathbf{P}(\beta') \neq \mathbf{P}(\alpha')$, then we have two cases.

In the first case, one of them is greater than the other one, without loss of generality $\mathbf{P}(\beta') \geq \mathbf{P}(\alpha')$, in which case, with Keller's theorem, we can construct from s' the cycle $\beta' \stackrel{\bullet}{-} \alpha'$ with $\mathbf{P}(\beta' \stackrel{\bullet}{-} \alpha') \not\geq \mathbf{P}(\pi)$ a contradiction with the fact that the Parikh vector of every cycle of TS is a multiple of π (with support T).

In the second case, $\beta' \stackrel{\bullet}{-} \alpha'$ and $\alpha' \stackrel{\bullet}{-} \beta'$ are both non-empty with disjoint supports, their support containing at least one null component. In this second case, we construct from $s_1 = s'$ a chain $s_1[\sigma\rangle s_2[\sigma\rangle s_3 \cdots$, with $\sigma = \beta' \stackrel{\bullet}{-} \alpha'$ as well as for $\sigma = \alpha' \stackrel{\bullet}{-} \beta'$. Since the lts is finite, we must have $s_i = s_j$ for some $i < j$, hence a cycle with Parikh vector $n \cdot \pi = (j - i) \cdot \mathbf{P}(\sigma)$ for some positive integer n, which is incompatible with the fact that the support of π is T and the support of σ does not contain all transitions.

Thus, we get a contradiction in both cases, implying that $\mathbf{P}(\beta') = \mathbf{P}(\alpha')$. We deduce that for every sequence σ from s_1 to s_2, there exists a sequence $\delta \not\geq \pi$ from s_1 to s_2 such that $\mathbf{P}(\sigma) = \mathbf{P}(\delta) + k \cdot \pi$ for some non-negative integer k. □

Insufficiency of the Necessary Conditions b and c for WMG-Solvability. In Fig. 5, we provide an example of an FA system whose reachability graph satisfies all conditions of properties **b** and **c** but is not WMG-solvable. We deduce that these conditions, when satisfied by a given lts, are not sufficient for ensuring the existence of a solution in the WMG subclass. Indeed, in Fig. 5, each possible attempt of a construction leads to a contradiction, as detailed next.

– Non-existence of a WMG solution with six places:
To obtain a WMG solution structured as on the right of Fig. 5, the following sequences must be feasible at the initial marking M_0:

$$t_3t_1 \Rightarrow M_0(p_1) \geq 3 \qquad t_3t_1t_3t_2t_1t_3t_2t_1t_3t_2t_3 \Rightarrow M_0(p_4) \geq 9$$
$$t_3t_1t_2t_3t_1t_3t_2t_1 \Rightarrow M_0(p_2) \geq 3 \qquad t_3t_1t_3 \Rightarrow M_0(p_5) \geq 6$$
$$t_2t_3t_1t_2 \Rightarrow M_0(p_3) \geq 9 \qquad t_2t_3t_1t_2t_3t_1t_3t_1t_2 \Rightarrow M_0(p_6) \geq 12$$

The sequence $\sigma = t_2t_3t_1t_3t_1t_3t_2t_1t_3t_3$ is then feasible in such a constrained WMG but is not enabled in the FA system ζ.

– Non-existence of a WMG solution with fewer places:
Since the previous WMG system with six places and its necessary marking are too permissive, we deduce that the same contradicting sequence σ is also feasible in all less constrained WMGs, typically obtained by removing some places while retaining the necessary initial marking in the other places.

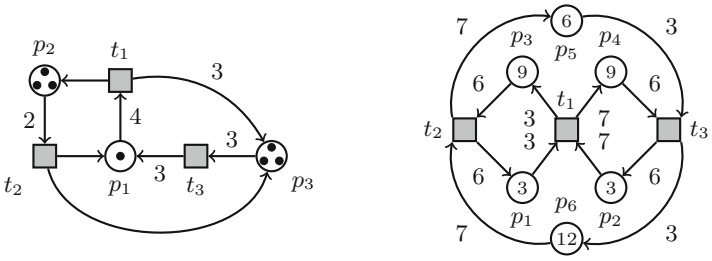


Fig. 5. A Fork-Attribution (FA) system ζ is pictured on the left. Its minimal prime T-semiflow $\pi = (6, 3, 7)$ equals the Parikh vector of each small cycle of $RG(\zeta)$. The latter is persistent and backward persistent, reversible, finite and fulfills properties **b** and **c**. The most constrained WMG solution ζ' whose reachability graph could be isomorphic to $RG(\zeta)$ is depicted on the right: its weights are directly deduced from π and, in each place, the given amount of tokens is necessary to enable the sequences of $RG(\zeta)$. However, this necessary initial marking already enables a sequence that is not feasible in ζ, namely $\sigma = t_2t_3t_1t_3t_1t_3t_2t_1t_3t_3$. Since every possible variant ζ'' of ζ' is less constrained than ζ', each such ζ'' also enables σ. We deduce that no WMG solves $RG(\zeta)$.

Checking the Necessary Conditions in a Pre-synthesis Phase. Since the lts is finite, all the necessary properties for solvability can be checked in a naive way. However, some algorithmic improvement can be achieved by considering an adequate checking order. For instance, from Proposition 3, property **b** implies full determinism and full backward determinism, whose checking is thus avoided. In the next subsection, we exhibit subsets of states of the lts whose analysis is sufficient to ensure some constraints, allowing to perform fewer operations.

4.2 Constraints and Subsets of States Relevant to WMG-Synthesis

In the following, we describe some constraints that must be fulfilled in order to synthesise a WMG. Also, we define two subsets of the states of the given lts that are sufficient to check in order to fulfill several constraints over all states, decreasing potentially the size of the systems of constraints to solve.

A WMG synthesis amounts to build places of the kind schematised in Fig. 6.

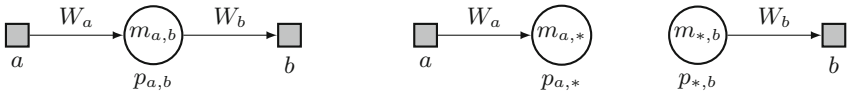


Fig. 6. Possible types of places for the synthesis of a WMG (N, M_0), with initial marking $m_{a,b} = M_0(p_{a,b})$, $m_{a,*} = M_0(p_{a,*})$ and $m_{*,b} = M_0(p_{*,b})$.

Constraints Related to Places in the WMG. Note that a place $p_{a,*}$ is equivalent to a place $p_{a,b}$ with $W_b = 0$, and a place $p_{*,b}$ is equivalent to a place $p_{a,b}$ with $W_a = 0$. In a place $p_{a,b}$, we can always choose W_a and W_b relatively prime without loss of generality, with an adequate initial marking M_0. If a and b are the same label, then we have a single transition and the place is equivalent to either a place $p_{a,*}$, $p_{*,a}$ or no place at all, depending on the sign of the difference between W_a and W_b. In a place $p_{a,*}$, the initial marking $M_0(p_{a,*})$ may always be chosen as 0 and the weight W_a as 1. In a place $p_{*,b}$, we must have $W_b \leq M_0(p_{*,b})$ (otherwise the lts would not be weakly live), and the weight W_b can always be chosen as 1, with an adequate choice of the initial marking M_0.

If $T = \emptyset$, TS is reduced to its initial state and the (minimal) solution is the empty Petri net. If $T = \{a\}$ is a singleton, either TS is acyclic, in the form of a single chain, and the minimal solution is a place $p_{*,a}$, with an initial marking deduced from the length of the chain, or it is a loop $\iota[a\rangle\iota$ with a minimal solution reduced to a transition a without any place. Hence, in the following, we assume without loss of generality that $|T| > 1$. We shall also assume that the lts to be synthesised satisfies property **b** and either acyclicity or **c**.

M_0 is the marking corresponding to the initial state ι; consider any state $s \in S$ with a shortest sequence from ι to s, meaning that no other sequence from ι to s has a smaller Parikh vector. By Lemma 4 (point 2 or 3), such a sequence exists, and all such sequences from ι to s share the same Parikh vector Δ_s. The marking corresponding to state s is given by $M_s(p_{a,b}) = M_0(p_{a,b}) + \Delta_s(a) \cdot W_a - \Delta_s(b) \cdot W_b$. The next conditions are necessary and sufficient for allowing (and realising) a synthesis, and are related to the classical regional approach [1]:

- The number of tokens in $p_{a,b}$ must remain non-negative at each reachable marking described by a state in S.
- For each state s not allowing b, there must exist a place p such that W_b is larger than $M_s(p)$, where M_s is the marking associated to s.
- Any two different states s', s'' must be distinguished by a place p' such that $M_{s'}(p') \neq M_{s''}(p')$.

In many cases, notably for CF and MG synthesis [3,4,9,12], hence in this study, the last constraint, called the *separation property*, arises from the other two and from the assumptions on TS.

Two Subsets of States Relevant to the WMG-Synthesis. The first two constraints above are linked to two particular subsets of states of TS: for each label $x \in T$, we define

$OX(x) = \{r \in S \mid r[t\rangle \Rightarrow t = x\}$ and $NXX(x) = \{s \in S | \neg s[x\rangle \wedge \forall s' \in s^\bullet : s'[x\rangle\}$.

For each state s in $OX(x)$ (the notation stemming from "Only X"), the only arc starting at s, if any, is labelled x. Let us consider a place $p_{a,b}$ and a longest sequence without a starting from some state s. This sequence is finite since the lts is finite, and each cycle along the sequence, if any, has support T, hence contains an a. Thus, we reach a state r either without successor (this may only occur if the lts is acyclic) or with a single output a, hence in $OX(a)$ in both cases, and $M_s(p_{a,b}) \geq M_r(p_{a,b})$. As a consequence, to check that all markings of $p_{a,b}$ reachable from ι are non-negative, we only have to check the states in $OX(a)$: $r \in OX(a) \Rightarrow M_r(p_{a,b}) \geq 0$ and $M_s(p_{a,b}) \geq 0$.

For each state s in $NXX(x)$, x cannot be executed at s (hence the first two letters NX of the notation), but in each next state s', if any, x is enabled (hence the last letter X of the notation). Let us assume that a place $p_{a,b}$ may be used to exclude performing b at some state s (i.e. $\neg s[b\rangle$), meaning $M_s(p_{a,b}) < W_b$. If $s'[t\rangle s$ with $\neg s'[b\rangle$ (which implies $t \neq b$), then $M_s(p_{a,b}) \geq M_{s'}(p_{a,b})$, so that the same place $p_{a,b}$ disables b at s'. Moreover, the longest chains of states excluding to perform b are necessarily finite since b occurs in any non-empty cycle; hence they all end in states of $NXX(b)$. As a consequence, in order to exclude performing b when necessary, one only has to find, for each state $r \in NXX(b)$, a place $p_{a,b}$ such that $M_r(p_{a,b}) < W_b$ (while allowing all valid transitions, as expressed through $OX(a)$). In some cases, a same place $p_{a,b}$ can be used for several states in $NXX(b)$.

In our case, for any label x, the states in $NXX(x)$ have a very special shape, highlighted in the next lemma whose proof is illustrated in Fig. 7.

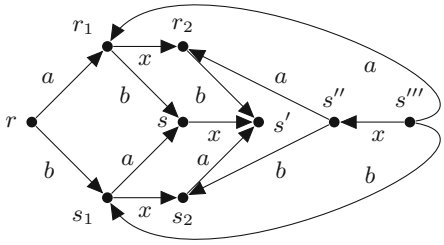


Fig. 7. Illustration of the proof of Lemma 5.

Lemma 5 (Single outputs of the states in NXX). *Let $TS = (S, \rightarrow, T, \iota)$ be a lts satisfying property* **b**. *If $x, a, b \in T$, $r \in NXX(x)$, $r[a\rangle$ and $r[b\rangle$, then $a = b$.*

Proof. Let us assume that $r[a\rangle r_1$ and $r[b\rangle s_1$ with $a \neq b$. Since $r \in NXX(x)$, we have $r_1[x\rangle r_2$ and $s_1[x\rangle s_2$ for some states r_2, s_2. By persistence (and determinism), we also have $r_1[b\rangle s$, $s_1[a\rangle s$, $s[x\rangle s'$, $r_2[b\rangle s'$ and $s_2[a\rangle s'$ for some s, s'. By backward persistence, we then have $s''[a\rangle r_2$ and $s''[b\rangle s_2$ for some s'', as well as $s'''[x\rangle s''$ and $s'''[b\rangle s_1$ for some s'''. Finally, by backward determinism, $s''' = r$ and $r[x\rangle$, contradicting the fact that $r \in NXX(x)$. □

If TS is acyclic, by persistence there is a unique (maximal) state without successor; let us call it s_∞; we then have $s_\infty \in \cap_{x\in T} NXX(x)$. If TS is cyclic, there is no such state. In any case, we may have several states s and label $a \neq x$ in some $NXX(x)$ with $s[ax\rangle$.

4.3 Computational Synthesis in the General Cyclic Case

Let $TS = (S, \rightarrow, T, \iota)$ be a lts satisfying properties **b** and **c**, denoting by π the unique minimal Parikh vector of small cycles, with support T. Each place $p_{a,b}$ must satisfy $W_a \cdot \pi(a) = W_b \cdot \pi(b)$, thus we can choose $W_a = \pi(b)$ and $W_b = \pi(a)$ (or any proportional values[1], in particular $\pi(b)/\gcd(\pi(a), \pi(b))$ and $\pi(a)/\gcd(\pi(a), \pi(b))$), and the only parameter that still needs to be fixed is the initial marking (plus the exact pairs a, b for which we need those places).

For each $b \in T$, we need such a place $p_{a,b}$ if there is a state $s \in NXX(b)$ such that $s[ab\rangle$ (otherwise, there is no way to enable a b after an a when b is not directly enabled). We denote by $pred(b)$ the set $\{a \in T | \exists s \in NXX(b), s[ab\rangle\}$ and, for any $a \in pred(b)$, $NXX(a, b) = \{s \in NXX(b) | s[ab\rangle\}$.

For each $a, b \in T$ such that $a \in pred(b)$, since $W_a = \pi(b)$ and $W_b = \pi(a)$, we have to solve the following constraints (in M_0, over the non-negative integers):

$$\begin{cases} \forall s \in OX(a) : M_0(p_{a,b}) \geq \Delta_s(b) \cdot \pi(a) - \Delta_s(a) \cdot \pi(b) \\ \forall s \in NXX(a, b) : M_0(p_{a,b}) < \Delta_s(b) \cdot \pi(a) - \Delta_s(a) \cdot \pi(b) + \pi(a) \end{cases}$$

This amounts to first compute

$$M_0(p_{a,b}) = \max_{s\in OX(a)} \{\Delta_s(b) \cdot \pi(a) - \Delta_s(a) \cdot \pi(b)\}$$

and then to check that, for each $s \in NXX(a, b)$,

$$M_0(p_{a,b}) < \Delta_s(b) \cdot \pi(a) - \Delta_s(a) \cdot \pi(b) + \pi(a).$$

If each such system of constraints is solvable, we obtain a WMG solution of TS. Otherwise, there is no solution and the reason is known.

4.4 Computational Synthesis in the General Acyclic Case

In the acyclic case, we may first apply the factorisation techniques of [14–16] to check if the given lts is prime and thus has a chance to have a connected solution. The weights W_a and W_b around the place $p_{a,b}$ are not constrained by a T-semiflow. Thus, we may need variants of such places (differing by the weights W_a, W_b and the initial marking). We may also need places $p_{*,b}$ and $p_{a,*}$; in particular, a place $p_{*,b}$ with $W_b = 1$ and $M_0 = \Delta_{s_\infty}(b)$ excludes executing b at the final state s_∞. Such a place may be redundant with other ones, but we do not aim here at building an optimal solution: we focus on the existence of a solution and on its construction.

[1] This is the only way to define an adequate $p_{a,b}$; in particular, there is no $p_{*,b}$ or $p_{a,*}$.

In this acyclic case, the enabledness of labels is described by the first set of constraints below, using again the sufficient condition stating that the markings at states from $OX(a)$ must be non-negative. The last constraint expresses that the place is useful for excluding some transition from some state.

For each $b \in T$, $a \in pred(b)$ and $s \in NXX(a, b)$, we have to solve the following constraints (in $M_0(p_{a,b}), W_a, W_b \in \mathbb{N}$):

$$\begin{cases} \forall s' \in OX(a) : M_0(p_{a,b}) \geq \Delta_{s'}(b) \cdot W_b - \Delta_{s'}(a) \cdot W_a \\ M_0(p_{a,b}) < \Delta_s(b) \cdot W_b - \Delta_s(a) \cdot W_a + W_b. \end{cases}$$

To solve such a system, we can first consider the system in W_a and W_b:

$$\forall s' \in OX(a) : \Delta_s(b) \cdot W_b - \Delta_s(a) \cdot W_a + W_b > \Delta_{s'}(b) \cdot W_b - \Delta_{s'}(a) \cdot W_a$$

i.e.: $\forall s' \in OX(a) : [\Delta_s(b) - \Delta_{s'}(b) + 1] \cdot W_b > [\Delta_s(a) - \Delta_{s'}(a)] \cdot W_a$

and then check if there exists a solution satisfying:

$$\Delta_s(b) \cdot W_b - \Delta_s(a) \cdot W_a + W_b > 0.$$

If each such system of constraints is solvable, we obtain a WMG solution of TS. Otherwise, no solution exists and we know the reason.

5 Conclusions and Perspectives

Weighted marked graphs (WMGs) form a well-known subclass of Petri nets with numerous real-life applications. These nets have been extensively studied in previous works, leading to strong theoretical results.

For this class, we obtained new structural and behavioural properties, such as backward persistence. We also delineated necessary conditions that must be fulfilled by a labelled transition system to be WMG-solvable. We showed that these necessary conditions are not sufficient. Finally, we specialised the synthesis procedures devised for choice-free nets in [3,5–7,9] to WMG nets.

A perspective is to develop additional properties of WMGs in order to enhance the pre-synthesis phase, allowing to discard non-solvable systems promptly. Ideally, such properties would characterise the WMG-solvable labelled transition systems in a purely structural way, in the spirit of the methods designed for plain marked graphs and T-systems in [2,4].

References

1. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47967-4
2. Best, E., Devillers, R.: Characterisation of the state spaces of live and bounded marked graph Petri nets. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) LATA 2014. LNCS, vol. 8370, pp. 161–172. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04921-2_13

3. Best, E., Devillers, R.: Synthesis of persistent systems. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 111–129. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07734-5_7
4. Best, E., Devillers, R.: State space axioms for T-systems. Acta Inf. **52**(2–3), 133–152 (2015). https://doi.org/10.1007/s00236-015-0219-0
5. Best, E., Devillers, R.: Synthesis and reengineering of persistent systems. Acta Inf. **52**(1), 35–60 (2015). https://doi.org/10.1007/s00236-014-0209-7
6. Best, E., Devillers, R.: Synthesis of bounded choice-free Petri nets. In: Aceto, L., Frutos Escrig, D. (eds.) Proceedings of 26th International Conference on Concurrency Theory (CONCUR 2015), pp. 128–141 (2015). https://doi.org/10.4230/LIPIcs.CONCUR.2015.128
7. Best, E., Devillers, R.: Synthesis of live and bounded persistent systems. Fundam. Inform. **140**, 39–59 (2015). https://doi.org/10.3233/FI-2015-1244
8. Best, E., Devillers, R.: Characterisation of the state spaces of marked graph Petri nets. Inf. Comput. **253**(3), 399–410 (2017). https://doi.org/10.1016/j.ic.2016.06.006
9. Best, E., Devillers, R., Schlachter, U.: Bounded choice-free Petri net synthesis: algorithmic issues. Acta Inform. **54**, 1–37 (2017)
10. Commoner, F., Holt, A., Even, S., Pnueli, A.: Marked directed graphs. J. Comput. Syst. Sci. **5**(5), 511–523 (1971). https://doi.org/10.1016/S0022-0000(71)80013-2
11. Crespi-Reghizzi, S., Mandrioli, D.: A decidability theorem for a class of vector-addition systems. Inf. Process. Lett. **3**(3), 78–80 (1975). https://doi.org/10.1016/0020-0190(75)90020-4
12. Darondeau, P.: Equality of languages coincides with isomorphism of reachable state graphs for bounded and persistent Petri nets. Inf. Process. Lett. **94**(6), 241–245 (2005). https://doi.org/10.1016/j.ipl.2005.03.002
13. Desel, J., Esparza, J.: Free Choice Petri Nets, Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, New York (1995)
14. Devillers, R.: Products of transition systems and additions of Petri nets. In: Desel, J., Yakovlev, A. (eds.) Proceedings of 16th International Conference on Application of Concurrency to System Design (ACSD 2016), pp. 65–73 (2016). https://doi.org/10.1109/ACSD.2016.10
15. Devillers, R.: Factorisation of transition systems. Acta Inform. **54**, 1–24 (2017). https://doi.org/10.1007/s00236-017-0300-y
16. Devillers, R., Schlachter, U.: Factorisation of Petri net solvable transition systems. In: 39th International Conference on Applications and Theory of Petri Nets and Concurrency, Bratislava (2018)
17. Hujsa, T., Delosme, J.M., Munier-Kordon, A.: On liveness and reversibility of equal-conflict Petri nets. Fundam. Inform. **146**(1), 83–119 (2016). https://doi.org/10.3233/FI-2016-1376
18. Hujsa, T., Devillers, R.: On liveness and deadlockability in subclasses of weighted Petri nets. In: van der Aalst, W., Best, E. (eds.) PETRI NETS 2017. LNCS, vol. 10258, pp. 267–287. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57861-3_16
19. Keller, R.M.: A fundamental theorem of asynchronous parallel computation. In: Feng, T. (ed.) Parallel Processing. LNCS, vol. 24, pp. 102–112. Springer, Heidelberg (1975). https://doi.org/10.1007/3-540-07135-0_113
20. Lee, E., Messerschmidt, D.: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. Comput. **C–36**(1), 24–35 (1987). https://doi.org/10.1109/TC.1987.5009446

21. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. Proc. IEEE **75**(9), 1235–1245 (1987)
22. Marchetti, O., Munier-Kordon, A.: A sufficient condition for the liveness of weighted event graphs. Eur. J. Oper. Res. **197**(2), 532–540 (2009). https://doi.org/10.1016/j.ejor.2008.07.037
23. Pino, J.L., Bhattacharyya, S.S., Lee, E.A.: A hierarchical multiprocessor scheduling framework for synchronous dataflow graphs. Technical report, University of California, Berkeley, May 1995
24. Silva, M., Terue, E., Colom, J.M.: Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 309–373. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_19
25. Sriram, S., Bhattacharyya, S.S.: Embedded Multiprocessors: Scheduling and Synchronization. Signal Processing and Communications. CRC Press, Boca Raton (2009)
26. Teruel, E., Chrzastowski-Wachtel, P., Colom, J.M., Silva, M.: On weighted T-systems. In: Jensen, K. (ed.) ICATPN 1992. LNCS, vol. 616, pp. 348–367. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55676-1_20
27. Teruel, E., Colom, J.M., Silva, M.: Choice-free Petri nets: a model for deterministic concurrent systems with bulk services and arrivals. IEEE Trans. Syst. Man Cybern. Part A **27**(1), 73–83 (1997). https://doi.org/10.1109/3468.553226
28. Teruel, E., Silva, M.: Structure theory of equal conflict systems. Theor. Comput. Sci. **153**(1–2), 271–300 (1996). https://doi.org/10.1016/0304-3975(95)00124-7

Elementary Net Synthesis Remains NP-Complete Even for Extremely Simple Inputs

Ronny Tredup[1(✉)], Christian Rosenke[2], and Karsten Wolf[1]

[1] Institut für Informatik, Universität Rostock, 18051 Rostock, Germany
{ronny.tredup,karsten.wolf}@uni-rostock.de
[2] arivis AG, Erika-Mann-Straße 19-23, 80636 Munich, Germany
christian.rosenke@arivis.com

Abstract. Elementary net systems (ENS) are the most fundamental class of Petri nets. Their synthesis problem has important applications in the design of digital hardware and commercial processes. Given a labeled transition system (TS) A, feasibility is the NP-complete decision problem whether A can be synthesized into an ENS. It is known that A is feasible if and only if it has the event state separation property (ESSP) and the state separation property (SSP). Recently, these properties have also been studied individually for their practical implications. A fast ESSP algorithm, for instance, would allow applications to at least validate the language equivalence of A and a synthesized ENS. Being able to efficiently decide SSP, on the other hand, could serve as a quick-fail preprocessing mechanism for synthesis. Although a few tractable subclasses have been found, this paper destroys much of the hope that many practically meaningful input restrictions make feasibility or at least one of ESSP and SSP efficient. We show that all three problems remain NP-complete even if the input is restricted to linear TSs where every event occurs at most three times.

1 Introduction

ENSs are a Petri net class that provides a lot of useful properties for the specification, verification, and synthesis of asynchronous or self-timed circuits [7,13]. That is why synthesizing ENSs is a useful act in the description of processes in digital hardware. The inherent concepts of choice and causality also make ENSs the ideal starting point for process modeling languages like the Business Process Modeling Notation [14], Event Driven Process Chains [15] or activity diagrams in the UML standard [16]. As pointed out in [1], especially the simpleness of ENSs is useful for the specification of workflow management systems like MILANO.

Feasibility for ENS synthesis is the problem to decide if a given TS A can be synthesized into an ENS, that is, if it is isomorphic to the state graph of some ENS. Traditionally, this problem is approached by the *state separation property*, SPP, and the *event state separation property*, ESSP. In fact, A is isomorphic to the state graph of an ENS if and only if it satisfies both properties [4].

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 40–59, 2018.
https://doi.org/10.1007/978-3-319-91268-4_3

This does not mean that the SSP and the ESSP are not of interest when considered alone. For instance, synthesizing TSs having only the ESSP leads to Petri nets that implement the given behavior but have fewer states [4].

Deciding the SSP or ESSP is NP-complete [3,8] shows that feasibility is NP-complete, too. Nevertheless, considerable efforts are made to find practically relevant tractable cases. For example, feasibility becomes tractable for Flip-Flop nets, a superclass of ENSs [12]. Free-Choice Acyclic ENSs [1], a sub-class of ENS applied in workflow models, allow polynomial time feasibility, too.

The subject of this paper is to analyze if, instead of altering the admissible output nets, restricting the allowed input TSs can lead to tractable cases, too. The hope is that there are considerably large subclasses of practically significant TSs that make, feasibility or at least one of the SSP or ESSP, efficiently decidable. Thinking of reasonable input constraints, putting restrictions on the number of events that can happen at a state is probably the first suggestion coming up. As a matter of fact, benchmarks in the digital hardware design community show that TSs often have few choices, that is, states with an overall limited degree of outgoing arcs [6]. Another immediate idea to reduce possible input is to limit the number of occurrences of an event in the whole TS.

This paper shows that natural input restrictions like these have little influence on the problems complexity. In fact, it shows that the SSP, the ESSP, and feasibility remain NP-complete even for what we call linear 3-fold TSs. Here, every state, except for initial and terminal, has exactly one successor and predecessor and every event occurs at most three times in the TS.

That the problem's hardness survives even on linear TSs is very surprising. They have already been studied in [5] where feasibility is shown to be decidable by a letter counting algorithm in quadratic time if the linear input TSs have just two different events and the sought net is a place/transition Petri net.

We organize this paper as follows: The following section introduces preliminary notions. Section 3 develops a modular concept for TSs and shows that, for linear TSs, the ESSP implies the SSP. Notice that this implies the equivalence of feasibility and ESSP for this class of TSs. Section 4 gives a polynomial time reduction of cubic monotone one-in-three 3-SAT to linear 3-fold ESSP, implying that the latter is NP-complete. Hence, 3-fold feasibility becomes NP-complete, too. Finally, Sect. 5 provides a polynomial time reduction of linear 3-fold ESSP to linear 3-fold SSP. Hence, the last of the three problems also turns NP-complete. This paper is an extended abstract for the first part of the technical report [11]. The proofs that had to be removed due to space limitation are given in the report.

2 Preliminaries

A (deterministic) *transition system* $A = (S, E, \delta, s_0)$ is defined by finite disjoint sets S of states and E of events, a partial transition function $\delta : S \times E \to S$, and an initial state $s_0 \in S$. We understand A as an edge-labeled directed graph using S as node set. Every triple $\delta(s, e) = s'$ is read as an e-labeled arc $s \xrightarrow{e} s'$

from s to s'. We say that an event e *occurs* at a state s if $\delta(s,e) = s'$ for some state s' and we formally abbreviate this with $s \xrightarrow{e}$.

This paper deals with *linear* TSs defined by $A = s_0 \xrightarrow{e_1} s_1 \xrightarrow{e_2} \dots \xrightarrow{e_t} s_t$, the sequence of states and events starting with s_0 and ending at a terminal s_t. The only present arcs $s_{i-1} \xrightarrow{e_i} s_i$ link consecutive states for $i \in \{1,\dots,t\}$. The events are $E = \bigcup_{i=1}^{t} \{e_i\}$. Aside from linearity this paper is mostly restricted to 3-fold TSs where every event in E occurs at most at three states.

For a *region*, the pivot concept of Petri net synthesis, we need the following notions: Given a set R of states, an arc $s \xrightarrow{e} s'$ *enters* R, if $s \notin R$ and $s' \in R$, *exits* R, if $s \in R$ and $s' \notin R$, or else *obeys* R. With respect to this, a set of states $R \subseteq S$ is a region if there is a so-called *signature* $sig : E \to \{-1, 0, 1\}$ that, for every event $e \in E$, specifies the behavior $sig(e)$ of all e-labeled arcs (1: enter, 0: obey, -1: exit). This means, all arcs $s \xrightarrow{e} s'$ have to satisfy the equation $R(s') = sig(e) + R(s)$, where, by a little abuse of notation, $R(s) = 1$ if $s \in R$ and otherwise $R(s) = 0$ for all $s \in S$. For a region R, there exists only one signature, so we may call it *the* signature sig_R of R. By $\mathcal{R}(A)$ we denote the set of all regions of a TS A. Moreover, we use $enter_R = \{e \mid sig_R(e) = 1\}$, $exit_R = \{e \mid sig_R(e) = -1\}$, and $obey_R = \{e \mid sig_R(e) = 0\}$ to classify events according to their behavior with respect to R's border.

Based on the definition of regions, we say that two states $s, s' \in S$ are *separable* in A if there is a region $R \in \mathcal{R}(A)$ with $R(s) \neq R(s')$. The *state separation property*, is satisfied by A if all states are pairwise separable. An event $e \in E$ is called *inhibitable* at state $s \in S$ with $\neg(s \xrightarrow{e})$ if there is a region $R \in \mathcal{R}(A)$ such that $R(s) = 0$ and $sig_R(e) = -1$. A TS A has the *event state separation property* if all events e are inhibitable at all states s that do not fulfill $s \xrightarrow{e}$. If A has the SSP and ESSP it is called *feasible*. We say that a subset $\mathcal{R} \subseteq \mathcal{R}(A)$ is a witness for the SSP, respectively the ESSP, of A if all states are separable, respectively all events are inhibitable, by the regions of $\mathcal{R}$.

The next observation for a region R follows by a simple induction over the transitions $s \xrightarrow{e} s'$ and the requirement $R(s') = R(s) + sig_R(e)$:

Observation 1. *If R is a region of a linear TS $A = s_0 \xrightarrow{e_1} \dots \xrightarrow{e_t} s_t$ then every subsequence $s_i \xrightarrow{e_{i+1}} \dots \xrightarrow{e_j} s_j$ fufills $\sum_{l=i+1}^{j} sig_R(e_l) = R(s_j) - R(s_i)$.*

Notice that we do not formally define ENSs in this paper. As we approach the feasibility problem by the SSP and ESSP which are defined on the basis of TSs it is not necessary. For the interested reader, we recommend the monograph [4] for an excellent introduction to the topic.

3 Unions, Transition System Containers

The following concept of *unions* allows us to easily generate gadgets for our NP-completeness proofs. Formally, if

$$A_0 = s_0^0 \xrightarrow{e_1^0} \dots \xrightarrow{e_{t_0}^0} s_{t_0}^0 \text{ to } A_n = s_0^n \xrightarrow{e_1^n} \dots \xrightarrow{e_{t_n}^n} s_{t_n}^n$$

are linear TSs with pairwise disjoint states (but not necessarily disjoint events) then we say that $U = U(A_0, \ldots, A_n)$ is their *union*. By $S(U)$ we denote the entirety of all states in $A_0, \ldots, A_n$ and $E(U)$ is the aggregation of all events. If every event in $E(U)$ occurs at most at three states of $S(U)$, which are not necessarily part of the same TS, then we say that U is 3-fold.

For simplicity, we also build unions recursively: Firstly, every TS A is identified with the union containing only A, that is, $A = U(A)$. Next, if $U_1 = U(A_0^1, \ldots, A_{n_1}^1)$ to $U_m = U(A_0^m, \ldots, A_{n_m}^m)$ are unions, where possibly $A_i = U_i$, then $U(U_1, \ldots, U_m)$ is the flattened union $U(A_0^1, \ldots, A_{n_1}^1, \ldots, A_0^m, \ldots, A_{n_m}^m)$.

Next, we lift regions, the SSP and ESSP to unions as follows: We say that $R \subseteq S(U)$ is a region of U if and only if it permits a signature $sig_R : E(U) \to \{-1, 0, 1\}$ in the following sense: For all $i \in \{0, \ldots, n\}$ the subset $R_i = R \cap S_i$ of R, coming from the states S_i of A_i, has to be a region of A_i with the signature sig_{R_i} that resembles sig_R on the events E_i of A_i. This means $sig_R(e) = sig_{R_i}(e)$ for all $e \in E_i$. Then, U has the SSP if all states $s, s' \in S(U)$, that are part of the same TS A_i, are separable by A_i's part $R \cap S_i$ of a region R of U. Moreover, U has the ESSP if for all events $e \in E(U)$ and all states $s \in S(U)$ with $\neg(s \xrightarrow{e})$ there is a region R of U that inhibits e at s, that is, $R(s) = 0$ and $sig_R(e) = -1$. Again, U is feasible if U has the SSP and ESSP.

To merge a union $U = U(A_0, \ldots, A_n)$, we define its *joining* $A(U)$ as the following concatenation of the independent TSs in the joined linear TS

$$A(U) = A_0 \xrightarrow{y_1^1} z^1 \xrightarrow{y_2^1} A_1 \xrightarrow{y_1^2} z^2 \xrightarrow{y_2^2} \cdots \xrightarrow{y_1^n} z^n \xrightarrow{y_2^n} A_n$$

which additionally uses $2n$ distinct connector events $y_1^1, y_2^1, \ldots, y_1^n, y_2^n$ that are not in $E(U)$ and n distinct connector states $z^1, \ldots, z^n$ that are not in $S(U)$. As $A(U)$ preserves the manifoldness of all events in $E(U)$ and as all connector events occur at just one state we can conclude that a 3-fold union U leads to a linear 3-fold TS $A(U)$. The following lemma justifies the replacement of bulky composite TSs with modular unions:

Lemma 1. *If $U = U(A_0, \ldots, A_n)$ is the union of linear TSs $A_0, \ldots, A_n$ and $A = A(U)$ is their joining then U has the SSP (ESSP) if and only if A has the SSP (ESSP).*

Before we can prove Lemma 1 we need the following statement:

Lemma 2. *If $\mathcal{R} \subseteq \mathcal{R}(A)$ is a witness for the ESSP of $A = s_0 \xrightarrow{e_1} \cdots \xrightarrow{e_t} s_t$ then it is also a witness for the SSP of A.*

Proof. Let $\mathcal{R} \subseteq \mathcal{R}(A)$ be a witness for the ESSP of A and, for a contradiction, assume that there is a subsequence $s_i \xrightarrow{e_{i+1}} \cdots \xrightarrow{e_j} s_j \xrightarrow{e_{j+1}} \cdots \xrightarrow{e_t} s_t$ of A such that s_i, s_j are not separable by any region of $\mathcal{R}$. If e_{i+1} and e_{j+1} are not the same event then the region $R \in \mathcal{R}$ that inhibits e_{i+1} at s_j has $sig_R(e_{i+1}) = -1$, implying $R(s_i) = 1$, and $R(s_j) = 0$, a contradiction. As each event has to be inhibit at the terminal state this also implies that s_j is different from s_t.

Hence, let $k \geq 1$ be the greatest integer satisfying $i + k \leq j$ such that each of the subsequences $s_i \xrightarrow{e_{i+1}} \ldots \xrightarrow{e_{i+k}} s_{i+k}$ and $s_j \xrightarrow{e_{j+1}} \ldots \xrightarrow{e_{j+k}} s_{j+k}$ of A are built on the same series of events, that is, $e_{i+\ell} = e_{j+\ell}$ for all $\ell \in \{1, \ldots, k\}$.

If $i + k < j$ then the event $e_{i+k+1} \neq e_j$ differs from e_{j+k+1}. The region $R \in \mathcal{R}$ that inhibits the event e_{i+k+1} at s_{j+k} has $sig_R(e_{i+k+1}) = -1, R(s_{i+k}) = 1, R(s_{j+k}) = 0$. If $R(s_j) = 0$, Observation 1 implies that

$$R(s_{j+k}) - R(s_j) = \sum_{\ell=j+1}^{j+k} sig_R(e_\ell) = 0 = \sum_{\ell=i+1}^{i+k} sig_R(e_\ell) = R(s_{i+k}) - R(s_i)$$

and, consequently, that $R(s_i) = 1$, a contradiction. If $R(s_j) = 1$ the same conclusions lead to $R(s_{i+k}) - R(s_i) = -1$, which makes $R(s_i) = 2$ and is impossible.

Hence, we have to assume that $s_{i+k} = s_j$. In this case, Observation 1 can be used to infer for every region R that $R(s_j) - R(s_i) = R(s_{j+k}) - R(s_j)$ and, thus, $R(s_{j+k}) + R(s_i) = 2R(s_j)$. Hence, $R(s_{j+k}) + R(s_i)$ is even, which, for the binary function R, means $R(s_{j+k}) = R(s_i)$. If the event e_{i+1} does not occur at s_{j+k} then there is no region R inhibiting e_{i+1} at s_{j+k}. Such a region would have $sig_R(e_{i+1}) = -1, R(s_i) = 1, R(s_{j+k}) = 0$, which leads to the contradiction $R(s_{j+k}) \neq R(s_i)$.

Consequently, $e_{j+k+1} = e_{i+1}$ and the sequence $e_{i+2} \ldots e_j e_{j+1} = e_{i+2} \ldots e_j e_{i+1}$ occurring at s_{i+1} equals the the sequence $e_{j+2} \ldots e_{j+k} e_{j+k+1} = e_{j+2} \ldots e_{j+k} e_{i+1}$ occurring at s_{j+1}. But then again, Observation 1 provides $R(s_{j+1}) - R(s_{i+1}) = R(s_{j+k+1}) - R(s_{j+1})$. By the same argumentation, we get $R(s_{j+k+1}) = R(s_{i+1})$ and, by the assumption that e_{i+2} does not occur at s_{j+k+1}, we get $R(s_{j+k+1}) \neq R(s_{i+1})$. This means $e_{j+k+2} = e_{i+2}$. This argument can be continued at most until s_t is reached where the assumption that $s_{i+k} = s_j$ finally fails. □

Armed with the result that the ESSP of a linear TS implies the SSP, we can show the validity of using unions:

Proof of Lemma 1. If: Projecting a region separating s and s', respectively inhibiting e at s, in $A(U)$ to the component TSs yields a region separating s and s', respectively inhibiting e at s in U. Hence, the (E)SSP of $A(U)$ trivially implies the (E)SSP of U.

Only if: A region R of U separating s and s', respectively inhibiting e at s, can be completed to become an equivalent region of $A(U)$ by setting

$$R(z^i) = 0, sig_R(y_1^i) = -R(s_{t_{i-1}}^{i-1}), \text{ and } sig_R(y_2^i) = R(s_0^i)$$

for all $i \in \{1, \ldots, n\}$. Notice that R also inhibits e at all connector states. Hence, constructing one region for every event as demonstrated inhibits all events at all connector states.

For the (E)SSP of $A(U)$ it is subsequently sufficient to analyze (event) state separation concerning the connector states (events). It is easy to see, that each connector state z^i defines a region $R^i = \{z^i\}$ of $A(U)$ that separates z^i from all

other states in $A(U)$. This already implies that $A(U)$ has the SSP if U has the SSP. Moreover, the region R^i also inhibits y_2^i at all required states of $A(U)$.

To show that $A(U)$ has the ESSP in case that U has the ESSP, it is only left to argue for all $i \in \{1, \ldots, n\}$ that y_1^i can be inhibited at all appropriate states. Firstly, the set S_{i-1} of states from component A_{i-1} is a region of $A(U)$ that makes sure that y_1^i is inhibited at all required states in $S(U) \setminus S_{i-1}$.

Secondly, the event y_1^i can be inhibited at any state $s \in S_{i-1}$ as follows: As U has the ESSP, there is a subset $\mathcal{R} \subseteq \mathcal{R}(U)$ of all regions of U witnessing the ESSP for U. Hence, for all $i \in \{0, \ldots, n\}$ the set $\mathcal{R}(A_i) \cap \mathcal{R}$ witnesses that A_i has the ESSP and, by Lemma 2, witnesses the SSP of A_i, too. Subsequently, U has the SSP, which, by the argumentation above, means that $A(U)$ has the SSP. Consequently, there is a region R of $A(U)$ separating s and $s_{t_{i-1}}^{i-1}$, that is, where $R(s) = 0$ and $R(s_{t_{i-1}}^{i-1}) = 1$. If $R(z^i) = 0$ then R already inhibits y_1^i at s. Otherwise, as each connector event is unique in $A(U)$, we simply get what we need by removing z^i from R to get a region $R' = R \setminus \{z^i\}$ that behaves like R except for $R'(z^i) = 0$, $sig_{R'}(y_1^i) = -1$ and $sig_{R'}(y_2^i) = R'(s_0^i)$. □

Notice that the connector states are *crucial* for the successful transfer of ESSP from U to $A(U)$. Without the additional degree of freedom, that they provide, it would be generally impossible to solve in $A(U)$ some of the ESSP problems which originate from the subsequences that represent the individual TSs of the union.

4 The Hardness of ESSP and Feasibility

This section starts with ESSP and shows that it remains a hard problem even if the input is restricted to linear 3-fold TSs:

Thoerem 1. *To decide the ESSP for linear 3-fold transition systems is NP-complete.*

By Lemma 2 the ESSP implies the SSP on linear TSs. Hence, if a linear TS A has the ESSP then it is immediately feasible. Reversely, if A is feasible then, by definition, it also has the ESSP. Consequently, on linear TSs both properties are equivalent and therefore the following theorem is a corollary of Theorem 1:

Thoerem 2. *Feasibility of linear 3-fold transition systems is NP-complete.*

To prove Theorem 1, we present a polynomial time reduction of a cubic monotonic set φ of boolean 3-clauses to a 3-fold union U^φ of linear TSs such that φ has a one-in-three model M if and only if U^φ has the ESSP. As deciding the existence of M is NP-complete [10] and as it is known from [3] that deciding the ESSP is in NP the announced construction of U^φ together with the discussion of $A(U)$ in Sect. 3 proves the theorem.

In compliance to [10], φ is a set $\{K_0, \ldots, K_{m-1}\}$ of m clauses, each a set $K_i = \{A_i, B_i, C_i\}$ of exactly three distinct elements from $V(\varphi)$, the set of all

boolean variables in φ. Moreover, every variable occurs in exactly three clauses of φ which implies $|V(\varphi)| = m$. A one-in-three model M of φ is a subset of the variables $V(\varphi)$ such that $|M \cap K_i| = 1$ for all $i \in \{0, \ldots, m-1\}$.

The development of union $U^\varphi = U(B, T)$ is divided into the subunions B and T. Basically, B provides all *basic* components for the translation of one-in-three satisfiability to ESSP. It implements a single key ESSP instance, that is, a key event k which is inhibited at a certain key state of B by a unique region R^B. By design, R^B fixes a negative signature for an event series called the key copies.

In T, the fixed signature of key copies is used for the actual *translation* of one-in-three satisfiability to ESSP. In fact, a region R of U^φ inhibiting k at the key state has to extend the unique region R^B of B by a region R^T of T that has a consistent signature for all events shared by B and T. As the only shared events are key copies, R^T inherits their negative signature from R^B.

Next, T applies the exiting key copies to make sure that R^T exists if and only if φ has a one-in-three model. To this end, T encodes all variables $X \in V(\varphi)$ as an event $X \in E(T)$ and every clause $K_i = \{A_i, B_i, C_i\}$ is implemented by a *translator* union T_i. In T_i, the three events of K_i are arranged in such a way that, if the key copies exit then exactly one of A_i, B_i, C_i has a positive signature while the other two obey. As this happens simultaneously for all $i \in \{0, \ldots, m-1\}$, there is a region $R^B \cup R^T$ that inhibits k at the key state if and only if exactly one event A_i, B_i or C_i enters in every translator T_i if and only if a variable subset $M \subseteq V(\varphi)$ intersects every clause $K_i = \{A_i, B_i, C_i\}$ in exactly one element if and only if M is a one-in-three model of φ. Finally, the construction of U^φ makes sure that if k is inhibitable at the key state then all other events are inhibitable at all other required states, too.

The behavior of B and T is created by several gadget TSs. A single *master* M provides the key event and the key state. Next, there are $6m$ *refreshers* F_j and $6m$ *duplicators* D_j that generate the negative signature of all key copies. Hence, $B = U(M, F_0, \ldots, F_{6m-1}, D_0, \ldots, D_{6m-1})$ is a union of altogether $12m + 1$ TSs. The union $T = U(T_0, \ldots, T_{m-1})$ consists of the m *translators*, each a union $T_i = U(T_{i,0}, T_{i,1}, T_{i,2})$ of three linear TSs. To create a complete picture of our reduction, we subsequently introduce the details of all these gadget TSs. We like to point out that this does not explain the role of all used events. Some of them are only included to enable the model-independent inhibition of event state pairs that are not the key instance. This applies especially to the helper and zero events h, h_j, z_{2j+1} with $j \in \{0, \ldots, 6m-1\}$.

Master. M is a linear TS providing the key event k and the key state m_6. Figure 1(a) defines M and shows R^M the part of region R^B belonging to M. Notice that the region R^M that inhibits k at m_6 is unique. The so-called *opposites* o_0, o_1 enter and the *zero* z_0 obeys. These events initialize the setup of the key copies' negative signature. The subsequent refreshes and duplicators implement this synchronization in an assembly line fashion.

Refreshers. F_j are linear TS that consume the opposites o_{2j}, o_{2j+1} to generate key copies $k_{3j}, k_{3j+1}, k_{3j+2}$. More precisely, for all $j \in \{0, \ldots, 6m-1\}$ refresher F_j takes the two previously prepared opposites o_{2j}, o_{2j+1} and forces a negative

signature onto the key copies $k_{3j}, k_{3j+1}, k_{3j+2}$. This consumes both remaining applications of o_{2j}, o_{2j+1} such that no further opposites are available at this point. The definition of F_j together with its fraction R^{F_j} of region R^B is given in Fig. 1(b). It is easy to see that, if the input opposites enter, there is no other way to choose R^{F_j} and that this leads to exiting $k_{3j}, k_{3j+1}, k_{3j+2}$.

Duplicators. D_j are linear TSs that provide the next opposites o_{2j+2}, o_{2j+3} and zero z_{2j+2}. To this end, they eat up the remaining two applications of k_{3j}, k_{3j+1} and one of the remaining applications of k_{3j+2} provided by refresher F_j, as well as one occurrence of z_{2j}. The generated opposites o_{2j+2}, o_{2j+3} and the zero z_{2j+2} prepare the work of F_{j+1} and D_{j+1}. Main result of step j, however, is the one remaining application of the key copy k_{3j+2} with negative signature. Therefore, the whole process chain produces $6m$ key copies $k_2, k_5, \ldots, k_{18m-1}$, each of them free to be applied one more time. Figure 1(c) introduces D_j as well as the part R^{D_j} of region R^B. Again, it is easy to verify that, if the input key copies exit and z_{2j} obeys, R^{D_j} is the only possible region of D_j. Note that, as the last needed key copy k_{18m-1} is already produced by F_{6m-1}, duplicator D_{6m-1} is actually not required. But for a unique case study it is more convenient to keep it the construction.

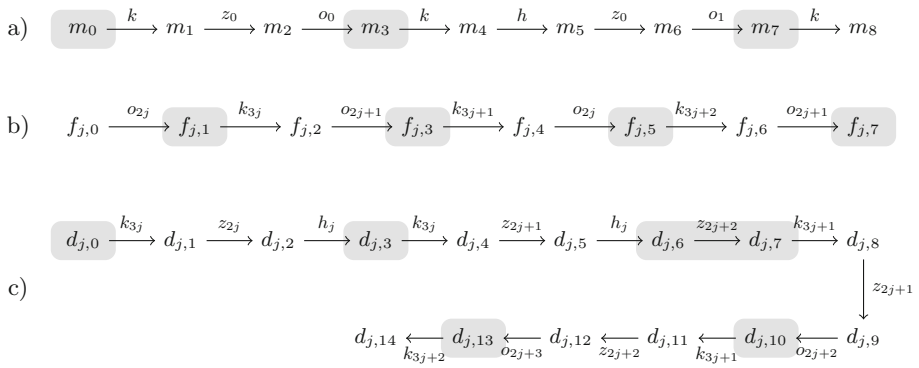


Fig. 1. Gadgets of union B together with their parts of the region R^B. States in the respective region are shown with a gray background. (a) M with R^M, (b) F_j with R^{F_j}, (c) D_j with R^{D_j}.

The following lemma summarizes the functionality of union B:

Lemma 3. *Let $R^M = \{m_0, m_3, m_7\}$ and, for all $j \in \{0, \ldots, 6m-1\}$, let $R^{F_j} = \{f_{j,1}, f_{j,3}, f_{j,5}, f_{j,7}\}$ and $R^{D_j} = \{d_{j,0}, d_{j,3}, d_{j,6}, d_{j,7}, d_{j,10}, d_{j,13}\}$. Except for the complement, the set of states*

$$R^B = R^M \cup R^{D_0} \cup \ldots R^{D_{6m-1}} \cup R^{F_0} \cup \cdots \cup R^{F_{6m-1}}$$

is the only region of B that inhibits k at m_6. For all $i \in \{0, \ldots, 18m-1\}$ the signature of the key copy k_i is exiting, that is, $sig_{R^B}(k_i) = -1$.

Proof. Figure 1(a) shows that R^M is a region of M that inhibits k at m_6 and that $R^{F_j} \cup R^{D_j}$ yields a region of $F_j \cup D_j$ where the key copies exit, the zeros obey and the opposites enter. We further observe, that for $i \neq j$ the unions $F_i \cup D_i$ and $F_j \cup D_j$ have at most zeros and opposites in common and with respect to these events the signature of the regions $R^{F_i} \cup R^{D_i}$ and $R^{F_j} \cup R^{D_j}$ are equal. Furthermore, the master M shares events only with F_0, D_0, namely o_0, o_1 and z_0. With respect to these events the signatures of R^M and $R^{F_0} \cup R^{D_0}$ are equal. Altogether, we conclude that R^B is a region of B that inhibits k at m_6 where all key copies exit.

Reversely, let R be a region of B that inhibits k at m_6. Without loss of generality we can assume that $R(k) = -1$ and $R(m_6) = 0$. Firstly, we show that $R \cap M = R^M$. From $sig_R(k) = -1$ we obtain $R(m_0) = R(m_3) = R(m_7) = 1$ and $R(m_1) = R(m_4) = R(m_8) = 0$. From $R(m_1) = 0$ and $R(m_6) = 0$ we conclude $sig_R(z_0) = 0$ which yields $R(m_2) = R(m_5) = 0$. That is $R \cap M = R^M$.

Now we show, that $R \cap F_j = R^{F_j}$ and that $R \cap D_j = R^{D_j}$. To this end, we initially argue that R^{F_j} is the only region of F_j such that all opposite events enter. Figure 1(b) proves R^{F_j} to be a region with announced event behavior. Observe, that R^{F_j} forces all key copies to exit. Let R' be a region of F_j such that all opposite events enter. It follows immediately that $R'(f_{j,1}) = R'(f_{j,3}) = R'(f_{j,5}) = R'(f_{j,7}) = 1$ and $R'(f_{j,0}) = R'(f_{j,2}) = R'(f_{j,4}) = R'(f_{j,6}) = 0$ which results in $R' = R^{F_j}$.

We further show, that R^{D_j} is the only region of D_j where all key copies exit and the event z_{2j} obeys. Again, Fig. 1(c) proves R^{D_j} to behave as announced. Now, let R' be a region of D_j where all key copies exit and z_{2j} obeys. We obtain $R'(d_{j,\ell}) = 1$ for $\ell \in \{0, 3, 7, 10, 13\}$ and $R'(d_{j,\ell}) = 0$ for $\ell \in \{1, 4, 8, 11, 14\}$. From $R'(d_{j,1}) = 0$ and $sig_{R'}(z_{2j}) = 0$ we have $R'(d_{j,2}) = 0$ which, together with $R'(d_{j,3}) = 1$, yields $sig_{R'}(h_j) = 1$. From $sig_{R'}(h_j) = 1$ we conclude $R'(d_{j,5}) = 0$ and $R'(d_{j,6}) = 1$ which, together with $R'(d_{j,4}) = 0$ and $R'(d_{j,7}) = 1$, yields $sig_{R'}(z_{2j+1}) = sig_{R'}(z_{2j+2}) = 0$. With $sig_{R'}(z_{2j+1}) = sig_{R'}(z_{2j+2}) = 0$ and $R'(d_{j,8}) = R'(d_{j,11}) = 0$, we obtain $R'(d_{j,9}) = R'(d_{j,12}) = 0$. This means $R' = R^{D_j}$.

Finally, with $R \cap M = R^M$ implying $sig_R(o_0) = sig_R(o_1) = 1$ and $sig_R(z_0) = 0$ we inductively obtain that $R \cap F_j = R^{F_j}$ and that $R \cap D_j = R^{D_j}$ for all $j \in \{0, \ldots, 6m-1\}$. This results in $R = R^B$. □

After B has finished the job, we have $6m$ key copies with one free application, namely the events k_{3j+2} for all $j \in \{0, \ldots, 6m-1\}$. In the construction of T, we need a sequence of six free key copies for every translator T_i. Consequently, we assign to T_i the events $k_{18i+2+3\ell}$ for all $\ell \in \{0, \ldots, 5\}$. We continue with the description of our translators:

Translators. $T_i = U(T_{i,0}, T_{i,1}, T_{i,2})$ are unions that represent the clauses $K_i = \{A_i, B_i, C_i\}$ of φ. Figure 2 defines the three linear TSs $T_{i,0}, T_{i,1}, T_{i,2}$ of T_i and presents $R^{T_i}_{B_i}$, a possible part of a region of T.

A negative signature of the key copies makes sure that exactly one of the variable events A_i, B_i, C_i gets a positive signature while the other two obey.

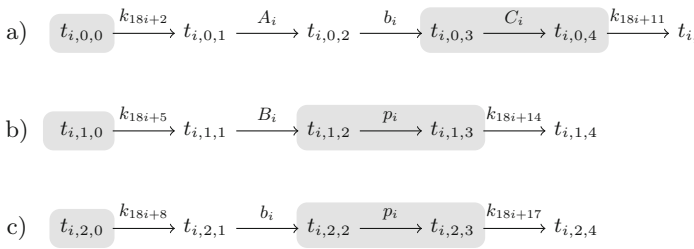


Fig. 2. The translators (a) $T_{i,0}$, (b) $T_{i,1}$, (c) $T_{i,2}$ with region $R^{T_i}_{B_i}$, one of three possibilities in case of exiting key copies. Here just B_i has a positive signature while A_i, C_i obey.

In fact, $R^{T_i}_{B_i}$ selects the event B_i and there are two other regions $R^{T_i}_{A_i}$ and $R^{T_i}_{C_i}$ of T_i for the selection of A_i, respectively C_i.
The TSs $T_{i,1}$ and $T_{i,2}$ create a copy of B_i, namely the event b_i that unlike B_i occurs only in the translator T_i, and guarantees that the signature of both events cannot be negative. To achieve this, both TSs surround a sequence, B_i, p_i or, respectively, b_i, p_i, with key copies. As the proxy event p_i and the key copies behave equally in both TSs, B_i and b_i have to be equal, too. Moreover, the negative signature of the key copies makes sure that their neighboring events B_i, b_i, p_i cannot exit.
The TS $T_{i,0}$ is simply the event sequence A_i, b_i, C_i surrounded by key copies. Again, the negative signature of key copies prevents a negative signature for their neighboring events, A_i and C_i. The exiting key copies also imply that the signature of the event sequence A_i, b_i, C_i has to add up to one. Hence, by the equality of b_i and B_i, exactly one of A_i, B_i, C_i enters.

Lemma 4. *Let R^T be a region of T where all contained key copies exit, that is, where $sig_R(k_{3j+2}) = -1$ for all $j \in \{0, \ldots, 6m-1\}$. Then $R^T = \bigcup_{i=0}^{m-1} R^{T_i}$ with R^{T_i} being one of*

$$R^{T_i}_{C_i} = \{t_{i,0,0}, t_{i,0,4}, t_{i,1,0}, t_{i,1,3}, t_{i,2,0}, t_{i,2,3}\},$$
$$R^{T_i}_{B_i} = R^{T_i}_{C_i} \cup \{t_{i,0,3}, t_{i,1,2}, t_{i,2,2}\},$$
$$R^{T_i}_{A_i} = R^{T_i}_{C_i} \cup \{t_{i,0,2}, t_{i,0,3}\}$$

for all $i \in \{0, \ldots, m-1\}$. Hence, for all clauses $K_i = \{A_i, B_i, C_i\}$ there is exactly one variable event $X_i \in K_i$ such that $sig_{R^T}(X_i) = 1$.

Proof. We prove the lemma in the following steps: Firstly, we show for all $i \in \{0, \ldots, 6m-1\}$ that R being a region of T_i where all key copies exit implies that exactly one event of $\{A_i, B_i, C_i\}$ enters and the others obey. Secondly, we show that R has to be equal to one of $R^{T_i}_{C_i}, R^{T_i}_{B_i}$, and $R^{T_i}_{A_i}$. Thirdly, we argue that R^T behave as announced.

Assume R to be a region of T_i such that all key copies exit. Then it follows immediately that $R(t_{i,1,1}) = R(t_{i,2,1}) = 0$ and $R(t_{i,1,3}) = R(t_{i,2,3}) = 1$. Hence,

by Observation 1 we have $sig_R(B_i) + sig_R(p_i) = sig_R(b_i) + sig_R(p_i) = 1$ implying $sig_R(B_i) = sig_R(b_i) = 1 - sig_R(p_i)$ which by $sig_R(p_i) \in \{-1, 0, 1\}$ again implies that $sig_R(B_i)$ and $sig_R(b_i)$ are nonnegative. Furthermore, by the exiting key copies, on the one hand, we have $R(t_{i,0,1}) = 0$ and $R(t_{i,0,4}) = 1$ implying that $sig_R(A_i)$ and $sig_R(C_i)$ are nonnegative. On the other hand, Observation 1 provides $sig_R(A_i) + sig_R(b_i) + sig_R(C_i) = R(t_{i,0,4}) - R(t_{i,0,1}) = 1$ which requires that exactly one of A_i, b_i, C_i enters. Altogether we have shown that exactly one event of $\{A_i, B_i, C_i\}$ enters and the others obey.

Now let R be a region of T_i such that all key copies exit. Then R includes the sources of the key copies and excludes their sinks. If $sig_R(C_i) = 1$, by $sig_R(A_i) = sig_R(b_i) = sig_R(B_i) = 0$, we immediately obtain that $R = R^{T_i}_{C_i}$. If $sig_R(B_i) = sig_R(b_i) = 1$ and $sig_R(A_i) = sig_R(C_i) = 0$ we have $R(t_{i,0,3}) = R(t_{i,1,2}) = R(t_{i,2,2}) = 1$, that is, $R = R^{T_i}_{B_i}$. If $sig_R(A_i) = 1$, $sig_R(C_i) = sig_R(b_i) = sig_R(B_i) = 0$, we obtain $R(t_{i,0,2}) = R(t_{i,0,3}) = 1$ and $R(t_{i,2,2}) = 0$ implying $R = R^{T_i}_{A_i}$.

Finally, if R^T is a region of T such that all key copies exit, we have argued that $R^T \cap T_i \in \{R^{T_i}_{C_i}, R^{T_i}_{B_i}, R^{T_i}_{A_i}\}$. This completes the proof. □

The following lemma establishes the foundation for the correctness of our reduction:

Lemma 5. *The union U^φ can inhibit the key event k at the key state m_6 if and only if φ has a one-in-three model.*

Proof. If: If M is a one-in-three model of φ, we firstly define a region $R^T = \bigcup_{i=0}^{m-1} R^{T_i}_{X_i}$ of T by adding $R^{T_i}_{X_i}$ for every T_i where $X_i = M \cap K_i$ is the one variable A_i, B_i or C_i of K_i covered by M. By definition, this lets all contained key copies exit. By Lemma 3 the region R^B of B inhibits k at m_6 and lets all key copies exit, too. As key copies are the only events shared by B and T, the two regions are compatible and $R^B \cup R^T$ is a region of U^φ inhibiting k at m_6.
Only if: If R is a region of U^φ that inhibits k at m_6 then Lemma 3 states that, without loss of generality, R contains R^B as subregion for B. This implies that all key copies exit. By Lemma 4, *every* $i \in \{0, \ldots, m-1\}$ *exactly* defines one variable event $X_i \in K_i = \{A_i, B_i, C_i\}$ that has $sig_R(X) = 1$, while $sig_R(Y) = 0$ for the other two $Y \in K_i \setminus X$. This yields a one-in-three model $M = \{X \mid X \in V(\varphi), sig_R(X) = 1\}$. □

Having established the connection between the key region and the original satisfiability problem, it remains to show for all other combinations of event e and state s with $\neg(s \xrightarrow{e})$ that e can be inhibited at s. This is done by one lemma for every event e of U^φ asserting the inhibition of e at every required state of U^φ. To show the lemmas, respective proofs provide tables with four columns *States, Exit, Enter* and *Affected TSs*, each. Every row of such a table defines a region that inhibits e at the states listed in the corresponding *States* column and at all states of TSs of the union that do not occur in the associated *Affected TSs* column. The *Exit* (*Enter*) cell shows the events with negative (positive)

signature of the respective region. Moreover, *Affected TS* lists the TSs of U^{φ} having at least one event that participates in *Exit* or *Enter*.

That the regions given in a proof's table inhibit e at all TSs that are not affected, is derived from the following lemma introducing a mechanism that we call *region enhancement*:

Lemma 6 (Region Enhancement). *Let $U = U(A_1, \ldots, A_m)$ be a union, R be a region of U, and $A_{m+1}, \ldots, A_n$ be linear TSs such that for all $i \in \{m+1, \ldots, n\}$ there is at most one arc $s_i \xrightarrow{e_i} s_i'$ in A_i where $sig_R(e_i)$ is defined and $sig_R(e_i) \neq 0$. Moreover, for all $i \in \{m+1, \ldots, n\}$, let P_i be the set of s_i and all its predecessor states and let $P_i = \emptyset$ in case s_i does not exist. Then there is an enhanced region $R' = R \cup R^{m+1} \cup \cdots \cup R^n$ of union $U(A_1, \ldots, A_m, A_{m+1}, \ldots, A_n)$ with $sig_{R'} = sig_R$ such that, for all $i \in \{m+1, \ldots, n\}$, the region is enhanced by $R^i = S_i \setminus P_i$ if $sig(e_i) = 1$, where S_i is the state set of A_i, and otherwise, the region is enhanced by $R^i = P_i$.*

The proof of Lemma 6 is straightforward and therefore omitted. The following lemma gives an example for the announced proof mechanism by showing that the key event can be inhibited at all states of the union:

Lemma 7. *The key event k is inhibitable in U^{φ}.*

Proof. By the key region, the key event is already inhibited in the entire master and in the duplicator D_0 except for the states $d_{0,0}, d_{0,3}, d_{0,6}, d_{0,7}, d_{0,10}, d_{0,13}$. Furthermore, it remains to show, that k is inhibitable in all the other TSs:

States	Exit	Enter	Affected TS
$d_{0,0}, d_{0,3}, d_{0,6}, d_{0,7}$	k, h_0	z_0, z_1	M, D_0
$d_{0,10}, d_{0,13}$	k, k_0	z_0	M, F_0, D_0

□

Hence, except for the key region, two additional regions are needed to inhibit k in U^{φ}. For instance, the first row of the given table represents a region R where k, h_0 exit, z_0, z_1 enter and all other events obey. The non-obeying events affect, that is, occur only in $U(M, D_0)$. This is used in region enhancement from Lemma 6 to extend R for a region of the entire union U^{φ}. By R the key event k is inhibited at $d_{0,0}, d_{0,3}, d_{0,6}, d_{0,7}$ and at all states of TSs different from M, D_0.

Applying this template for all other states finally yields the inhibition of all events at all states. However, this is fairly tedious work and not too much enlightening. Therefore, we have shifted the remaining lemmas of this kind to [11]. Since furthermore the polynomial running time of our reduction is obvious, we consider the proof of Theorem 1 completed.

5 The Hardness of SSP

If SSP was of a lesser complexity than ESSP, it could serve as a fast fail preprocess for feasibility. That means, if a linear TS fails the efficient SSP test in question one could save the costly ESSP test. However, this possibility is ruled out by a polynomial time reduction of linear 3-ESSP to linear 3-SSP provided in this section:

Thoerem 3. *To decide the SSP for linear* 3*-fold transition systems is NP-complete.*

Given a linear 3-fold TS $A = (S, E, \delta, s_0)$, our reduction creates a separate union U_s^e of linear 3-fold TSs for every pair of event $e \in E$ and state $s \in S$ that do not fulfill $s \xrightarrow{e}$. By construction, U_s^e has two key states that are separable if and only if e can be inhibited at s in A. Then, as the unions get mutually disjoint state and event sets by design, they can be smoothly merged into an aggregate union $U^A = U(U_s^e \mid e \in E, s \in S, \neg(s \xrightarrow{e}))$. If U^A has the SSP then the key states of U_s^e are separable for all relevant (e, s), which implies the ESSP for A. Reversely, if A has the ESSP, we use Lemma 2 to get the SSP for U^A. Finally, we join the TSs of U^A forming $A' = A(U^A)$ and, by Lemma 1, we get that A' has the SSP if and only if U^A has the SSP if and only if A has the ESSP.

Having an outline of the primal reduction approach, the remainder of this section focuses on the introduction of U_s^e for event $e \in E$ and state $s \in S$ failing $s \xrightarrow{e}$. Basically, U_s^e installs a TS M with two key states m_0, m_1 and a TS C representing a copy of A such that, effectively, m_0 and m_1 can be separated only by a key region R where $sig_R(e) = -1$ and $R(s) = 0$. Hence, the separability of m_0 and m_1 implies that e is inhibitable at s and vice versa. One difficulty with this idea is to get along with just three assignments of event e. The union solves this by including additional TSs that exploit properties of a key region to copy the signature of e to other events subsequently serving as replacements for e.

Actually, the approach to copy the behavior of an event into other events is similar to our approach in Sect. 4. Lemma 3 shows, that the copy functionality of the refreshers and the duplicators in the region R^B depends crucially on the determined signature of o_0, o_1 *and* z_0. Unfortunately, the restriction on linear 3-fold TSs heavily complicates the use of state separation to determine the signature of these three events. Therefore, we refrain from reusing the gadgets of Sect. 4. However, the following reduction uses events with names and functions similar to the respective counterparts in Sect. 4. In the following we go into the details of the used gadget TSs:

Mapper. M is a linear TS on states $\{m_0, \ldots, m_5\}$ where the separation of m_0 and m_1 by a suitable region R^M makes sure that $e \in exit_{R^M}$. Figure 3(a) introduces M together with M's part of a key region, R^M. M uses the event e already thrice to force the first two opposites o_0 and o_1 into $enter_{R^M}$.

Duplicators. D_j are linear TSs on states $\{d_{j,0}, \ldots, d_{j,13}\}$ that, for a duplicator part R^{D_j} of a key region, are supposed to synchronize the signature of e with events $\{e_{2j}, e_{2j+1}\}$, so-called *e-copies*. In this way we work around the limitation of using e only three times. There are five duplicators $D_0, \ldots, D_4$ and each of them generates one e-copy $e_{2j+1}, j \in \{0, \ldots, 4\}$ that has a free assignment to be used somewhere else. As Fig. 3(b) demonstrates, every duplicator D_j exploits the previous opposites o_{2j}, o_{2j+1}, which are appointed to $enter_{R^{D_j}}$, to force a negative signature onto the e-copies and to synchronize two further opposites o_{2j+2}, o_{2j+3} to be used in the next duplicator. The *helpers* h_{2j}, h_{2j+1} help to solve other state separation instances different from the key instance.

Provider. P is a linear TS on states $\{p_0, \ldots, p_7\}$ which, for the provider part R^P of a key region, applies the negative signature of the e-copies e_7 and e_9 and the positive signature of the last two opposites o_{10} and o_{11} to provide *witness* events w_1 and w_2 with $sig_{R^P}(w_1) = 1$ and $sig_{R^P}(w_2) = -1$. The purpose of w_1 and w_2 is to enhance the following copy of A in such a way that s is guaranteed to be outside a key region. See Fig. 3(c) for a definition of P and R^P.

Copy. C is a linear TS that basically copies A into the union. However, C replaces every of the at most three occurrences of e by a free e-copy e_1, e_3, e_5. Also, s is enhanced by two edges $s \xrightarrow{w_1} p \xrightarrow{w_2} s'$. If there was an edge $s \xrightarrow{e'} q$ in A we let C continue with this edge after s', that is, $s' \xrightarrow{e'} q$. In C's part R^C of a key region, the e-copies inherit the negative signature of e. Moreover, we get $sig_{R^C}(w_1) = 1$ and $sig_{R^C}(w_2) = -1$ which means that neither s nor s' are in R^C. Combining these two facts, we get a slightly modified version of R^C that can be used as a region of A to inhibit e at s.

Altogether, the construction of U_s^e results in $U(M, D_0, \ldots, D_4, P, C)$. Notice that the same construction scheme generates unions U_s^e for multiple instances (e, s). Although not explained in detail at this point, we later enhance the construction by a renaming mechanism that prevents possible state or event clashes by enhancing the used state and event names with a unique identifier of the union U_s^e they occur in.

The correctness of the given reduction is based on the following argumentation. Firstly, the following lemma formalizes that the separation of key states implies a unique key region that inhibits e at s:

Lemma 8. *If R is a region of U_s^e with $m_0 \in R$ and $m_1 \notin R$ then*

1. $R \cap \{m_0, \ldots, m_5\} = R^M = \{m_0, m_2, m_4\}$,
2. $R \cap \{d_{j,0}, \ldots, d_{j,13}\} = R^{D_j} = \{d_{j,1}, d_{j,3}, d_{j,5}, d_{j,7}, d_{j,8}, d_{j,10}, d_{j,12}\}$ *for all* $j \in \{0, \ldots, 4\}$,
3. $R \cap \{p_0, \ldots, p_5\} = R^P = \{p_0, p_2, p_5, p_7\}$,
4. $sig_R(e) = sig_R(e_0) = \cdots = sig_R(e_9) = -1$,
5. $sig_R(o_0) = \cdots = sig_R(o_{11}) = 1$,

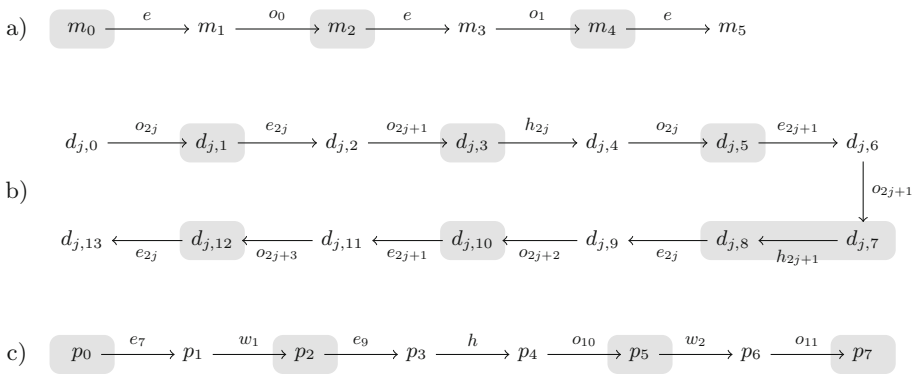


Fig. 3. Gadgets of union U_s^e together with their parts of a key region. (a) M with R^M, (b) D_j with R^{D_j}, (c) P with R^P.

6. $sig_R(w_1) = 1$*, and* $sig_R(w_2) = -1$*, and*
7. the set $R' = R \cap S$ *is a region of* A *inhibiting* e *at* s*.*

Proof. By $m_0 \in R$ and $m_1 \notin R$ it is easy to see that $sig_R(e) = -1$, which implies $m_2, m_4 \in R$ and $m_1, m_3, m_5 \notin R$ and, thus, $sig_R(o_0) = sig_R(o_1) = 1$. Iterating through the duplicators for $j \in \{0, \ldots, 4\}$, we get from $sig_R(o_{2j}) = sig_R(o_{2j+1}) = 1$ that $d_{j,0}, d_{j,2}, d_{j,4}, d_{j,6} \notin R$ and $d_{j,1}, d_{j,3}, d_{j,5}, d_{j,7} \in R$ and $sig_R(e_{2j}) = sig_R(e_{2j+1}) = -1$, which in turn means that $d_{j,8}, d_{j,10}, d_{j,12} \in R$ and $d_{j,9}, d_{j,11}, d_{j,13} \notin R$ and that $sig_R(o_{2j+2}) = sig_R(o_{2j+3}) = 1$. Finally, using $sig_R(e_7) = sig_R(e_9) = -1$ and $sig_R(o_{10}) = sig_R(o_{11}) = 1$ we get $p_0, p_2, p_5, p_7 \in R$ and $p_1, p_3, p_4, p_6 \notin R$ which means that $sig_R(w_1) = 1$ and $sig_R(w_2) = -1$.

To see the correctness of statement 7, recall that, by definition, R provides a (sub) region R^C for C. If C had not been modified with e-copies and $s \xrightarrow{w_1} p \xrightarrow{w_2} s'$ we basically could use R^C as a region of A. However, as $sig_R(e) = sig_R(e_1) = sig_R(e_3) = sig_R(e_5) = -1$, we get that the signature of e is negative and appropriately translated into R'. Moreover, by $sig_R(w_1) = 1$ and $sig_R(w_2) = -1$, we correctly have $s, s' \notin R$ which means that $s \notin R'$ and that, in the case $s \xrightarrow{e'} q$ exists, $sig_R(e') = sig_{R'}(e')$. □

Hence, as the part R^C basically is a region of A that inhibits e at s, the SSP for U^A, which includes the separability of the key states in U_s^e for all relevant (e, s), implies the ESSP of A.

Next, the following lemma makes sure that every region R of A can be translated into a meaningful region R' of U_s^e. Meaningful is to say that R' is a region that adopts the signature of A for the according events of C. We take care that the translation works in a way that forces as many events of U^A as possible to obey. This will simplify the conclusion of the SSP of U^A from the ESSP of A.

Lemma 9. *If R is region of A, then the set $R' = R^C \cup R^U$ with*

$$R^U = \begin{cases} \bigcup_{j=0}^{2} \{d_{j,6}, d_{j,7}, d_{j,11}, d_{j,12}, d_{j,13}\}, & \text{if } sig_R(e) = -1, \\ \emptyset, & \text{if } sig_R(e) = 0, \\ \bigcup_{j=0}^{2} \{d_{j,0}, \ldots, d_{j,5}, d_{j,8}, d_{j,9}, d_{j,10}\}, & \text{if } sig_R(e) = 1, \text{ and} \end{cases}$$

$$R^C = \begin{cases} R, & \text{if } s \notin R, \\ R \cup \{p, s'\}, & \text{otherwise}, \end{cases}$$

is a region of U_s^e where for all $x \in E \setminus \{e\}$ hold that $sig_R(x) = sig_{R'}(x)$ and $sig_R(e) = sig_{R'}(e_1) = sig_{R'}(e_3) = sig_{R'}(e_5)$ and $R(y) = R'(y)$ for all $y \in S$ and $R'(s) = R'(p) = R'(s')$.

Proof. We start by showing that $R^U = R' \cap S(U)$ is a region of the union $U = U(M, D_0, \ldots, D_4, P)$ and that $R^C = R' \cap S^C$ is a region of the TS C having state set S^C. Subsequently, we argue that the signatures sig_{R^U} and sig_{R^C} merge into an aggregate signature $sig_{R'}$ such that R' becomes a region of U_s^e.

For R^U it is easy to check that it admits the signature

$$sig_{R^U}(x) = \begin{cases} 0, & \text{if } x \in \{e, e_0, e_2, e_4, e_6, \ldots, e_9, w_1, w_2, o_0, \ldots, o_{11}, h, h_0, \\ & \qquad h_2, h_4, h_6, \ldots, h_9\}, \\ sig_R(e), & \text{if } x \in \{e_1, e_3, e_5\}, \\ -sig_R(e), & \text{if } x \in \{h_1, h_3, h_5\} \end{cases}$$

and therefore is a region of U.

Next, to show that R^C is a region of C is pretty straightforward as C is an enhanced copy of A and R^C is a superset of R, a region of A. It is sufficient to argue that, (1) with respect to events $E \setminus \{e\}$, the signature sig_R can be kept for R^C, (2) the e-copies e_1, e_3, e_5 inherit their signature from e, and (3) the signature of w_1 and w_2 is simply zero. Observe, s, p and s' are either all in R^C or all not in R^C such that a possible edge $s' \xrightarrow{e'} q$ in C behaves just like the original edge $s \xrightarrow{e'} q$ and therefore fulfills $sig_{R^C}(e') = sig_R(e')$. Hence, R^C admits the signature

$$sig_{R^C}(x) = \begin{cases} sig_R(x), & \text{if } x \in E \setminus \{e\}, \\ sig_R(e), & \text{if } x \in \{e_1, e_3, e_5\}, \\ 0, & \text{if } x \in \{w_1, w_2\} \end{cases}$$

and, consequently, is a region of C. Having R^U as a region of U and R^C as a region of C, it remains to show that $R' = R^U \cup R^C$ is a region of $U_s^e = U(U, C)$. This requires to show that the signatures sig_{R^U} and sig_{R^C} coincide on shared events and, thus, can be merged to a signature $sig_{R_s^e}$. However, by construction, the only events shared by U and C are e_1, e_3, e_5, w_1, w_2 which, by definition, have equal signatures in both regions. □

In the following argumentation, Lemma 9 is used to show that the ESSP of A, which by Lemma 2 also provides the SSP of A, implies the SSP of U^A, too. But before we aim for U^A, we show this property for U_s^e:

Lemma 10. *If A has the ESSP, then for every event $e \in E$ and state $s \in S$ with $\neg(s \xrightarrow{e})$ the union U_s^e has the SSP.*

Proof. In this proof, we again use the mechanism of *region enhancement* from Lemma 6 to show that all pairs of states of U_s^e are separable. States that originate from different TSs of U_s^e are separable by definition. Hence, we have to consider four cases, namely that both come from the same TS, either M, D_j, P, or C.

We start with M and in particular the key states m_0 and m_1. To separate them, we use $R^U = R^M \cup R^{D^0} \cup \cdots \cup R^{D^4} \cup R^P$, which is a region of union $U = U(M, D_0, \ldots, D_4, P)$ as easily verified by Fig. 3. Notice that $sig_{R^U}(e_1) = sig_{R^U}(e_3) = sig_{R^U}(e_5) = sig_{R^U}(w_2) = -1$ and $sig_{R^U}(w_1) = 1$. Moreover, as A has the ESSP, we can find a region R inhibiting e at s such that $sig_R(e) = -1$ and $R(s) = 0$ and enhance it to $R^C = R \cup \{p\}$. It is easy to see that R^C is a region of C with $sig_{R^C}(e_1) = sig_{R^C}(e_3) = sig_{R^C}(e_5) = sig_{R^C}(w_2) = -1$ and $sig_{R^C}(w_1) = 1$. Clearly, the regions are compatible on the events e_1, e_3, e_5, w_1, w_2 shared by U and C and we can combine them to the key region $R_{key} = R^U \cup R^C$ of U_s^e that separates the key states.

Moreover, in M the region R_{key} separates every state in $\{m_0, m_2, m_4\}$ from every state in $\{m_1, m_3, m_5\}$. Here, it remains to show that m_i and m_j are separable if they originate from the same of both sets, that is, if i and j are of the same parity. Consider the region $R_0 = \{m_0, m_1, d_{0,0}, d_{0,4}\}$ of U having just obeying events except for $sig_{R_0}(o_0) = -1$ and $sig_{R_0}(h_0) = 1$. As o_0 and h_0 do not occur in C, region R_0 can be enhanced to a region R_0' of $U_s^e = U(U, C)$. Obviously, R_0' separates all states in $\{m_0, m_1\}$ from all states in $\{m_2, \ldots, m_5\}$. Consider the region $R_1 = \{m_0, \ldots, m_3, d_{0,0}, d_{0,1}, d_{0,2}, d_{0,4}, d_{0,5}, d_{0,6}\}$ of U where again all events are obeying except for $sig_{R_1}(o_0) = -1$ and $sig_{R_1}(o_1) = 1$. The TS C does not contain o_0 and o_1 and, consequently, region R_1 can be enhanced to a region R_1' for U_s^e that separates m_2 from m_4 and m_3 from m_5.

Next, we consider two states s_1, s_2 of C. We firstly assume both states s_1 and s_2 to be in the state set S of A. Then, we use the fact that, by Lemma 2, A inherits the SSP from the ESSP. Let R be a region of A separating s_1 and s_2. By Lemma 9, we can use the according region R' that separates s_1 and s_2 in C.

If exactly one of the states, without loss of generality s_1, is in $S \setminus \{s\}$ and $s_2 \in \{p, s'\}$ then let R be a region of A separating s_1 and s. Again, we can enhance region R to a region R' that separates s_1 and s in C. As s, p and s' are not separated by this region, R' separates s_1 and s_2, too. The states s and p respectively s' and p are separated by the key region R_{key} introduced above. Finally, for s and s' we can generate a separating region of U_s^e as follows: We let $R^U = \{p_0, p_1\}$ and R^C is the set of states from S that are predecessors of or equal to s in A. It is easy to see that $sig_{R^U}(w_1) = sig_{R^C}(w_1) = -1$ and for all other events, both signatures are zero. Hence, the set $R^U \cup R^C$ is a region of U_s^e that contains s but not s'.

In the following, we investigate D_j for $j \in \{0, \ldots, 4\}$ as well as P. The key region R_{key} already separates every state of $\{d_{j,0}, d_{j,2}, d_{j,4}, d_{j,6}, d_{j,9}, d_{j,11}, d_{j,13}\}$ from every state of $\{d_{j,1}, d_{j,3}, d_{j,5}, d_{j,7}, d_{j,8}, d_{j,10}, d_{j,12}\}$ in D_j and in P every state of $\{p_0, p_2, p_5, p_7\}$ from every state of $\{p_1, p_3, p_4, p_6\}$. The argumentation for the remaining state pairs always works like already seen for M: We define a region R of U by specifying the signature of all non-obeying events, we observe that C has at most one edge labelled with such an event, we enhance region R to a region of $U_s^e = U(U, C)$, and we list two sets of states X and Y such that every state in X is separated from every state of Y by R:

	Signature	X	Y
R_2	$sig_{R_2}(h_{2j}) = -1$	$\{d_{j,0}, \ldots, d_{j,3}\}$	$\{d_{j,4}, \ldots, d_{j,13}\}$
R_3	$sig_{R_3}(h_{2j+1}) = -1$	$\{d_{j,4}, \ldots, d_{j,7}\}$	$\{d_{j,8}, \ldots, d_{j,13}\}$
R_4	$sig_{R_5}(e_{2j+1}) =$ $-sig_{R_5}(e_{2j}) = 1$	$\{d_{j,0}, d_{j,1}, d_{j,6}, \ldots, d_{j,8}, d_{j,11}, d_{j,12}\}$	$\{d_{j,2}, \ldots, d_{j,5}, d_{j,9}, d_{j,10}, d_{j,13}\}$
R_5	$sig_{R_6}(e_{2j+1}) =$ $-sig_{R_6}(h_{2j}) = 1$	$\{d_{j,8}, d_{j,9}\}$	$\{d_{j,12}, d_{j,13}\}$
R_6	$sig_{R_7}(h) = -1$	$\{p_0, \ldots, p_3\}$	$\{p_4, \ldots, p_7\}$
R_7	$sig_{R_8}(w_1) = -1$	$\{p_0, p_1\}$	$\{p_2, p_3\}$
R_8	$sig_{R_9}(h) =$ $-sig_{R_9}(w_2) = 1$	$\{p_4, p_5\}$	$\{p_6, p_7\}$

This completes the proof. □

Before we can put together the proof of Theorem 3 there is one last thing that we have to consider: Our idea is to create a union $U^A = (U_s^e \mid e \in E, s \in S, \neg s \xrightarrow{e} \})$ to finish the reduction for input A. However, in the given form, the unions U_s^e would not have mutually disjoint state and event sets. To resolve the name clash, we do the following renaming:

Lemma 11. *Let $U_s^e = U(M, D_0, \ldots, D_4, P, C)$ be a union of TSs as defined above. Then, we let $\tilde{U}_s^e = U(\tilde{M}, \tilde{D_0}, \ldots, \tilde{D_4}, \tilde{P}, \tilde{C})$ be the rectified union where, for all TSs $T = (S, E, s_0, \delta)$ in $\{M, D_0, \ldots, D_4, P, C\}$, we define $\tilde{T} = (\tilde{S}, \tilde{E}, \tilde{s}_0, \tilde{\delta})$ by $\tilde{S} = \{(e, s, x) \mid x \in S\}$ and $\tilde{E} = \{(e, s, x) \mid x \in E\}$ and $\tilde{s}_0 = (e, s, s_0)$ and*

$$\tilde{\delta}((e, s, x), (e, s, y)) = \begin{cases} (e, s, \delta(x, y)), & if\ \delta(x, y)\ is\ defined, \\ undefined, & otherwise \end{cases}$$

for all $(e, s, x) \in \tilde{S}$ and all $(e, s, y) \in \tilde{E}$. Then R is a region of U_s^e if and only if $\tilde{R} = \{(e, s, x) \mid x \in R\}$ is a region of the rectified union $\tilde{U}_s^e$.

The proof of this lemma is straightforward and therefore omitted. However, as the rectified unions have mutually disjoint state and event sets, the union U^A can now be defined as

$$U^A = (\tilde{U}_s^e \mid e \in E, s \in S, \neg s \xrightarrow{e}).$$

Proof of Thoerem 3. We start by showing that A has the ESSP if and only if U^A has the SSP. If U^A has the SSP then, for every $e \in E$ and every $s \in S$ with $\neg(s \xrightarrow{e})$, there is a region $\tilde{R}$ of U^A separating the key states $(e, s, m_0), (e, s, m_1)$ in $\tilde{U}_s^e$. This means, by definition and by Lemma 11, that

$$R_s^e = \{x \mid (e, s, x) \in \tilde{R} \cap S(\tilde{U}_s^e)\}$$

is a region of U_s^e separating m_0 and m_1. Lemma 8 provides that e is inhibtable at s. Hence, A has the ESSP.

Reversely, if A has the ESSP then, by Lemma 10, U_s^e has the SSP for all $e \in E$ and all $s \in S$ where $\neg s \xrightarrow{e}$. Using Lemma 11, we get for all $e \in E$ and all $s \in S$ where $\neg(s \xrightarrow{e})$ that $\tilde{U}_s^e$ has the SSP. As these union are pairwise disjoint with respect to states and events, we have that U^A has the SSP, too.

Given A, we can now create a linear 3-fold TS $A' = A(U^A)$ by joining the individual TSs of U^A. With the help of Lemma 1, we subsequently get that A' has the SSP if and only if U^A has the SSP if and only if A has the ESSP. As creating A' out of A is obviously doable in polynomial time, we get that linear 3-ESSP $\leq_p$ linear 3-SSP. In Sect. 4 we show that ESSP of linear 3-fold TSs is NP-complete and [3] states that SSP is in NP, which, together with our argumentation, imply that linear 3-SSP is NP-complete. □

6 Conclusion

This paper shows that synthesis of elementary net systems, one of the most frequently used Petri net classes, is hard, even if the permitted input is set close to triviality. Notice in this context that concatenating the individual segments of our unions in long linear transition systems is not our only option. In fact, we could assemble them in many possible ways and would still be able to derive our NP-completeness results. By that, the hardness of the three problems, SSP, ESSP, and feasibility, spreads to various other TS classes, as for instance nets with cycles. This means that there is not much hope for reasonable input subclasses allowing polynomial time ENS synthesis.

Consequently, encouraged by the class of Flip-Flop nets that provides a polynomial time algorithm for deciding the ESSP respectively feasibility [12], it is probably more reasonable for future work to focus research on synthesis complexity towards Petri net classes generalizing ENSs. There are a lot of cases that have not yet been studied well, as for instance Set nets [9], Trace nets [2], and variations of Flip-Flop nets [12] which are all contained in the class of *boolean nets* [4]. Therefore, we think it may be interesting to do further research on the complexity of SSP, ESSP and feasibility with respect to the subclasses of boolean nets, that, at best, clarifies exactly and extensively for which kinds of boolean nets the decision problems are tractable, respectively NP complete.

Moreover, we may need to look for algorithms that work well in practice, despite a non-polynomial worst-case complexity.

Acknowledgements. We thank the anonymous reviewers for their helpful suggestions. The first author thanks Eric Badouel for the inspiring correspondence about the synthesis of ENS.

References

1. Agostini, A., De Michelis, G.: Improving flexibility of workflow management systems. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 218–234. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_14
2. Badouel, E., Darondeau, P.: Trace nets and process automata. Acta Informatica **32**(7), 647–679 (1995)
3. Badouel, E., Bernardinello, L., Darondeau, P.: The synthesis problem for elementary net systems is NP-complete. Theor. Comput. Sci. **186**(1–2), 107–134 (1997)
4. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Texts in Theoretical Computer Science. An EATCS Series, pp. 1–325. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47967-4. ISBN 978-3-662-47966-7
5. Best, E., Erofeev, E., Schlachter, U., Wimmel, H.: Characterising Petri net solvable binary words. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016. LNCS, vol. 9698, pp. 39–58. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39086-4_4
6. Cortadella, J.: Private correspondence (2017)
7. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Complete state encoding based on the theory of regions. In: International Symposium on Advanced Research in Asynchronous Circuits and Systems, pp. 36–47. IEEE (1996)
8. Hiraishi, K.: Some complexity results on TS and elementary net systems. Theor. Comput. Sci. **135**(2), 361–376 (1994)
9. Kleijn, J., Koutny, M., Pietkiewicz-Koutny, M., Rozenberg, G.: Step semantics of Boolean nets. Acta Informatica **50**(1), 15–39 (2013)
10. Moore, C., Robson, J.M.: Hard tiling problems with simple tiles. Discret. Comput. Geom. **26**(4), 573–590 (2001)
11. Rosenke, C., Tredup, R.: The hardness of synthesizing elementary net systems from highly restricted inputs. arXiv:1711.00220 (2017)
12. Schmitt, V.: Flip-flop nets. In: Puech, C., Reischuk, R. (eds.) STACS 1996. LNCS, vol. 1046, pp. 515–528. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-60922-9_42
13. Yakovlev, A.V., Koelmans, A.M.: Petri nets and digital hardware design. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1492, pp. 154–236. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65307-4_49
14. OMG. Business Process Model and Notation (BPMN). Object Management Group (2016)
15. Scheer, A.-W.: Business Process Engineering, Reference Models for Industrial Enterprises. Springer, Heidelberg (1994). https://doi.org/10.1007/978-3-642-79142-0
16. UML. Unified Modeling Language (UML). Object Management Group (2016)

Petri Net Synthesis with Union/Find

Karsten Wolf(✉)

Institut für Informatik, Universität Rostock, Rostock, Germany
karsten.wolf@uni-rostock.de

Abstract. We propose a new algorithm for the synthesis of a Petri net from a transition system. It is presented for a class of place/transition Petri nets we call $\Delta 1$-Petri nets. A $\Delta 1$-Petri net has an incidence matrix where entries have values 0, 1, and -1 only. This class includes safe Petri nets as well as ordinary place/transition nets. The proposed algorithm can be adapted to these net classes. The algorithm employs Tarjan's union/find algorithm for managing sets of vertices. It requires just $O(|V||T|)$ space where V is the set of vertices and T is the set of transition labels. Consequently, problem instances even beyond 1,000,000 vertices have a manageable memory footprint. Our results are experimentally validated using a prototype implementation.

1 Introduction

Petri net synthesis [2] is the task of translating a labelled transition system (LTS) L with label set T into a Petri net N with transitions T such that the reachability graph of N is isomorphic to L. The problem can be reasonably generalised by permitting N to have a larger set of transitions T', with a mapping from T' to T [7]. In this paper we consider only the basic problem. Synthesis may target various classes of Petri nets, including elementary net systems [8,9], place/transition nets [1,11], or others. Several tools for synthesis already exist, for instance `petrify` [6], `genet` [5], `synet` [3], or `apt` [4].

Solutions to the synthesis problem are based on the theory of regions [2,8,9]. A region is the footprint, that a place of N leaves in the reachability graph of N. The precise notion of region depends on the class of Petri nets targeted for synthesis. Every region is connected to the set of transitions through its *signature*. The signature of region R specifies the effect that transitions have on the place represented by R. Synthesis tools assemble the resulting net from the given set T of transitions, a reasonably chosen set of regions serving as the set of places, and arcs that are deduced from the signature of the included regions.

An LTS usually has a large number of regions. Generally, many of them produce redundant places that may be left out of the resulting Petri net. In fact, it is sufficient to compute a set of regions that satisfies the *state separation* and *event/state separation* properties. State separation (SSP) means that each vertex corresponds to a distinct marking of the synthesised net. Event/state separation (ESSP) means that no transition fires in the synthesised net where that is not specified in the input LTS.

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 60–81, 2018.
https://doi.org/10.1007/978-3-319-91268-4_4

Our approach is based on regarding a region as a partition of the set V of vertices of L. We propose to calculate it from a trivial partition (all classes are singleton) by a sequence of union operations. These operations are managed by the famous and very efficient union/find algorithm proposed by Tarjan [14]. We call the intermediate partitions obtained during computation *proto-regions*. At the same time, we keep track of the partial information about the signature of the region-to-be and call that record of information a *proto-signature*. The data structures for proto-signatures and proto-regions are shallow enough to permit $O(|V||T|)$ space complexity. This is an important pre-requisite for being able to handle inputs with 1,000,000 vertices and beyond.

A pair of proto-region and proto-signature represents a set of regions that accord to the given record of information. Every proto-region imposes constraints on the corresponding proto-signature and vice versa. This way, we can refine both proto-region and proto-signature while preserving the set of according regions. We apply such refinements until no further conclusions can be made. Then, by considering all remaining cases on an underspecified element of the proto-signature, we divide the computation problem into several subproblems. Every according region accords to one of the subproblems. Hence, we ultimately traverse a search tree that branches at deliberate decisions made for the proto-signature. The leaves of the search tree represent the regions of the given LTS, or they represent inconsistent situations from which we backtrack. We further prune the search tree by backtracking from proto-regions and proto-signatures without contribution to SSP or ESSP. The result is not necessarily a minimal Petri net. We believe that further minimisation can then be organised as post-processing. We do not consider that task in this article. Our approach is quite different from the currently dominating approach where the synthesis problem is translated into a system of linear equations [1,4].

We demonstrate our approach for $\Delta 1$-Petri nets. A $\Delta 1$-Petri net is a place/transition net where the entries in the incidence matrix are limited to $\{-1, 0, 1\}$. This class includes safe Petri nets as well as ordinary place/transition nets with or without loops. Our approach can be adapted to target these classes.

We start with defining transition systems and Petri nets. Then we briefly recap the union/find approach. We continue with presenting our concepts of proto-region and proto-signature and reveal their mutual dependencies. After that, we sketch our calculation procedure. Instead of a running example. We describe a complete walk-through for a simple example. Finally, we present experimental results and conclude.

2 Transition Systems, Petri Nets, Synthesis

Definition 1. *A labelled transition system (LTS)* $L = [V, E, T, \lambda, v_0]$ *consists of a finite set* V *of vertices, a set* $E \subseteq V \times V$ *of edges, a finite set* T *of transition labels, a labelling function* $\lambda : E \to T$*, and an initial vertex* v_0*.*

For a transition label t, let $E_t = \{[v, v'] \mid \lambda([v, v']) = t\}$ be the set of t-labelled edges of L. For synthesis, it is convenient to consider only LTS that are

deterministic ($[v, v'] \in E, [v, v''] \in E$, and $v' \neq v''$ implies $\lambda([v, v']) \neq \lambda([v, v'])$) and *rooted* (every vertex is reachable via E from v_0).

Definition 2 (Petri Net). *A Petri net* $[P, T, F, W, m_0]$ *consists of finite and disjoint sets* P *and* T *of places resp. transitions, an arc relation* $F \subseteq (P \times T) \cup (P \times T)$*, a weight function* $W : F \to \mathbb{N} \backslash \{0\}$*, and the initial marking* m_0 *where a marking is a mapping* $m : P \to \mathbb{N} \cup \{0\}$*. Let* $W([x, y]) = 0$ *for* $[x, y] \notin F$*. Then transition* t *is enabled in marking* m *if, for all places* p*,* $m(p) \geq W([p, t])$*. Firing transition* t *in marking* m *yields marking* m' *(*$m \xrightarrow{t} m'$*) if* t *is enabled in* m *and, for all places* p*,* $m'(p) = m(p) - W([p, t]) + W([t, p])$*. The set of reachable markings* $R_N(m_0)$ *contains all markings that can be obtained from* m_0 *by any sequence of transition firings.* $R_N(m_0)$ *and the firing relation define the reachability graph of* N*, a labelled transition system with* T *as set of transition labels and* m_0 *as initial vertex.*

Definition 3 (Synthesis Problem). *The synthesis problem is formulated as follows: Given an LTS* L*, return a Petri net* N *such that the reachability graph of* N *is isomorphic to* L*, if such an* N *exists, and return* $\perp$*, otherwise.*

3 Union/Find

The union/find data structure has been designed for managing partitions of finite sets. A partition of M is a family of non-empty and pairwise disjoint sets that have a union equal to M. The union/find algorithm [14] considers elements and sets (classes). At each stage of computation, every set has a distinguished name that happens to be one of the contained elements. The algorithm supports `union` and `find` operations. `union(x,y)` replaces the sets with name `x` and `y` with their union. `find(x)` returns the unique name of the set that contains element `x`. The algorithm supports any sequence of union and find operations and represents, at each stage, the resulting partition. Its data structure consists of a single integer array of size $|M|$. The elements correspond to indices in the array. Each set is represented as a tree. The root of the tree is the name of the set. The array entry of a root is a negative number and represents the size of the net. For all other nodes, the array index is not negative and represents the index of its parent node in the array. Table 1 shows an example of a union/find array.

Table 1. Example of a union/find array representing the partition $\{\{0, 2, 4, 5\}, \{1\}, \{3, 6, 7\}\}$. The names of the classes are 1, 2, and 7.

2	−1	−4	7	0	2	7	−3

A find operation is realised by traversing the tree from the given element to the corresponding root. For efficiency, all visited nodes are then directly linked to their root (in our example, `find(4)` would replace the 0 by 2 in component 4).

This way, future find operations require less time. A union operation consists of linking the root of the smaller set to the root of the larger set. In our example, `union(1,7)` would write 7 into component 1 and -4 into component 7.

Initially, all sets are singleton. Performing a sequence of n union and find operations requires a worst case run time of $O(nlog^*n)$ where log^* is a function that grows so slowly that its impact can be neglected (call $O(nlog^*n)$ *quasi-linear*).

4 Backtracking in a Union/Find Structure

In the sequel, we shall derive a region by performing a sequence of union and find operations on the set of all vertices. If a region is found, or if a proto-region turns out to be inconsistent, we need to backtrack to an earlier stage of computation. Unfortunately, a union operation cannot be easily reversed. Moreover, storing explicit copies of intermediate partitions is not recommendable as each copy would consume space as large as $|V|$.

We solve the backtracking issue with the help of a stack that records all pairs of set names (i.e. integer numbers) for which actual union operations have been performed. Since there cannot be more than $n - 1$ subsequent union operations on n sets, this stack requires linear space. Every intermediate stage of computation corresponds to some index on that stack that separates earlier union operations from later union operations. Backtracking to such an intermediate stage consists of resetting the initial partition and then re-playing all union operations recorded up to the given index. This requires quasi-linear run-time which has basically the same order of magnitude as making and replacing explicit copies of the union/find data structure. Our space requirement is, however, only $3 \cdot |V|$ rather than $|V|^2$ required for a naive implementation.

This backtracking mechanism is important for meeting the $O(|V||T|)$ space goal of our approach.

5 Theory

A place/transition net N is a Δ1-Petri net if, for all $p \in P$ and $t \in T$, we have $|W([p,t]) - W([t,p])| \leq 1$. That is, arc weights may be arbitrarily large but the total effect of a transition to a place must not exceed consumption or production of a single token. This class includes the popular classes of ordinary (or plain) place/transition nets (all arc weights are 1) as well as safe Petri nets (at most one token on each place). It is included in the class of place/transition nets.

A region of a place/transition net is a mapping f from V to the natural numbers. The rationale behind this choice is to identify a place p_f in the reachability graph of N through the mapping f from m to $m(p)$, for all reachable markings m. That is, having a region f and a vertex v with $f(v) = k$, p_f would have k tokens in the marking $\phi(v)$ where ϕ is the isomorphism between L and N to be established.

We compute f in two steps. First, we calculate the partition R of V induced by f as follows: Let $R = \{V_i | i \in \mathbb{N}\}$ where $V_i = \{v | f(v) = i\}$. If f is indeed a region of a $\Delta 1$-Petri net, edges of L turn R into some double linked list where an edge in E_t with $W(t, p_f) - W(p_f, t) = 1$ establishes a forward link, an edge in E_t with $W(t, p_f) - W(p_f, t) = -1$ represents a back link, and an edge in E_t with $W(t, p_f) - W(p_f, t) = 0$ is a self loop. For classes linked this way (disregarding self-loops), the value of f differs by exactly 1, and is the same for all reachable markings. This property can be used to deduce the actual values of f.

Since the effect of a transition to a place is the same in all markings, a transition can be *ascending* (all t-edges are forward links), it can be *descending* (all t-edge are back links), or *ignoring* (all t-edges are self-loops). If none of the attributes can be assigned to t, R cannot be related to a place in a $\Delta 1$-Petri net, so it is not a region. We formalise these ideas as follows.

Definition 4 (Signature). *A signature of a set T of transition labels is a mapping $s : T \to \{-1, 0, 1\}$ where a transition t with $s(t) = 1$ is called ascending, a transition with $s(t) = -1$ is called descending, and a transition with $s(t) = 0$ is called ignoring.*

Definition 5 (Region). *Let $L = [V, E, T, \lambda, v_0]$ be an LTS. A region with signature s is a partition $\{V_0, \ldots, V_k\}$ such that, for every $[x, y] \in E$ with $x \in V_i$, $y \in V_{i+s(\lambda([x,y])}$.*

In this definition, we slightly abuse notation. We write $\{V_0, \ldots, V_k\}$ as a family of sets but actually use the indices to impose a particular order on these sets. This way, we avoid over-formalisation. We shall take care of the introduced ambiguity at due time.

Let, for a transition t, i_t be the smallest index such that V_{i_t} contains source nodes of edges labelled with t. Given a region $\{V_0, \ldots, V_k\}$, a corresponding place can be derived as follows.

- the capacity (i.e. highest number of tokens on p) is k,
- its initial marking is j if V_j contains v_0,
- for all t, $W([p, t]) = i_t$,
- for all t, $W([t, p]) = i_t + s(t)$.

This way, the region net for a given set $\mathcal{F}$ of regions is fully specified. Observe that $W([t, p])$ cannot become negative for descending transitions since no transition can descend from V_0.

According to [2], the region net for $\mathcal{F}$ has a reachability graph that is isomorphic to the given LTS L if and only if $\mathcal{F}$ meets all separation problems of L. We consider state separation problems (SSP) and event/state separation problems (ESSP). SSP makes sure that every pair of vertices has a different "code" in the synthesised net. This means that, for vertices v, v' ($v \neq v'$) of L, markings $\phi(v)$ and $\phi(v')$ differ in the token count of a least one place. ESSP makes sure that a transition cannot fire in the region net if that is not prescribed in L. The separation problems can actually be formulated in terms of the partitions induced by a set of regions.

Definition 6 (SSP, ESSP for Δ1-Petri nets). *Let $L = [V, E, T, \lambda, v_0]$ be an LTS. Let $\mathcal{F}$ a set of regions of L and $\mathcal{R}$ be the set of partitions induced by the regions in $\mathcal{F}$. Two vertices v, v' are separated w.r.t. $\mathcal{F}$ if there exists $R = \{V_0, \ldots, V_k\} \in \mathcal{R}$ and two indices $i \neq j$ with $v \in V_i$, $v' \in V_j$. Transition t is separated from vertex v w.r.t. $\mathcal{F}$ if there exists a vertex v' with $[v, v'] \in E_t$, or there exists $R = \{V_0, \ldots, V_k\} \in \mathcal{R}$ where $v \in V_i$ and $j \leq i$ implies that there are no source nodes of t-labelled edges contained in V_j. $\mathcal{F}$ satisfies SSP if all pairs of vertices ($v \neq v'$) are separated, $\mathcal{F}$ satisfies ESSP if all pairs of transitions and vertices are separated.*

Indeed, if a region R separates two vertices v and v', the place corresponding to R has a different number of tokens for v (i tokens) and v' (j tokens). Likewise, if a vertex v and a transition t are separated by region R, t is enabled in v, or the place built from R disables t in the marking that models v.

We obtain

Proposition 1 ([2])

- *A set $\mathcal{F}$ of regions of L defines a Petri net that has a reachability graph isomorphic to L if and only if $\mathcal{F}$ satisfies SSP and ESSP.*
- *For L there is a Δ1-Petri net with isomorphic reachability graph if and only if there is a set of regions where the corresponding region net has a reachability graph isomorphic to L.*

Consequently, our task amounts to finding sufficiently many regions to meet all SSP and ESSP instances.

The following notion of *generalised path* will be useful for identifying regions in L. It helps us to lift the concept of edges from the level of vertices to the level of classes in the partition. At the same time, it implements the idea that ascending and descending edges have opposite effects.

Definition 7 (Generalised path). *Let $L = [V, E, T, \lambda, v_0]$ be an LTS and $R = \{V_0, \ldots, V_k\}$ be a partition of V. A sequence $\pi = t_1 \ldots t_n$ of transitions in T is a generalised path in R if there exist vertices $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ such that the following conditions hold:*

1. *for all i, $[x_i, y_i] \in E_{t_i}$ or $[y_i, x_i] \in E_{t_i}$.*
2. *for all $i < n$, there exists a $V \in R$ with $x_{i+1} \in V$ and $y_i \in V$.*

We say that π starts in V_i if $x_1 \in V_i$. π ends in V_j if $y_n \in V_j$.

That is, we may travel through edges in arbitrary direction, and we may jump between vertices if they belong to the same class in R. Nevertheless, every actual path in L is a generalised path as well, regardless of R (just let all $[x_i, y_i] \in E_{t_i}$ and $x_{i+1} = y_i$). We extend the concept of signature to transition sequences such that the signature of the involved transitions and the travel direction for edges is taken into account. The result is a mapping from sequences of transition into the set of integer numbers.

Definition 8 (Generalised signature). *Let $L = [V, E, T, \lambda, v_0]$ be an LTS and $R = \{V_0, \ldots, V_k\}$ be a partition induced by a region of V. Let $\pi = t_1 \ldots t_n$ be a generalised path for R and s a signature. Let all x_i and y_i be as in Definition 7. Then the generalised signature s^* is inductively defined by the following rules:*

- *for the empty sequence ε, let $s^*(\varepsilon) = 0$.*
- *if $[x_i, y_i] \in E_{t_i}$, let $s^*(\pi t_i) = s^*(\pi) + s(t_i)$.*
- *if $[y_i, x_i] \in E_{t_i}$, let $s^*(\pi t_i) = s^*(\pi) - s(t_i)$.*

Since we expect R to be induced by a region with signature s, s^* is actually well-defined independently of the assigned x_i and y_i. From the Definitions 5 and 8, we can immediately deduce the fundamental property of generalised paths in Δ1-Petri nets:

Lemma 1. *Let $L = [V, E, T, \lambda, v_0]$ be an LTS and $R = \{V_0, \ldots, V_k\}$ be a partition induced by a region of V with signature s. Let $t_1 \ldots t_n$ be a generalised path in R that starts in V_i and ends in V_j. Then $j = i + s^*(\pi)$.*

6 Proto-Regions of Δ1-Petri Nets

A proto-region represents an intermediate step in the computation of a region. It is in fact a partition of V. It represents the fact that unified nodes shall ultimately belong to the same class of vertices in a region f to be computed. In other words, the isomorphism ϕ to be established will map vertices v, v' in the same class to markings $\phi(v), \phi(v')$ such that $\phi(v)(p_f) = \phi(v')(p_f)$.

Definition 9 (Proto-Region for Δ1-Petri net). *A proto-region is a partition of the set of vertices of a LTS. A region f inducing partition $R = \{V_0, \ldots, V_k\}$* accords *to a proto-region $S = \{V'_0, \ldots, V'_m\}$ ($f \models S$), if, for all i ($0 \leq i \leq m$), there is a j ($0 \leq j \leq k$}) such that $V'_i \subseteq V_j$.*

Obviously, for every region f, $f \models \{\{v\} | v \in V\}$. It is easy to see:

Lemma 2. *Let S be a proto-region. Let f be a region with $f \models S$, and R be the partition induced by f. Every generalised path in S is a generalised path in R.*

7 Proto-Signatures of Δ1-Petri Nets

Consider a region with induced partition R. The edges of the given LTS define the neighbourhood relation between its classes. However, the very same partition can then still be interpreted as $\{V_0, \ldots, V_k\}$ or as $\{V_k, \ldots, V_0\}$. All other orderings can be ruled out by the fact that, in a Δ1-Petri net, edges form the already mentioned "double linked list" structure. The actual ordering of the classes is important for defining ascending versus descending transitions as well as the initial marking of the place corresponding to R. In fact, for every place p, its complement place induces the same partition. Only the ordering of the

classes is reversed. That is, computing a partition that satisfies all properties of a region, we end up in a structure that describes both a place and its complement place. This is not a problem, though. Whether or not we finally ship a place, its complement, or both, to the resulting Petri net, ultimately depends on the contribution of the two places to the separation problems. For ESSP, the contribution may actually differ. We shall pick up this thought in Sect. 12.

To address the duality between a place and its complement place, we do not treat transitions as ascending or descending in the concepts below. Instead, we record the *relative* orientation for *pairs* of transitions They may have the *same* orientation (both ascending, both descending) or *opposite* orientation (one ascending, the other one descending). Additionally, relative orientation may include, exclude, or be exclusively set to *ignoring*. We shall use the following symbols: SA for "same", SI for "same or ignoring", OP for "opposite", OI for "opposite or ignoring", and II for "ignoring". The ideas lead to the following formalisation:

Definition 10 (Proto-Signature of Δ1-Petri net). *A proto-signature is a subset of* $T \times \{SA, SI, OP, OI, II\} \times T$. *Signature* s accords *with proto-signature* σ *(*$s \models \sigma$*) if the following conditions are satisfied:*

- $[t, SA, t'] \in \sigma$ *implies* $s(t) = s(t') \neq 0$;
- $[t, SI, t'] \in \sigma$ *implies* $s(t) = s(t')$;
- $[t, OP, t'] \in \sigma$ *implies* $s(t) = -s(t') \neq 0$;
- $[t, OI, t'] \in \sigma$ *implies* $s(t) = -s(t')$;
- $[t, II, t'] \in \sigma$ *implies* $s(t) = s(t') = 0$.

We formalised a proto-signature as a set of triples. This way, new facts can be easily added. However, we can translate a proto-signature into a LTS with T as set of vertices and $\{SA, SI, OP, OI, II\}$ as set of labels. The reason is that, whenever there exist two triples $[t, X, t'] \in \sigma$ and $[t, Y, t'] \in \sigma$ with $X \neq Y$, σ is either inconsistent (there is no s with $s \models \sigma$), or the two triples can be replaced with a single triple, keeping the set of according signatures invariant. Actually, if $X, Y \in \{SA, OP, II\}$, σ is inconsistent. If $X, Y \in \{SI, OI, II\}$, the two triples can be replaced with $[t, II, t']$. If $X, Y \in \{SA, SI\}$, only $[t, SA, t']$ needs to be kept, and for $X, Y \in \{OP, OI\}$, only $[t, OP, t']$ is significant. They remaining combinations $X, Y \in \{SA, OI\}$ and $X, Y \in \{SI, OP\}$ are again inconsistent. Hence, for each pair of transitions, there is a unique "best" label. Only this label needs to be kept.

Moreover, we may transitively propagate from $[t, X, t']$ and $[t', Y, t'']$ to some value Z such that every signature that accords with σ also accords with $\sigma \cup \{[t, Z, t'']\}$. The appropriate value of Z depending on X and Y is given in Table 2.

In the sequel, we assume that, whenever we modify σ, all deductions described so far are applied. We further assume that, if a modification leads to an inconsistent proto-signature, this is detected and we backtrack to some other subproblem. So, we silently assume σ to be consistent.

Table 2. Deduced value of Z in $[t, Z, t'']$ depending on X and Y in $[t, X, t'], [t', Y, t''] \in \sigma$. CC denotes "inconsistent".

	SA	SI	OP	OI	II
SA	SA	SA	OP	OP	CC
SI	SA	SI	OP	OI	II
OP	OP	OP	SA	SA	CC
OI	OP	OI	SA	SI	II
II	CC	II	CC	II	II

Otherwise, the labelled graph given by a proto-signature enjoys some interesting properties. It partitions T into connected components. In every component, the transitivity rules in Table 2 establish that every component actually is a clique. Applying all mentioned rules until nothing changes, we actually arrive at three types of cliques:

- Type 1: all labels are II;
- Type 2: all labels are SI or OI;
- Type 3: all labels are SA or OP.

In the sequel, let $[t]_\sigma$ be the clique containing t, i.e., $[t]_\sigma = \{t' \mid \exists X : [t, X, t'] \in \sigma\}$. Assume further that every such clique has a canonical representative $canrep([t]_\sigma)$.

In adding more and more knowledge to a proto-signature, we approach a situation where the set of transitions falls into at most two cliques. The first clique has type 2 or 3 while the optionally present second clique has type 1. Two signatures accord with such a proto-signature. Setting the signature for one of the transitions in the first clique to 1, the signature of all other transitions in the clique can be deduced to be either 1 or -1. Setting the signature to -1 instead, the signature of all other transitions can be deduced accordingly. In both cases, the second clique contains the transitions with signature value 0. The two signatures then correspond to a place and its complement place. If the first clique has type 2, there is a third according signature. It assigns 0 to all transitions. We can, however, ignore this case as the resulting place would never change its marking and is thus redundant anyway.

8 Simple Deductions Between Proto-Region and Proto-Signature

The following rules in a proto-region $S = \{V_1, \ldots, V_k\}$ suggest values for a proto-signature σ, based on simple patterns in a proto-region R. We use the notation $\sigma := \sigma'$ for: if a region accords with R and its signature s accords with σ, then s accords with σ', too.

RS1 If $[x, y] \in E_t$ and $x, y \in V_i$ then $\sigma := \sigma \cup \{[t, II, t]\}$:
RS2 If $[x, y] \in E_t$, $[x', y'] \in E_{t'}$, $x, x' \in V_i$, and $y, y' \in V_j$, then $\sigma := \sigma \cup \{[t, SI, t']\}$:
RS3 If $[x, y] \in E_t$, $[x', y'] \in E_{t'}$, $x, y' \in V_i$, and $y, x' \in V_j$, then $\sigma := \sigma \cup \{[t, OI, t']\}$:

Lemma 3. *Rules RS1, RS2, and RS3 preserve the set of according regions.*

Proof. Consider generalised paths t and t' (if applicable) in any region f that accords to σ and S. Let R be the partition induced by f. We apply Lemma 1. In RS1, t starts and ends in the same class of R, so $s(t) = 0$. In RS2, t starts in the same class as t', and t ends in the same class as t'. Hence, $s(t) = s(t')$. In RS3, The generalised path tt' starts and ends in the same class. Hence, $s^*(tt') = 0$, so $s(t) = -s(t)$. □

The other way round, a proto-signature σ has an impact on a proto-region S. In the subsequent set of rules, we derive union operations for S such that every region that accords with the original S and has a signature according with σ, accords with the new proto-region, too. $union(v, v')$ means: unify the set containing v with the set containing v'.

SR1 If $[t, II, t] \in \sigma$ then, for all $[v, v'] \in E_t$, $union(v, v')$.
SR2 If $\{[t, SA, t'], [t, SI, t']\} \cap \sigma \neq \emptyset$, $[v_1, v_2 \in E_t$, $[v_3, v_4] \in E_{t'}$, and there is a $V \in R$ with $v_1, v_3 \in V$, then $union(v_2, v_4)$.
SR3 If $\{[t, SA, t'], [t, SI, t']\} \cap \sigma \neq \emptyset$, $[v_1, v_2] \in E_t$, $[v_3, v_4] \in E_{t'}$, and there is a $V \in R$ with $v_2, v_4 \in V$, then $union(v_1, v_3)$.
SR4 If $\{[t, OP, t'], [t, OI, t']\} \cap \sigma \neq \emptyset$, $[v_1, v_2] \in E_t$, $[v_3, v_4] \in E_{t'}$, and there is a $V \in R$ with $v_2, v_3 \in V$, then $union(v_1, v_4)$.

Lemma 4. *Rules SR1, SR2, SR3, and SR4 preserve the set of according regions.*

Proof. Consider generalised paths t and t' (if applicable) in any region f that accords to σ and S. Let R be the partition induced by f. We apply Lemma 1. In SR1, $s^*(t) = 0$, so any start and end of t must actually be contained in the same class of R. In SR2 and SR3, we have $s^*(t) = s^*(t')$. In SR2, t and t' start in the same class of R, so they must end in the same class of R, too. In SR3, t and t' end in the same class of R, so they must start in the same class of R, too. In SR4, we have tt' is a generalised path with $s^*(tt') = 0$. So, its start and end must be contained in the same class of R. □

9 Topological Deductions

For more sophisticated deductions, we consider an LTS that we derive from the given LTS L, a proto-region S, and a proto-signature σ. Its vertices are the classes of S and its labels are the canonical representatives of the cliques induced by σ. The new structure reflects the fact that transitions identified as "same" actually have the same effect. Accordingly, transitions identified as "opposite" have reverse effect. Hence, an edge labelled with t^* is drawn from V to V' iff there is an edge $[v, v']$ in L labelled with t where one of the following two conditions holds:

(1) $v \in V$, $v' \in V'$ and $\{[t, SA, t^*], [t, SI, t^*]\} \cap \sigma \neq \emptyset$, or
(2) $v \in V'$, $v' \in V$ and $\{[t, OP, t^*], [t, OI, t^*]\} \cap \sigma \neq \emptyset$.

Call the resulting LTS the *proto-LTS* for S and σ. It is an intermediate result in folding L into a region.

Lemma 5. *Let L be an LTS, S a proto-region, σ a consistent proto-signature, and L^* the corresponding proto-LTS. For every path $t_1 \dots t_n$ in L^* there exists a generalised path $t'_1 \dots t'_n$ in S such that, for all i, $t_i = canrep(t'_i)$. For all such $t'_1 \dots t'_n$, $s^*(t'_1 \dots t'_n) = \sum_{i=1}^{n} s(t_i)$.*

Proof. Path $t'_1 \dots t'_n$ can be assembled according to the defining conditions of the proto-LTS. For all i, either condition (1) or condition (2) is applicable. In case (1), we have $s(t_i) = s(t'_i)$ and the directions of edges $[v, v']$ and $[V, V']$ coincide. In case (2), we have $s(t_i) = -s(t'_i)$ but $[v, v']$ and $[V, V']$ have reverse directions. In both cases, Definition 8 yields the same value for t_i and t'_i to be added to s^*. □

For a transition sequence π, consider the Parikh vector (transition count vector) $\Psi(\pi)$. $\Psi(\pi)(t)$ is the number of occurrences of t in π. Using Ψ, we can rewrite the equation in Lemma 5 to

$$s^*(\pi') = \sum_{t \in T} \Psi(\pi)(t) s(t).$$

Our first application of the lemma concerns the refinement of the proto-region. Consider a proto-region S, a proto-signature σ, and the induced proto-LTS.

TR Let π_1 and π_2 be paths in the proto-LTS with $\Psi(\pi_1) = \Psi(\pi_2)$. Let V, V' and V'' be classes in S such that one of the following two conditions holds:
(1) π_1 starts in V and ends in V', π_2 starts in V and ends in V'', or
(2) π_1 starts in V' and ends in V, π_2 starts in V'' and ends in V. Then: $union(V', V'')$.

Lemma 6. *Rule TR does not change the set of according regions.*

Proof. Let f be a region according to S and σ and R be the partition induced by f. Since f accords to S, all vertices in V will be contained in the same class of R. By $\Psi(\pi_1) = \Psi(\pi_2)$ and Lemma 5, all nodes in $V' \cup V''$ are contained in the same class of R. □

The second application of Lemma 5 permits the refinement of the proto-signature.

TS Let π_1 and π_2 be paths of the proto-LTS that both start in some vertex V and both end in some vertex V'. Let, for all t, $k(t) = \Psi(\pi_1)(t) - \Psi(\pi_2)(t)$.
(1) if there is a transition label t^* such that $|k(t^*)| > \sum_{t \neq t^*} |k(t)|$ then $\sigma := \sigma \cup \{[t^*, II, t^*]\}$;
(2) if there is a transition label t^* such that $|k(t^*)| = \sum_{t \neq t^*} |k(t)|$ then $\sigma := \sigma \cup \{[t^*, II, t^*]\}$ or, for all $t \neq t^*$:

- if $sign(k(t)) = sign(k(t^*))$ then $\sigma := \sigma \cup \{[t, OI, t^*]\}$;
- if $sign(k(t)) = -sign(k(t^*))$ then $\sigma := \sigma \cup \{[t, SI, t^*]\}$.

Lemma 7. *Rule TS does not change the set of according regions.*

Proof. Since, for all t, $s(t) \in \{-1, 0, 1\}$, the proposed values are the only possibilities to solve the equation of Lemma 5. □

10 Branching the Search Space

If none of the rules is applicable, computation continues with a separation of the search space into disjoint parts. This way, the search tree branches. Since in general, the number of transitions is much smaller than the number of vertices, we branch w.r.t. the proto-signature σ. To this end, we consider two transitions t and t' for which no mutual relation is recorded in σ (i.e. they belong to different cliques). We consider the following four cases:

- t is ignoring;
- t is not ignoring and t' is ignoring;
- none of t and t' is ignoring and they have the same signature;
- none of t and t' is ignoring and they have opposite signatures.

These cases can be implemented by adding one of the following tuples to σ: $[t, II, t]$, $[t', II, t']$, $[t, SA, t']$, $[t, OP, t']$. In the second case, we additionally have to take care that, subsequently, t is not set to ignoring. We implement this issue with the help of an additional bit per transition. Setting this bit, any attempt to add $[t, II, t]$ will be recognised as inconsistency. Any subsequent addition of $[t, SI, t']$ or $[t, OI, t']$ will be modified to $[t, SA, t']$ or $[t, OP, t']$, respectively. This way, our cases cover all situations (which is important for correctness) and are disjoint (which is important for efficiency). Consequently, every decision point in the search tree has four branches. Although the number of transition pairs is quadratic in the number of transition labels, the depth of the search tree cannot grow beyond the number of transitions. This is due to the transitive conclusions mentioned above. This means that every case unifies two transition cliques induced by σ and the number of union operations is bounded by the number of transitions.

For the selection of a transition pair, we implemented a greedy strategy. We select the transition with the largest number of classes where both transitions have source or sink vertices. This way, it is likely that the simple rules quickly provoke a large number of union operations. In effect, the proto-LTS becomes smaller and operations on it need less time.

11 Detecting a Region

The search process approaches a situation where only two regions accord to proto-region and proto-signature: one for a place p and one for its complement

place. This situation is reached if σ consists of at most two cliques: one of type 2 or 3, and optionally one of type 1. We further assume that all simple rules are exhaustively applied and did not cause any inconsistency. Finally we assume that at least the following spacial case of rule TS is not applicable: Let $\pi_1 = t^k$, for arbitrary k, and $\pi_2 = \varepsilon$. In such a case (for $k > 0$), t^k is a uniformly labelled cycle in the proto-LTS and rule TS would set t to ignoring. Consequently, by rule SR1, all vertices on the cycle would collapse into a single class.

Under the described conditions, there is only one label that connects different vertices in the proto-LTS: it is the component of type 2 or 3. There cannot be forward nor backward branches in the proto-LTS since otherwise one of the rules SR2, SR3, or SR4 would be applicable. There cannot be a cycle as otherwise rule TS would be applicable. Finally, the proto-LTS is connected since the original LTS is connected and the process of folding the original LTS into a proto-LTS does not interrupt existing connections. Consequently, the proto-LTS has the shape of a linked list. This list contains a unique vertex without predecessor. This serves as V_0. Then, let the unique successor of V_i be V_{i+1} and we have an ordering of the classes of proto-region as required for a region. Let t^* be the canonical representative of the non-ignoring clique of σ. All transitions t with $[t, SA, t^*]$ or $[t, SI, t1*]$ naturally become ascending transitions and all transitions t with $[t, OP, t^*]$ or $[t, OI, t1*]$ become descending. Transitions in the other clique, if present, are ignoring. The complement place is generated by reversing the order of the classes in the proto-region, and by multiplying all signature values with -1.

Should we arrive at a situation with only one clique of type 1, we can immediately backtrack. A resulting place would never change its marking, so it cannot contribute to SSP nor ESSP.

12 Managing the Separation Problems

Throughout the whole calculation process, we record information about solved instances of the two separation problems SSP and ESSP. Whenever a region is detected, we update that information. At any time during the search, we analyse whether, for the given proto-region and proto-signature, contributions to the remaining instances may be expected from according regions. If not, we backtrack and continue our search in another part of the search space. As soon as all separation problems are completely solved, we exit.

For recording information on solved and unsolved SSP instances, we use the linear data structure proposed in [12]. The data structure represents a partition of the set of vertices. Its semantics is as follows: v and v' are separated w.r.t. the already known regions if and only if v and v' appear in different classes of the partition. Unfortunately, this is not another application of union/find since the dominating operation is intersection, not union. We implement the partition using three arrays of size $|V|$. In the first array, the indices of all vertices of some class S_i are recorded as a continuous section. In the second and third array, the i-th entry represents the first and last index containing elements of set S_i.

If a new region is detected, we use the induced partition R for updating the SSP information. We sort every part of the first array according to set names of its vertices in R. Then, for each distinct value, a new set is introduced by recording start and end in the second and third arrays. Singleton sets are removed as they do not represent unsolved SSP instances. As soon as the last set is separated into singletons, SSP is satisfied for the regions detected so far.

A proto-region S does not contribute to SSP as soon as no according region solves an instance of SSP that is still open.

Lemma 8. *A proto-region S does not contribute to SSP if every set of the SSP data structure is included in a class of the partition S.*

Proof. A region can separate only states that are in different classes of its induced partition R. Since only regions accord to S where every class of S is included in some class of R, no contribution to open instances of SSP is possible. □

For ESSP, we keep a two-dimensional bit-array of size $|V||T|$. v and t are separated w.r.t. existing regions if and only if the bit at index v and t is set to true. Upon detection of a region, we determine, for every transition t, the smallest class of the induced partition R that contains sources of t-edges. Vertices in smaller classes of R are those where this region contributes to ESSP, and we update the corresponding bits. If any new bit is changed, the region contributes to ESSP. The same procedure is independently executed for the reverse region (i.e. the complement place). If a region contributes to SSP or ESSP, the corresponding place is added to the resulting region net. The complement place is added if it has its own contribution to ESSP (for SSP, the contribution of place and complement place are identical).

The remaining problem in this section is to find out whether a given proto-region S and proto-signature σ can contribute to open ESSP problems. To this end, let us fix a transition t^* and a vertex v^* to be separated. Consider the proto-LTS L^* for S and σ. Let V^* be the unique class of S that contains v^* and let t be an arbitrary transition appearing in L^*. We assume that the check for ESSP contribution is executed in a situation where no simple rules are applicable (so there is no forward nor backward branching in L^* with edges that have the same label), and the topological rules have been applied at least to the degree that there are no uniformly labelled cycles in L^*. Then there is a unique and finite longest (but possibly empty) path using t-edges starting from V^*. It ends in some node V_y of L^*. Analogously, there is a longest and possibly empty path from V^* using reverse t-edges. It ends in some V_x. Together, we have a path (that still can be empty) from V_x via V^* to V_y using only t-edges. We claim that the current situation comprised of S and σ cannot contribute to the ESSP instance $[v^*, t^*]$ if some vertex V on the path between V_x and V^* (including both) contains a source of a t^*-edge, and some vertex V' on the path between V^* and V_y (including both contains a source of a t^*-edge.

To verify this claim, assume a situation as described and consider any according region f and its induced partition R. Depending on the signature of t, there are two cases to be considered. In the first case, t becomes ignoring in f. Then,

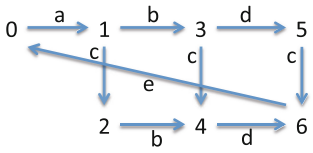


Fig. 1. Example of an LTS.

all classes on the considered path are contained in the same class of R. This contains a source of a t^*-edge, so f cannot contribute to this ESSP instance. In the second case, t is not ignoring. Then the classes on the considered path are contained in different but connected classes of R. Since the least class of R containing sources of t^* determines the weight of the input arc to t^*, one of the existing sources proves that the resulting place cannot disable t^* while the existing source in the other sub-path proves that the complement pace cannot disable t^*. Consequently, no according region can contribute to this ESSP instance.

We check the ESSP contribution by traversing, for all t, the transition systems $[V^*, E_t^*]$, taking $O(|V||E_t|)$ time. This sums up to a total of $O(|V^*||E^*|)$. Taking into consideration that most checks are performed on deep levels of the search tree, i.e. on already small proto-LTS, the run-time is acceptable and is in any case less expensive than traversing the useless branches of the search tree.

If our algorithm is applied to a subclass of Δ1-Petri nets, the management of SPP and ESSP needs to be adapted.

13 A Walkthrough

Consider the transition system in Fig. 1. We start in the following situation. In the presentation of σ, $\{a, b/d, e\} : k$ stands for: k is the type of the component. If both transitions appear before or both appear after the slash, the have orientation SA or SI. Otherwise their orientation is OP or OI.

$$S = \{\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}\}\ \sigma = \{\{a\} : 2, \{b\} : 2, \{c\} : 2, \{d\} : 2, \{e\} : 2\} \tag{1}$$

The SSP data structure is $\{0, 1, 2, 3, 4, 5, 6\}$. In the ESSP structure, all bits $[v, t]$ where v is source of a t-transition, are set as "solved". In this situation, no rules are applicable, so we set a first decision point. We select a pair of transitions, b and c, and start with the first case: b is ignoring. By rule SR1, this leads to the following situation:

$$S = \{\{0\}, \{1, 3\}, \{2, 4\}, \{5\}, \{6\}\}\quad \sigma = \{\{a\} : 2, \{b\} : 1, \{c\} : 2, \{d\} : 2, \{e\} : 2\} \tag{2}$$

For the second decision point, we select transition pair d and c and make d ignoring. We obtain

$$S = \{\{0\}, \{1, 3, 5\}, \{2, 4, 6\}\}\quad \sigma = \{\{a\} : 2, \{b, d\} : 1, \{c\} : 2, \{e\} : 2\} \tag{3}$$

Rules are not applicable. A third decision point uses c and a. Making c ignoring, leads us to proto-region $\{\{0\},\{1,2,3,4,5,6\}\}$. Rule RS3 is applicable for transitions a and e, so we obtain

$$S = \{\{0\},\{1,2,3,4,5,6\}\} \quad \sigma = \{\{a/e\} : 2, \{b,c,d\} : 1\} \tag{4}$$

In this situation, we have found two regions f_1 and f_2. We have $f_1(0) = 0$ and $f_1(x) = 1$, for $x \neq 0$. a is ascending, e is descending. The corresponding place has $\{a,b,c,d\}$ as pre-set and $\{b,c,d,e\}$ as post-set. All multiplicities are 1. f_2 with $f_2(x) = 1 - f_1(x)$ is the complement place. It has pre-set $\{e\}$ and post-set $\{a\}$. Both f_1 and f_2 contribute to SSP and ESSP, so they are added to the resulting region net. The new SSP record is $\{\{1,2,3,4,5,6\}\}$. In ESSP, bits a1, a2, a3, a4, a5, a6 are set due to f_2, and bits b0, c0, d0, e0 are set due to f_1. Hence, ESSP is completed for transition a. We backtrack to situation (3) and try the second option. That is, a becomes ignoring while ignoring is excluded for c. With rule RS1 applied to a, we obtain

$$S = \{\{0,1,3,5\},\{2,4,6\}\} \quad \sigma = \{\{c\} : 3, \{a,b,d\} : 1, \{e\} : 2\} \tag{5}$$

With rule RS3, a and e become opposite, so we get another pair of regions. The first place has pre-set $\{c\}$ and post-set $\{e\}$ while the complement place has pre-set $\{a,e\}$ and post-set $\{a,c\}$. SSP reduces to $\{\{1,3,5\},\{2,4,6\}\}$. In ESSP, bits e1, e3, e5, c2, c4, c6 are set. So ESSP is completed for transition c. We backtrack once more to situation (3) and set a and c to SA. Considering the proto-region in situation 3, there is a path from $\{0\}$ with a to $\{1,3,5\}$, with c to $\{2,4,6\}$, and with e back to $\{0\}$. a and c are in the same clique now. Let c be its canonical representative. Apply rule TS with $\pi_1 = cce$ and $\pi_2 = \varepsilon$. The rule tries to set c to ignoring, in contradiction to the setting that c should not be ignoring. So we backtrack. The final option in situation (3) is to set a and c to OP. We result in

$$S = \{\{0\},\{1,3,5\},\{2,4,6\}\} \quad \sigma = \{\{c/a\} : 3, \{b,d\} : 1, \{e\} : 2\} \tag{6}$$

Here, S is not able to split any class in the remaining SSP record. Since b and d both have outgoing edges from $\{1,3,5\}$ and $\{2,4,6\}$ and e has an outgoing edge from $\{2,4,6\}$, we may conclude that the current situation has only according regions without contribution to the separation problems. We backtrack and return to the second decision point since the third decision point has been exhaustively explored. So we return to situation (2), set transition c to ignoring and mark d as "not ignoring". We arrive at situation

$$S = \{\{0\},\{1,2,3,4\},\{5,6\}\} \quad \sigma = \{\{a\} : 2, \{b,c\} : 1, \{d\} : 3, \{e\} : 2\} \tag{7}$$

Since no rule is applicable, we open a fourth decision point and choose e and a as transition pair. In the first branch, e is set to ignoring. So $\{0\}$ and $\{5,6\}$ are unified and with rule RS3, a and d become OP. In this situation,

$$S = \{\{1,2,3,4\},\{0,5,6\}\} \quad \sigma = \{\{a/d\} : 3, \{b,c,e\} : 1\} \tag{8}$$

We produce the next two places. One has pre-set $\{a, b\}$ and post-set $\{b, d\}$, the other one has pre-set $\{d, e\}$ and post-set $\{a, e\}$. The SSP record reduces to $\{\{2, 4\}, \{1, 3\}\}$ and in the ESSP record, only bits b3, b4, d1, d2 remain unset. Returning to situation (7), there cannot be any further contribution to SSP since class $\{1, 2, 3, 4\}$ in incapable to split the remaining SSP sets. In addition, $\{1, 2, 3, 4\}$ contains sources of b and d, so no contribution to ESSP is possible. We backtrack to the second decision point (situation 2). Again, we can see that there is no more contribution to SSP since all sets of the remaining SSP records are included in some set of S. The classes containing vertices 1, 2, 3, 4 all have sources of b-edges and d-edges, so no contribution to ESSP is possible either. We backtrack to the first decision point. Having already demonstrated all major ingredients of our approach, we skip the details of the remaining approach. It leads to a final pair of places where one has pre-set $\{b\}$ and post-set $\{d\}$ while the other one has pre-set $\{a, d, e\}$ and post-set $\{a, b, e\}$. This pair of places solves the remaining instances of SSP and ESSP, so we exit the whole procedure. We have produced a net with 5 transitions and 8 places.

14 Quality of Resulting Net

As we keep track on SSP and ESSP, the Petri net produced by our algorithm is indeed isomorphic to the input LTS if and only if all SSP and ESSP instances are solved. Since we add a place to the resulting net only if it contributes to SSP or ESSP, we have by construction that no place in our region net is redundant w.r.t. the places returned earlier. It may, however, be redundant if places produced later are taken into consideration. Hence, we cannot guarantee that our result is minimal. Furthermore, at the current stage of development, our resulting net contains a lot of self-loops (situations where $W([p, t]) = W([t, p]) \neq 0$) . In a refined algorithm, we could restrict insertion of self-loops to cases where that contributes to ESSP. However, we believe that such issues can be solved in post-processing where, given a set of places, redundant places and redundant self-loops can be eliminated. Then the resulting net would at least be minimal in the sense that no place is redundant.

On the other hand, we believe that our algorithm is *not* suitable for getting a net with minimal *number* of included places. For this goal, algorithms based on linear programming will probably remain the best solution.

15 Complexity

With the sketched procedure, we indeed achieve a space complexity of $O(|V||T|)$. The main data structures in our approach are

- the input LTS which takes $O(|V||T|)$ space since no vertex can have more than $|T|$ outgoing edges;
- the proto-region, an array with $|V|$ elements;
- the proto-signature that can be implemented in $O(|T|)$ exploiting the clique structure (in fact, we use union/find);

- the SPP record that takes $O(|V|)$ as sketched above;
- the ESSP record, an array with $|V||T|$ entries;
- a stack for managing decision points that cannot grow beyond size $|T|$ and contains only information of constant size. In particular, we refer to our solution for backtracking in proto-regions explained in Sect. 3.

Additional data structures for implementing the application of rules do not exceed this space requirement. To our best knowledge, the memory footprint of our algorithm is better than the memory usage of existing tools. Our experiments confirm this assumption. In fact, we were able to execute problem instances with our procedure on a machine with 16 GB RAM where competing tools ran out of memory on another machine with 1 TB of RAM.

Concerning run-time, only a tiny fraction of time is used for managing a proto-region. This is remarkable since we tried instances with more than one million vertices. The most time consuming part of our algorithm is the investigation of the data-structures for applicability of rules. All procedures implemented have a polynomial run time, though. The decisive factor for the run-time of our procedure is its branch-and-backtrack nature. Since we may run into branches that lead to inconsistent or redundant situations, our procedure has essentially exponential run-time. For further improvements of our algorithm, it is thus necessary to add additional and more powerful rules as well as heuristics such that the number of fruitless branches in our search tree is further reduced. In fact, instances that we cannot currently solve have search trees with a large number of recursive decision points. This number has some correlation to $|T|$ and, to a much smaller degree, to $|V|$.

16 Implementation

We implemented our results in a prototype. We call it `ptsynthia`. It reads an LTS from the input and writes the computed region net to the output. We implemented both proto-region and proto-signature using the union/find algorithm. The prototype supports all simple rules. We did not implement the full power of the rules TS and TR. We only support several special cases for which we found efficient implementations. The special cases are strong enough for making sure that, if no rule is applicable, the pre-requisites for the considerations in Sect. 11 are satisfied.

The implementation language is C++. We also implemented a modified prototype `synthia`. This prototype support the synthesis of safe Petri nets. In this tool, the restricted target class permits a lot of simplifications in the application of rules which we cannot describe in detail, due to the page limit. The whole project comprises of almost 5000 lines of code.

17 Experiments

We applied our tool to a benchmark of 105 labelled transition systems and compared our results to existing tools. The benchmark stems from the model

Table 3. Solved instances, depending on number of states.

Number of vertices	Total	`apt`		`ptsynthia`		`synthia`	
		Number	Percent	Number	Percent	Number	Percent
0 .. 9,999	46	43	93%	43	93%	44	96%
10,000 .. 100,000	15	13	87%	12	80%	13	87%
100,000 .. 1,000,000	18	3	17%	16	89%	16	89%
Beyond 1,000,000	26	0	0%	7	27%	18	69%
Total	105	59	56%	78	74%	91	87%

checking contest (MCC) [10] in 2016. We collected all Petri nets where the *state space* category of the contest reported less than 10 million reachable markings. We further included the smallest instance beyond that threshold. It has some 11 million reachable states. From this set, we removed some of the most trivial instances as well as some net instances that were mostly identical to other nets concerning net structure and number of reachable states. For the remaining nets, we used our tool LoLA [13] for generating the full state space. With a patch of the tool, we produced labelled transition systems in several formats. This includes the format for `synthia`, for `petrify` (which is also readable for `genet`), and `apt`. The decision to use MCC 2016 instead of MCC 2017 is motivated by the fact, that the organisers of the MCC removed many net instances with small state space in 2017 as they were considered to be too simple for verification.

The benchmark contains a small number of nets that are not Δ1-Petri nets. Most of them produce an LTS that is not synthesable (that is, there is no Δ1-Petri net producing that state space). We nevertheless report such instances as "solved" if the tool returns and reports open SSP or ESSP instances. About half of the nets are not safe and thus not synthesable by `synthia`. Again, we report the run-time for all instances. After some experimentation, we got the impression that, among the other tools, `apt` has the strongest performance. This impression coincides with existing literature [4]. Consequently, we ran the whole benchmark for `synthia`, `ptsynthia`, and `apt`. We executed the instances on a Mac-Book Pro with 4 cores, 2.2 GHz, and 16 GB RAM. For instances that ran out of memory, we repeated experiments on a LINUX machine with 64 cores, 2.7 GHz, and 1 TB of RAM. We gave each instance about 24 h of run time. On the LINUX machine, we executed about 20 instances in parallel. While `synthia` and `ptsynthia` are purely sequential tools, `apt` uses all available cores for parallelisation. `apt` offers several options that are relevant in our context:

- `plain` for producing ordinary nets
- `safe` for producing safe nets.

Run times vary, but in the end we concluded that `apt` performs best if applied without using such modifiers. Our experiments refer to that way of application.

Table 4. Solved instances, depending on number of transition labels.

Number of labels	Total	`apt`		`ptsynthia`		`synthia`	
		Number	Percent	Number	Percent	Number	Percent
0 .. 49	43	29	67%	40	93%	43	100%
50 .. 99	21	19	90%	21	100%	21	100%
100 .. 199	14	7	50%	9	64%	12	86%
200 .. 500	16	4	25%	6	38%	9	56%
Beyond 500	11	0	0%	2	18%	6	55%
Total	105	59	56%	78	74%	91	87%

Table 5. Number of solved instances, depending on consumed run time

Time	`apt`	`ptsynthia`	`synthia`
<1 min	40	40	59
1 min .. 1 h	18	23	24
1 h .. 10 h	0	8	7
10 h .. 24 h	0	7	1

Table 3 lists the number of instances solved by the compared tools, depending on the number of states of the models. Table 4 presents the same experiments, depending on the number of transition labels of the input LTS. In Table 5, we report on the consumed run time.

For only a handful of transition systems, `ptsynthia` and `synthia` needed more than 16 GB memory. They are the only tools that were able to break the sonic barrier of one million states, and even that was sometimes possible with less than 16 GB. In turn, `apt` did not solve any problem having more than 500,000 states. In comparison, the largest instance solved by `ptsynthia` has more than 9,000,000 vertices. `synthia` even solved the largest example in the benchmark. If all three tools solved a problem, we typically saw `synthia` coming in first and `apt` second. `apt` produces smaller nets and perhaps uses some of its run time to minimise the size of the net. The run time of `synthia` and `ptsynthia` depends only partly on the size of the net. Managing large sets of vertices is not an important issue, thanks to the union/find algorithm. We observed that success of our tools mostly depends on the depth of the search tree, that is, the number of subsequent decision points. Our run time is exponential in that number. Of course, the depth of the search tree has some correlation to the number of transitions in the net, so, in Table 4, the largest number of instances solved by our tools but not by `apt` is in the interval of 0 to 49 labels. However, there are instances where `ptsynthia` was successful on instances with several thousand transitions. In these instances, the rules performed above average thus reducing the depth of the search tree.

We can see that the $O(|V||T|)$ memory consumption is decisive for the success of the new algorithms. For the other tools, memory usage runs beyond limits much earlier than for our tools. With the larger memory consumption, run time grows due to the necessity to handle much larger data structures. In particular, Table 5 shows that the new tools more likely profit from additional time spent for computation.

The benchmark and our prototypes can be downloaded from https://users.informatik.uni-rostock.de/~ks249/pn2018.tar.

18 Conclusion and Future Work

We developed the idea of Petri net synthesis. Our main focus was a lean memory footprint. We achieved our goal by observing that synthesis can be viewed as a calculation process that is centered around partitions. This way, we can successfully apply the union/find algorithm.

There are several obvious sources for improvement. Much time is spent for checking for applicable rules. This task can easily be parallelised. We could also run several instances of our algorithm in parallel that synchronise only on the found regions and solved separation problems. If the instances use different strategies for selecting the transitions when branching, we increase the likelihood that all separation problems are covered earlier. We should work on better exploitation of the topological rules, and on finding additional rules. This way, the depth of the search tree can be reduced. We have an open issue for post-processing our results. We should remove places that finally turn out to be redundant, and we should remove self-loops if they are not needed for separation.

Apart from these obvious improvements, the concept of label splitting and the extension of the approach to arbitrary place/transition nets establish the natural next challenges.

References

1. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Mosses, P.D., Nielsen, M., Schwartzbach, M.I. (eds.) CAAP 1995. LNCS, vol. 915, pp. 364–378. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59293-8_207
2. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. TTCSAES. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47967-4
3. Badouel, E., Caillaud, B., Darondeau, P.: Distributing finite automata through Petri net synthesis. Form. Asp. Comput. **13**(6), 447–470 (2002)
4. Best, E., Schlachter, U.: Analysis of Petri nets and transition systems. In: Proceedings of the ICE, EPTCS, vol. 189, pp. 53–67 (2015)
5. Carmona, J., Cortadella, J., Kishinevsky, M.: Genet: a tool for the synthesis and mining of Petri nets. In: Proceedings of the ACSD, pp. 181–185 (2009)
6. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. IEICE Trans. Inf. Syst. **E80–D**(3), 315–325 (1997)

7. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets for finite transition systems. IEEE Trans. Comput. **47**(8), 859–882 (1998)
8. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. Part I: basic notions and the representation problem. Acta Inf. **27**(4), 315–342 (1990)
9. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. Part II: state spaces of concurrent systems. Acta Inf. **27**(4), 343–368 (1990)
10. Kordon, F., et al.: Complete Results for the 2016 Edition of the Model Checking Contest, June 2016. http://mcc.lip6.fr/2016/results.php
11. Mukund, M.: Petri nets and step transition systems. Int. J. Found. Comput. Sci. **3**(4), 443–478 (1992)
12. Schlachter, U.: Petri net synthesis for restricted classes of nets. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016. LNCS, vol. 9698, pp. 79–97. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39086-4_6
13. Schmidt, K.: LoLA: a low level analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 465–474. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44988-4_27
14. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. J. ACM **22**(2), 215–225 (1975)

Factorisation of Petri Net Solvable Transition Systems

Raymond Devillers[1] and Uli Schlachter[2(✉)]

[1] Université Libre de Bruxelles, Boulevard du Triomphe - C.P. 212, 1050 Bruxelles, Belgium
rdevil@ulb.ac.be

[2] Department of Computing Science, Carl von Ossietzky Universität, 26111 Oldenburg, Germany
schlachter@informatik.uni-oldenburg.de

Abstract. In recent papers, general conditions were developed to characterise when and how a labelled transition system may be factorised into non-trivial factors. These conditions combine a local property (strong diamonds) and a global one (separation), the latter being of course more delicate to check. Since one of the aims of such a factorisation was to speed up the synthesis of Petri nets from such labelled transition systems, the problem arises to analyse if those conditions (and in particular the global one) could be simplified, or even dropped, in the special case of Petri net solvable behaviours, i.e., when Petri net synthesis is possible. This will be the subject of the present paper.

Keywords: Labelled transition systems · Composition
Decomposition · Petri net synthesis

1 Introduction

The product of two labelled transition systems (lts) with disjoint label sets combines their behaviour by interleaving. Namely, the product can do an a-transition if the subsystem to which a belongs can. In factorisation, an lts is given which should be decomposed into disjoint subsystems, or factors, when possible.

In [16,17], conditions were devised to characterise when an lts is the disjoint product of non-trivial factors, how to check these conditions, and how to find an (optimal) decomposition. This uses essentially two types of properties: a general diamond property on pairs of labels and a separation property on subsets of labels. The first one is local, in the sense that it only relies on the close neighbourhood of the various states, hence is rather easy to check. The second

U. Schlachter—This author is supported by the German Research Foundation (DFG) project ARS (Algorithms for Reengineering and Synthesis), reference number Be 1267/15-1, and partially supported by DFG Research Training Group (DFG GRK 1765) SCARE.

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 82–98, 2018.
https://doi.org/10.1007/978-3-319-91268-4_5

one is global, needing to look at long distance relationships, hence is more delicate to check. Moreover, the first condition leads to an efficient way to partition the set of labels in such a way that labels from different components have the diamond property, by defining an equivalence relation on the label set so that only unions of equivalence classes may drive an adequate decomposition. However, the second condition allows no such treatment and instead the powerset of those classes has to be checked, leading to a potentially exponential blowup of the factorisation procedure.

The motivation for factorisation came from the Petri net synthesis problem, where an lts is given and one tries to find a Petri net whose behaviour corresponds to a reachability graph isomorphic to this lts. When ths is possible, one says the lts is solvable and the constructed Petri net is a solution. This problem was initially studied for elementary Petri nets [15,19,20], and later extended to the full class of Petri nets and some sub-classes like choice-free Petri nets or marked graphs [6,8,9]. The synthesis problem is usually polynomial in terms of the size of the lts, with degree between 2 and 5 depending on the subclass of Petri nets one searches for [3,4,8,12], but can also be NP-complete [5]. Hence the interest to apply a "divide and conquer" synthesis strategy. Moreover, the approach of [8] (extended to weighted marked graphs in [1]) only works for connected Petri nets. By factorising the given lts into prime (i.e., not further decomposable) factors, which automatically lead to connected solutions when solvable, this requirement can be lifted.

Hence, one could wonder if, in the specific framework of Petri net synthesis, the needed conditions and related algorithms could be simplified to avoid the exponential blowup and actually make synthesis faster. We shall see that this is indeed the case, and that the needed Petri net synthesis algorithms already perform the needed "long distance" checks.

The structure of the paper is as follows. After recalling the bases of the Petri net synthesis problem, disjoint products of lts are linked to disjoint sums of Petri nets. Then conditions for the factorisation of general lts are recalled, before the special case of factorisation in the context of Petri net synthesis is handled. This leads to a quick factorisation algorithm that may be run before proper synthesis starts, and its efficiency is illustrated on selected cases. The last section concludes.

2 Labelled Transition Systems and Petri Nets

A classical way for representing the possible (sequential) evolutions of a dynamic system is through its labelled transition system [2].

Definition 1. Labelled Transition Systems
A *labelled transition system (lts for short)* with initial state is a tuple $TS = (S, \rightarrow, T, \iota)$ with node (or state) set S, edge label set T, edges $\rightarrow \subseteq (S \times T \times S)$, and an initial state $\iota \in S$.

Two lts $TS_1 = (S_1, \to_1, T, \iota_1)$ and $TS_2 = (S_2, \to_2, T, \iota_2)$ with the same label set T are (state-)isomorphic, denoted $TS_1 \equiv_T TS_2$, if there is a bijection $\zeta\colon S_1 \to S_2$ with $\zeta(\iota_1) = \iota_2$ and $(s,t,s') \in \to_1 \Leftrightarrow (\zeta(s),t,\zeta(s')) \in \to_2$, for all $s, s' \in S_1$ and $t \in T$.

Usually, one considers the forward reachability relation: a label $t \in T$ is enabled at $s \in S$, written formally as $s[t\rangle$, if $\exists s' \in S\colon (s,t,s') \in \to$. A state s' is reachable from s through the execution of $\sigma \in T^*$, denoted by $s[\sigma\rangle s'$, if there is a directed path from s to s' whose edges are labelled consecutively by σ. The set of states reachable from s is denoted by $[s\rangle$.

But it is also possible to consider the forward-backward reachability relation: for each label $t \in T$ we shall denote by $-t$ the corresponding *reverse label*, i.e., $s[-t\rangle s'$ if $s'[t\rangle s$ (this may also be denoted $s\langle t]s'$). We shall assume that $-T = \{-t \mid t \in T\}$ is disjoint from T, and that $-t = t$.

Define $\pm T = T \cup -T$ to be the set of all forward and reverse labels. The general paths $s[\alpha\rangle s'$ for $\alpha \in (\pm T)^*$ are then defined like the forward ones, and we shall denote by $\langle s\rangle$ the set of states reachable from s through a general path.

We shall generalise the usual Parikh vectors to general paths: let $\alpha \in (\pm T)^*$ and $a \in T$, $\Psi(\alpha)(a)$ is defined inductively by

$$\Psi(\varepsilon)(a) = 0 \text{ and } \Psi(\tau t)(a) = \begin{cases} \Psi(\tau)(a) + 1 \text{ if } t = a \\ \Psi(\tau)(a) - 1 \text{ if } t = -a \\ \Psi(\tau)(a) \qquad \text{otherwise} \end{cases}$$

An lts is (forward) deterministic if $\forall t \in T, \forall s, s', s'' \in S : s[t\rangle s' \wedge s[t\rangle s'' \Rightarrow s' = s''$. Similarly, it is forward/backward (f/b for short) deterministic if $\forall t \in \pm T, \forall s, s', s'' \in S : s[t\rangle s' \wedge s[t\rangle s'' \Rightarrow s' = s''$, i.e., the reached state is additionally unique in backwards direction. It is strongly deterministic if $\forall \alpha, \alpha' \in (\pm T)^*, \forall s, s', s'' \in S : \Psi(\alpha) = \Psi(\alpha') \wedge s[\alpha\rangle s' \wedge s[\alpha'\rangle s'' \Rightarrow s' = s''$. This last definition means that two (general) paths with the same Parikh vector must reach the same state from s, e.g., permutations do not reach a different state.

A general cycle (around some state $s \in S$) is a general path $s[\alpha\rangle s$. Directed cycles are defined similarly, but using forward paths only.

Classical properties of an lts are:

1. It is finite if so are S and T (hence also $\to$).
2. It is totally reachable if $S = [\iota\rangle$, and generally reachable if $S = \langle\iota\rangle$ (then it is also connected).
3. TS is strongly cycle-consistent if, whenever there is a general path $s[\alpha\rangle s'$, general cycles $s_1[\beta_1\rangle s_1$, $s_2[\beta_2\rangle s_2, \ldots, s_n[\beta_n\rangle s_n$ and numbers $k_1, k_2, \ldots, k_n \in \mathbb{Q}$ such that $\Psi(\alpha) = \sum_1^n k_i \cdot \Psi(\beta_i)$, then $s = s'$. Intuitively, this means that a combination of Parikh vectors of cycles cannot produce non-cycles.

Definition 2. Petri Nets
An *initially marked Petri net* is denoted as $N = (P, T, F, M_0)$ where P is a set of places, T is a disjoint set of transitions ($P \cap T = \emptyset$), F is the flow function $F\colon ((P \times T) \cup (T \times P)) \to \mathbb{N}$ specifying the arc weights, and M_0 is the initial marking (where a marking is a mapping $M\colon P \to \mathbb{N}$, indicating the number of tokens in each place). Its *incidence matrix* is the matrix $C : (P \times T) \to \mathbb{N}$ defined as $C(p,t) = F(t,p) - F(p,t)$, i.e., the difference between the number of tokens produced and absorbed by t on p.

Two Petri nets $N_1 = (P_1, T, F_1, M_0^1)$ and $N_2 = (P_2, T, F_2, M_0^2)$ with the same transition set T are isomorphic, denoted $N_1 \equiv_T N_2$, if there is a bijection $\zeta\colon P_1 \to P_2$ such that, $\forall p_1 \in P_1, t \in T$, $M_0^1(p_1) = M_0^2(\zeta(p_1))$, $F_1(p_1,t) = F_2(\zeta(p_1),t)$ and $F_1(t,p_1) = F_2(t,\zeta(p_1))$.

A transition $t \in T$ is enabled at a marking M, denoted by $M[t\rangle$, if $\forall p \in P\colon M(p) \geq F(p,t)$. The firing of t leads from M to M', denoted by $M[t\rangle M'$, if $M[t\rangle$ and $M'(p) = M(p) - F(p,t) + F(t,p)$. This can be extended, as usual, to $M[\sigma\rangle M'$ for sequences $\sigma \in T^*$, and $[M\rangle$ denotes the set of markings reachable from M. The net is bounded if there is $k \in \mathbb{N}$ such that $\forall M \in [M_0\rangle, p \in P : M(p) \leq k$. Firing sequences may be generalised easily to sequences $\sigma \in (\pm T)^*$.

The reachability graph $RG(N)$ of N is the labelled transition system with the set of vertices $[M_0\rangle$, initial state M_0, label set T, and set of edges $\{(M,t,M') \mid M, M' \in [M_0\rangle \wedge M[t\rangle M'\}$. If an lts TS is isomorphic to the reachability graph of a Petri net N, we say that TS is *solvable* and that N *solves* TS. □ 2

Corollary 1. Independence from isomorphisms
Let N_1, N_2 be two Petri nets; if $N_1 \equiv_T N_2$, then $RG(N_1) \equiv_T RG(N_2)$.
Let TS_1, TS_2 be two lts; if $TS_1 \equiv_T TS_2$ and N solves TS_1, then N solves TS_2.
□ 1

But it may happen that two Petri nets with completely different structures have isomorphic reachability graphs, as illustrated in Fig. 1.

The classical state equation of Petri nets may be extended to general paths.

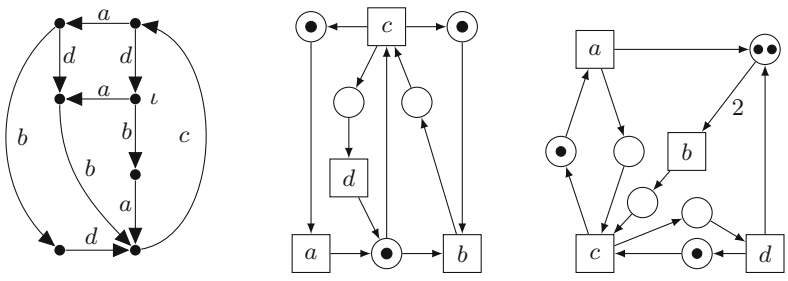


Fig. 1. A finite lts (on the left) and two possible Petri net solutions, with very different structures.

Lemma 1. General state equation
Let N be a Petri net with incidence matrix C, M_1, M_2 be two of its markings and $\alpha \in (\pm T)^*$ be some general firing sequence, then if $M_1[\alpha\rangle M_2$ we have $M_2 = M_1 + C \cdot \Psi(\alpha)$.

Proof. This immediately results, by induction on the length of α, from the definition of the firing rule. □ 1

Lemma 2. General properties of reachability graphs
Let $N = (P, T, F, M_0)$ be any Petri net. Its reachability graph $RG(N)$ is totally reachable, strongly deterministic and strongly cycle-consistent. It is finite iff N is bounded.

Proof. Total reachability holds by definition of the reachability graph. Similarly, boundedness and finiteness are equivalent by definition.

For strong determinism, assume that $M[\alpha\rangle M'$, $M[\alpha\rangle M''$, and $\Psi(\alpha) = \Psi(\alpha')$. By Lemma 1 we have $M' = M + C \cdot \Psi(\alpha) = M + C \cdot \Psi(\alpha') = M''$.

Strong cycle-consistency can be shown similarly with Lemma 1: $M_i[\beta_i\rangle M_i$ implies $C \cdot \Psi(\beta_i) = 0$, from which $C \cdot \Psi(\alpha) = 0$ can be deduced, thus $M = M'$. □ 2

Other kinds of properties, linked to infinite paths in those reachability graphs, are the following and are illustrated in Fig. 2:

Definition 3. Weak periodicity and two-way uniform chains An lts $TS = (S, \to, T, \iota)$ is weakly periodic if, for every $\alpha \in T^*$ and infinite path $s_1[\alpha\rangle s_2[\alpha\rangle s_3 \cdots$, either for every $i, j \in \mathbb{N}$: $s_i = s_j$ or for every $i, j \in \mathbb{N}$: $i \neq j \Rightarrow s_i \neq s_j$.

It presents a forward two-way uniform chain if there is a sequence $\alpha \in T^*$ and states $s_i \in S$ for $i \in \mathbb{Z}$ such that $s_i[\alpha\rangle s_{i+1}$ for each $i \in \mathbb{Z}$ and $\forall i, j \in \mathbb{Z} : (i \neq j \Rightarrow s_i \neq s_j)$. It presents a general two-way uniform chain if the same is true for some $\alpha \in (\pm T)^*$. □ 3

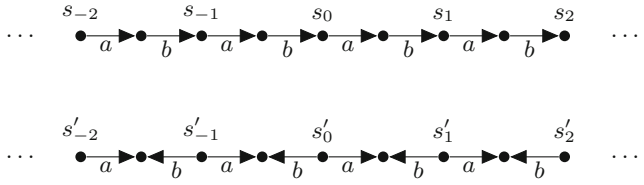


Fig. 2. A forward two-way uniform chain (top, with $\alpha = ab$), and a general one (bottom, with $\alpha = a\,\overline{b}$).

Lemma 3. Reachability graphs and infinite uniform chains
The reachability graph of a Petri net is weakly periodic and does not present two-way (forward or general) uniform chains.

Proof. Weak periodicity has been introduced and analysed in [7]. The proof for Lemma 10 of [7] also holds if, in the definition of weak periodicity we allow $\alpha \in (\pm T)^*$ instead of $\alpha \in T^*$. The proof follows from Lemma 1: Either a firing sequence α reproduces the marking ($C \cdot \Psi(\alpha) = 0$), or not.

For two-way uniform chains, the property also follows from Lemma 1. Indeed, let N be a Petri net with incidence matrix C. If there is a (forward or general) two-way uniform chain $\ldots M_i[\alpha\rangle M_{i+1}[\alpha\rangle M_{i+2} \ldots$ in its reachability graph, since $M_1 \neq M_2$, there is at least one place p such that the marking on it is strictly increased or decreased by α. From Lemma 1, the effect of α is the same on any marking: $\forall i : M_{i+1} - M_i = C \cdot \Psi(\alpha)$. Hence, if the marking on p is decreased, it is not possible to have such a chain going infinitely to the right. If it is increased, it is not possible to have such a chain going infinitely to the left. □ 3

3 Disjoint Product of lts and Disjoint Sum of Petri Nets

A product of two disjoint lts is again an lts. Its states are pairs of states of the two lts and an edge exists if one of the underlying states can do the transition. An example is shown in Fig. 3.

Definition 4. Product of two disjoint lts
Let $TS_1 = (S_1, \rightarrow_1, T_1, \iota_1)$ and $TS_2 = (S_2, \rightarrow_2, T_2, \iota_2)$ be two lts with disjoint label sets ($T_1 \cap T_2 = \emptyset$). The (disjoint) product $TS_1 \otimes TS_2$ is the lts $\big(S_1 \times S_2, \rightarrow, T_1 \uplus T_2, (\iota_1, \iota_2)\big)$, where $\rightarrow = \{\big((s_1, s_2), t_1, (s_1', s_2)\big) \mid (s_1, t_1, s_1') \in \rightarrow_1\} \cup \{\big((s_1, s_2), t_2, (s_1, s_2')\big) \mid (s_2, t_2, s_2') \in \rightarrow_2\}$. □ 4

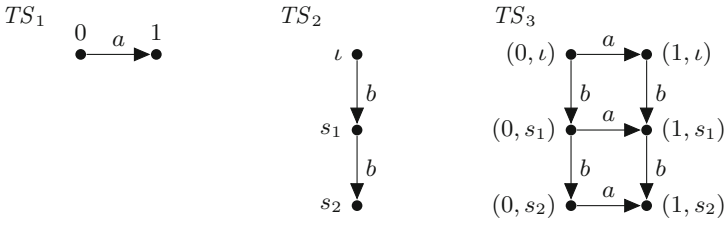


Fig. 3. Example for a disjoint product. We have $TS_1 \otimes TS_2 = TS_3$.

When a product is given and the individual label sets T_1 and T_2 are known, the factors can be computed by only following edges with labels in T_1, resp. T_2, from the initial state:

Corollary 2. Factors of a product
If $TS = (S, \rightarrow, T_1 \uplus T_2, \iota) \equiv_T TS_1 \otimes TS_2$ *with* TS_1 *and* TS_2 *generally reachable, then* $TS_1 \equiv_T \langle \iota \rangle^{T_1}$, *where* $\langle \iota \rangle^{T_1} = (S^1, \rightarrow^1, T_1, \iota)$ *with* $S^1 = \{s \in S \mid \exists \alpha_1 \in (\pm T_1)^* : \iota[\alpha\rangle s\}$ *and* $\rightarrow^1 = \{(s_1, t_1, s_2) \in \rightarrow \mid s_1, s_2 \in S^1, t_1 \in T_1\}$, *and similarly for* TS_2. □ 2

It is easy to see that, up to isomorphism[1], the disjoint product of lts is commutative, associative and has a neutral (the lts with a single state and no label).

There is an interesting relation between lts products and Petri nets: if two nets are disjoint, putting them side by side yields a new net whose reachability graph is (up to isomorphism) the disjoint product of the reachability graphs of the two original nets. This may be generalised up to Petri net isomorphism:

Definition 5. (Disjoint) sum of Petri nets
Let $N_1 = (P_1, T_1, F_1, M_0^1)$ and $N_2 = (P_2, T_2, F_2, M_0^2)$ be two Petri nets with disjoint transition sets ($T_1 \cap T_2 = \emptyset$). The disjoint sum $N_1 \oplus N_2$ is defined (up to isomorphism) as the net $N = (P, T_1 \cup T_2, F, M_0)$ where $P = \zeta_1(P_1) \cup \zeta_2(P_2)$; $F(\zeta_1(p_1), t_1) = F_1(p_1, t_1)$, $F(t_1, \zeta_1(p_1)) = F_1(t_1, p_1)$, $F(\zeta_2(p_2), t_2) = F_2(p_2, t_2)$, $F(t_2, \zeta_2(p_2)) = F_2(t_2, p_2)$, $M_0(\zeta_1(p_1)) = M_0^1(p_1)$ and $M_0(\zeta_2(p_2)) = M_0^2(p_2)$, for $t_1 \in T_1, t_2 \in T_2, p_1 \in P_1, p_2 \in P_2$. In these formulas, ζ_1 is a bijection between P_1 and $\zeta_1(P_1)$ and ζ_2 is a bijection between P_2 and $\zeta_2(P_2)$ such that $\zeta_1(P_1) \cap (\zeta_2(P_2) \cup T_1 \cup T_2) = \emptyset$ and $\zeta_2(P_2) \cap (\zeta_1(P_1) \cup T_1 \cup T_2) = \emptyset$. □ 5

It may be observed that the resulting net is not uniquely defined since it depends on the choice of the two bijections ζ_1 and ζ_2 used to separate the place sets from the rest, but this is irrelevant since we want to work up to isomorphism. Again, up to isomorphism, the disjoint sum of Petri nets is commutative, associative and has a neutral (the empty net).

An additional remark that may be precious for applications is that many subclasses of Petri nets found in the literature (free-choice, choice-free, join-free, fork-attribution, homogeneous,..., not defined here) are compatible with the presented (de)composition, in the sense that a disjoint sum of nets belongs to such a subclass if and only if each component belongs to the same subclass.

Corollary 3. Reachability graph of a sum of nets
The reachability graph of a disjoint sum of nets is isomorphic to the disjoint product of the reachability graphs of the composing nets. □ 3

There is a kind of reverse of this property [16,17].

Proposition 1. Petri net solution of a disjoint product of lts
A disjoint product of lts has a Petri net solution iff each composing lts has a Petri net solution, and a possible solution is the disjoint sum of the latter. □ 1

[1] Those properties could be expressed in terms of categories, but we shall refrain from doing this here.

4 Factorisation of a Connected lts

Let us now examine when an lts may be decomposed into (non-trivial) disjoint factors, and how to get them. These results arose in [16,17].

Definition 6. Label separation
An lts $TS = (S, \rightarrow, T, \iota)$ with $T_1, T_2 \subseteq T$ is $\{T_1, T_2\}$-separated if $\forall s, s' \in S, \forall \alpha \in (\pm T_1)^*$ and $\beta \in (\pm T_2)^*$: $s[\alpha\rangle s' \wedge s[\beta\rangle s' \Rightarrow s = s'$. □ 6

That means that, if $s \neq s'$, it is not possible to go from s to s' while using only labels (and reverse labels) from T_1 as well as only labels (and reverse labels) from T_2. Note that, trivially, the lts is always $\{\emptyset, T\}$-separated.

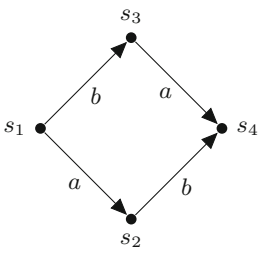
Proposition 2. Product implies separation
Let $TS_1 = (S_1, \rightarrow_1, T_1, \iota_1)$ and $TS_2 = (S_2, \rightarrow_2, T_2, \iota_2)$ be two lts with disjoint label sets, then $TS_1 \otimes TS_2$ is $\{T_1, T_2\}$-separated. □ 2

Another remarkable property of product lts is the following:

Definition 7. General diamond property
An lts $TS = (S, \rightarrow, T, \iota)$ presents the *general diamond property* for two distinct labels $a, b \in T$ if $\forall s, s_1, s_2 \in S$, $\forall u \in \{a, -a\}, \forall v \in \{b, -b\}$: $s[u\rangle s_1 \wedge s[v\rangle s_2 \Rightarrow (\exists s' \in S : s_1[v\rangle s' \wedge s_2[u\rangle s')$.

If $T_1, T_2 \subseteq T$ with $T_1 \cap T_2 = \emptyset$, TS will be said $\{T_1, T_2\}$-*gdiam* if it presents the general diamond property for each pair of labels $a \in T_1, b \in T_2$. □ 7

In other words, TS presents the general diamond property for $a \neq b \in T$ if whenever there are two adjacent edges in a diamond like in Fig. 4, the other two are also present. Note that any lts $TS = (S, \rightarrow, T, \iota)$ is $\{\emptyset, T\}$-gdiam.



$$s_1[a\rangle s_2 \ \wedge \ s_1[b\rangle s_3 \Rightarrow \exists s_4 : s_3[a\rangle s_4 \ \wedge \ s_2[b\rangle s_4$$
(forward persistence)

$$s_3[a\rangle s_4 \ \wedge \ s_2[b\rangle s_4 \Rightarrow \exists s_1 : s_1[a\rangle s_2 \ \wedge \ s_1[b\rangle s_3$$
(backward persistence)

$$s_1[a\rangle s_2 \ \wedge \ s_2[b\rangle s_4 \Rightarrow \exists s_3 : s_1[b\rangle s_3 \ \wedge \ s_3[a\rangle s_4$$
(permutation)

$$s_1[b\rangle s_3 \ \wedge \ s_3[a\rangle s_4 \Rightarrow \exists s_2 : s_1[a\rangle s_2 \ \wedge \ s_2[b\rangle s_4$$
(permutation)

Fig. 4. General diamond property.

Proposition 3. Projections and permutation
Let $TS = (S, \rightarrow, T, \iota)$ be a $\{T_1, T_2\}$-gdiam lts with $T_1 \cap T_2 = \emptyset$. If $s[\alpha\rangle s'$ for some $s, s' \in S$ and general path $\alpha \in (\pm T_1 \ \cup \ \pm T_2)^*$, let α_1 be the projection of α on T_1 (i.e., α where all the elements in $\pm T_2$ are dropped) and α_2 be the projection of α on T_2 (thus dropping the elements in $\pm T_1$). Then there are $s_1, s_2 \in S$ such that $s[\alpha_1\rangle s_1[\alpha_2\rangle s'$ and $s[\alpha_2\rangle s_2[\alpha_1\rangle s'$. □ 3

This also implies a variant of the well-known Keller's theorem [21].

Proposition 4. General diamonds imply big general diamonds
Let $TS = (S, \to, T, \iota)$ be a $\{T_1, T_2\}$-gdiam lts with $T_1 \cap T_2 = \emptyset$. If $s[\alpha_1\rangle s_1$ and $s[\alpha_2\rangle s_2$ for some $s, s_1, s_2 \in S$, $\alpha_1 \in (\pm T_1)^*$ and $\alpha_2 \in (\pm T_2)^*$, then for some $s' \in S$, $s_1[\alpha_2\rangle s'$ and $s_2[\alpha_1\rangle s'$. □ 4

This may be interpreted as the fact that big general diamonds are filled with small ones. Now, the interest of the notion of general diamonds arises from the following observation:

Proposition 5. Product implies general diamonds
Let $TS_1 = (S_1, \to_1, T_1, \iota_1)$ and $TS_2 = (S_2, \to_2, T_2, \iota_2)$ be two lts with disjoint label sets, then $TS_1 \otimes TS_2$ presents the general diamond property for each $a \in T_1$ and $b \in T_2$. □ 2

Unfortunately, the reverse property does not hold in all generality, as shown by the counterexamples in Fig. 5. On the left, in TS_4, a and b form general diamonds, however the lts is not a product of two lts based on label sets $\{a\}$ and $\{b\}$, since the general paths $\iota[b\rangle s_2$ and $\iota[a - a\rangle s_2$ show that the lts is not $\{\{a\}, \{b\}\}$-separated, hence contradicting Proposition 2. On the right, in TS_5, a and b also form general diamonds, but we have $s_5[b - b\rangle s_2$ and $s_5[-aa\rangle s_2$, contradicting again Proposition 2. But note that both counterexample are non-deterministic, hence cannot have Petri net solutions.

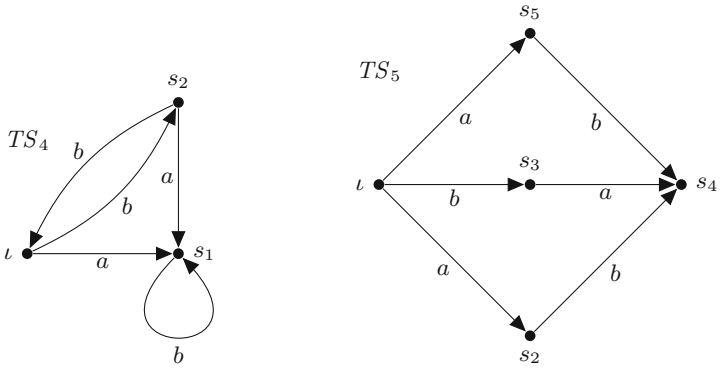


Fig. 5. General diamond property does not imply product.

In the following, if $s \in S$ and $T' \subseteq T$, we shall denote by $\langle s\rangle_{T'}$ the subset of states generally reachable from s while only using labels from T', i.e., $\{s' \in S \mid s[\alpha\rangle s' \text{for some } \alpha \in (\pm T')^*\}$, and by $\langle s\rangle^{T'}$ the corresponding sub-system, i.e., $(\langle s\rangle_{T'}, \to_{T'}, T', s)$, where $\to_{T'} = \{s'[a\rangle s'' \in \to \text{ with} s', s'' \in \langle s\rangle_{T'}, a \in T'\}$.

The main result of [16,17] is then:

Theorem 1. General diamonds and separation imply product
Let $TS = (S, \to, T, \iota)$ be a connected lts with $T = T_1 \uplus T_2$.

There are two lts $TS_1 = (S_1, \rightarrow_1, T_1, \iota_1)$ *and* $TS_2 = (S_2, \rightarrow_2, T_2, \iota_2)$ *such that* $TS \equiv_T TS_1 \otimes TS_2$ *iff* TS *is* $\{T_1, T_2\}$*-separated and* $\{T_1, T_2\}$*-gdiam. Moreover, we then have* $TS_1 \equiv_{T_1} \langle \iota \rangle^{T_1}$ *and* $TS_2 \equiv_{T_2} \langle \iota \rangle^{T_2}$. □ 1

5 The Petri Net Synthesis Case

Given an lts and two disjoint sets T_1 and T_2 of labels, it is easy to check if the lts is $\{T_1, T_2\}$-gdiam: For each state a property that only considers its immediate neighborhood has to be examined. We call such a property *local*. In contrast, checking if the lts is $\{T_1, T_2\}$-separated has to consider paths of arbitrary length and is *global*.

Let us now re-examine the previous results in the framework of Petri net synthesis. The main result of this paper is that the local property suffices:

Theorem 2. General diamonds and Petri net synthesis imply separation
If a totally reachable lts $TS = (S, \rightarrow, T_1 \uplus T_2, \iota)$ *is f/b deterministic and satisfies the general diamond property for each pair of labels* $a \in T_1$ *and* $b \in T_2$*, then it is Petri net synthesisable iff so are* $\langle \iota \rangle^{T_1}$ *and* $\langle \iota \rangle^{T_2}$*; moreover, we then have* $TS \equiv_T \langle \iota \rangle^{T_1} \otimes \langle \iota \rangle^{T_2}$ *and therefore a possible solution of the synthesis problem for* TS *is the disjoint sum of a solution of* $\langle \iota \rangle^{T_1}$ *and a solution of* $\langle \iota \rangle^{T_2}$.

Proof. Let us first assume TS is Petri net synthesisable. In particular, that means by Lemmas 2 and 3 that it is weakly periodic, has no general two-way uniform chains and is strongly cycle-consistent. If it has the general diamond property for $\{T_1, T_2\}$, let us show it is also $\{T_1, T_2\}$-separable. Let us assume that for some $s_0, s_1 \in S$, $\alpha \in (\pm T_1)^*$ and $\beta \in (\pm T_2)^*$, we have $s_0[\alpha\rangle s_1 \wedge s_0[\beta\rangle s_1$. Then, from Proposition 4, we have for some s_2 as well as some s_{-1}: $s_1[\alpha\rangle s_2 \wedge s_1[\beta\rangle s_2 \wedge s_{-1}[\alpha\rangle s_0 \wedge s_{-1}[\beta\rangle s_0$, and we may continue the construction forward as well as backward. Since the reachability graph of a Petri net has no general two-way uniform chains (Lemma 3), we must have $s_i = s_j$ for some $i \neq j \in \mathbb{Z}$. But then, from the strong cycle consistency (as well as the weak periodicity), we must have $s_i = s_j \ \forall i, j \in \mathbb{Z}$, and in particular $s_0 = s_1$, hence the separation property for $\{T_1, T_2\}$. From Theorem 1 and Proposition 1, we thus have that a possible solution of the synthesis problem is the disjoint sum of the solutions of $\langle \iota \rangle^{T_1}$ and $\langle \iota \rangle^{T_2}$.

Conversely, let us assume that $\langle \iota \rangle^{T_1}$ and $\langle \iota \rangle^{T_2}$ are Petri net solvable. Let us show TS satisfies the separation property for $\{T_1, T_2\}$. Like in the first part, since by hypothesis we have the general diamond property, if for some $s_0, s_1 \in S$, $\alpha \in (\pm T_1)^*$ and $\beta \in (\pm T_2)^*$ we have $s_0[\alpha\rangle s_1 \wedge s_0[\beta\rangle s_1$, we may construct a structure $s_i[\alpha\rangle s_{i+1} \wedge s_i[\beta\rangle s_{i+1}$. Since the lts is totally reachable, let us assume that $\iota[\gamma\rangle s_0$ for some $\gamma \in T^*$, with γ_1 being its projection on T_1 and γ_2 being its projection on T_2.

From Proposition 3, for each $i \in \mathbb{Z}$ we have $\iota[\gamma_1 \alpha^i\rangle s_i'[\gamma_2\rangle s_i$ for some s_i'. From the f/b determinism, each s_i' is unique, and since $s_i'[\gamma_2\rangle s_i$ and $s_j'[\gamma_2\rangle s_j$ for any $i, j \in \mathbb{Z}$, $s_i' = s_j'$ iff $s_i = s_j$; hence all the s_i''s are distinct iff so are all the s_i's. But

in $\langle \iota \rangle^{T_1}$, $s'_i[\alpha\rangle s'_{i+1}$ for each $i \in \mathbb{Z}$. Since $\langle \iota \rangle^{T_1}$ is Petri net solvable, from Lemma 3 this implies that $s'_i = s'_j$ for some $i \neq j \in \mathbb{Z}$ (no two-way general uniform chains) and $s'_i = s'_j$ for all $i, j \in \mathbb{Z}$ (weak periodicity), and consequently $s_i = s_j$ for all $i, j \in \mathbb{Z}$. In particular $s_0 = s_1$, hence the claimed separation property. And from Theorem 1 and Proposition 1, we thus have that a possible solution of the synthesis problem is the disjoint sum of the solutions of $\langle \iota \rangle^{T_1}$ and $\langle \iota \rangle^{T_2}$. □ 2

In some sense that means that the missing (global) separation property is hidden in the Petri net synthesis of the sub-systems $\langle \iota \rangle^{T_1}$ and $\langle \iota \rangle^{T_2}$, when this succeeds.

It remains to find adequate subsets of labels T_1 and T_2, partitioning T and satisfying the general diamond property. To do that, one may rely on the following, which is again a local property.

Definition 8. Connected labels
Let $TS = (S, \to, T, \iota)$ be an lts and $a, b \in T$ be two distinct labels.
We shall denote by $a \leftrightarrow b$ the fact that they do not form general diamonds, i.e., there are states $s, s_1, s_2 \in S$, $s_1[u\rangle s_2$, $s_1[v\rangle s_3$ with $u \in \{a, -a\}$ and $v \in \{b, -b\}$ such that $(\nexists s_4 : s_2[v\rangle s_4 \wedge s_3[u\rangle s_4)$.

We shall also note $\not\Diamond^= \leftrightarrow^*$, i.e., the reflexive and transitive closure of $\leftrightarrow$, meaning in some sense that the labels are 'non-diamondisable'. □ 8

These relations mean that in any decomposition, if $a \leftrightarrow b$ they must belong to the same component, i.e., $a \in T_1 \Rightarrow [a] \subseteq T_1$, where $[a] = \{b \in T \mid a \not\Diamond b\}$. But from Theorem 2, we know this is enough: for each equivalence class, either the synthesis works and we have a global solution by taking the disjoint sum of all the solutions, or one (or more) of the subproblems fails, and we know there is no global solution for the whole system.

Our proposed algorithm now works as follows: First, iterate over all states of the given lts, and for each state check if the adjacent edges form general diamonds. If not, their labels must be in the same equivalence class. Then, for each equivalence class $[a]$ try Petri net synthesis on $\langle \iota \rangle^{[a]}$. If it works, the result is the disjoint sum of the computed Petri nets. This constructs the equivalence relation by repeatedly joining classes, but it also allows to stop the iteration early when only one equivalence class remains.

Concerning the estimation of the efficiency of this procedure, we may observe that, from the forward and the backward determinism, any state can be connected to $2 \cdot |T|$ edges (each label may occur once in forward and backwards direction). Checking the presence of general diamonds requires examining each pair of these edges, so requires time in $\mathcal{O}(|T|^2)$ for a each state and $\mathcal{O}(|S| \cdot |T|^2)$ for the full lts. However, in practice, in the non-factorisable case, the procedure is likely to be a lot faster since one may stop as soon as it is discovered that there is a single equivalence class; in this case the factorisation procedure is then useless, but the extra burden may be expected to be negligible with respect to the time of the proper synthesis. For a factorisable lts, if there are f factors approximately of the same size, the size of each subsystem to synthesise is the

fth root of the original size. This divides the polynomial degree of the synthesis by f, i.e., replacing $\mathcal{O}(|S|^d)$ with $\mathcal{O}(f \cdot |S|^{d/f})$. However, this concerns the worst case complexity analysis and the true one may be rather different, which is the reason why we conducted various experiments, a summary of which will be described in the next section. Note also that if the factorisation procedure is placed at the end of the pre-synthesis phase, that checks the structure of the lts to determine if there is a chance that the synthesis succeeds (see [12]), the factorisation will not be entered at all if a failure is detected before.

6 Experimental Results

The proposed algorithm was implemented and its effect on Petri net synthesis was measured. For this, several families of Petri nets were used. The reachability graph of such a Petri net was calculated and then the running time of several algorithms was measured: (1) Re-synthesising the lts into a (possibly different) Petri net. (2) Factorising the lts without attempting synthesis. (3) Factorising the lts and synthesising its factors.

For Petri net synthesis, an existing implementation in APT [13,14] based on an algorithm from [3] was used. Previous experiments [12] showed that its performance is similar to other existing implementations of Petri net synthesis.

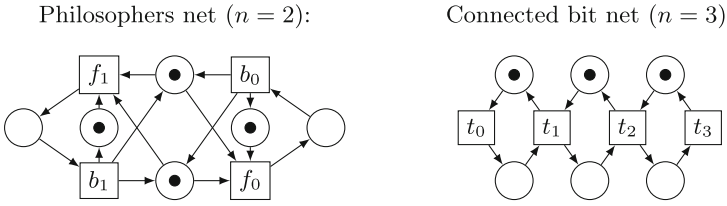


Fig. 6. Instances of the net families. In the philosophers net, transition f indicates that forks are taken, while b describes putting the forks back.

We considered many net families[2], where the size is controlled by a parameter n, but the results were very similar so that we only present here the following families of nets (see Fig. 6):

- Philosophers nets: This is Dijkstra's dining philosophers example [18] modelled as a Petri net. Each of the n philosophers can be either thinking or eating. To eat, a philosopher grabs his left and right forks atomically. When a philosopher starts to think again, he releases his forks. The state space of such a Petri net has between $2^{\lfloor \frac{n}{2} \rfloor}$ and 2^n reachable markings. The upper bound corresponds to the case where each philosopher can freely visit his two states. The lower bound corresponds to the case where each even philosopher thinks while the odd ones are in one of their two states.

[2] For example, the well-known KANBAN example, or Petri nets with k tokens, n transitions and n places arranged in a cycle.

- Connected bit nets: A bit can be in one of two states, *set* and *unset*, and there are n such bits. Initially, all bits are unset. The lowest bit has a transition to set it and the highest bit has a transition to unset it. For all other bits, there is a transition unsetting bit i and simultaneously setting bit $i+1$ for $0 < i < n$. The reachability graph of such a Petri net has 2^n reachable states and is one of the many possible implementations of an n-bit buffer.

Since these examples cannot be factored, the benchmarks actually used the disjoint sum of two such nets. This means there are 2^{2n} reachable states in the sum of two connected n-bit nets. A similar behaviour arises for the philosopher nets; due to this fast growth, we decided not to use the classical model of philosophers were forks are picked up one after another[3].

We also want to show that the factorisation algorithm has a low overhead even when it is useless, i.e., when factorisation is not possible. For this, we generated another variant of the examples: All tokens were removed from the initial marking and a new transition was added that can only fire once and which produces the original initial marking. The effect of this modification is to make factorisation fail while preserving the size of the reachability graph.

Another possibility, that we however did not measure, is that a factorisation is found, but the resulting Petri net synthesis fails for one of the factors. The performance of such an example would depend on the order in which the factors are synthesised, which is non-deterministic. Also, since the factorisation is not interwoven with synthesis, this case would likely have similar characteristics to the case with successful factorisation and synthesis.

The benchmarks were performed in Java and their results are shown in Figs. 7, 8, 9 and 10 using semi-logarithmic plots.

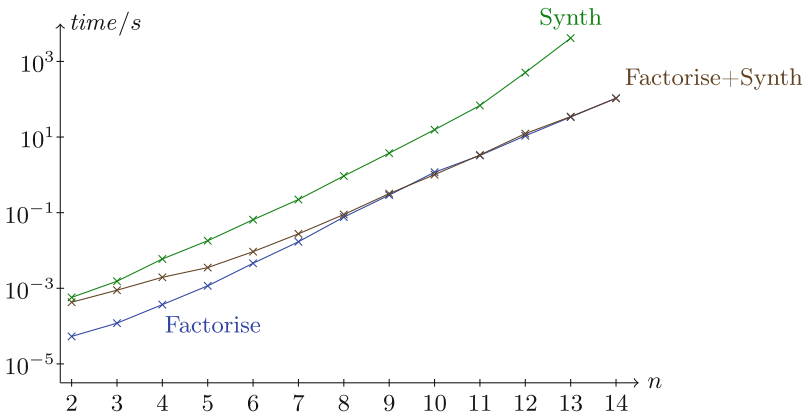


Fig. 7. Runtime for the disjoint sum of two philosophers of size n.

[3] Otherwise, only reachability graphs for $n \leq 8$ could be measured before running out of memory on the computer used for measurements.

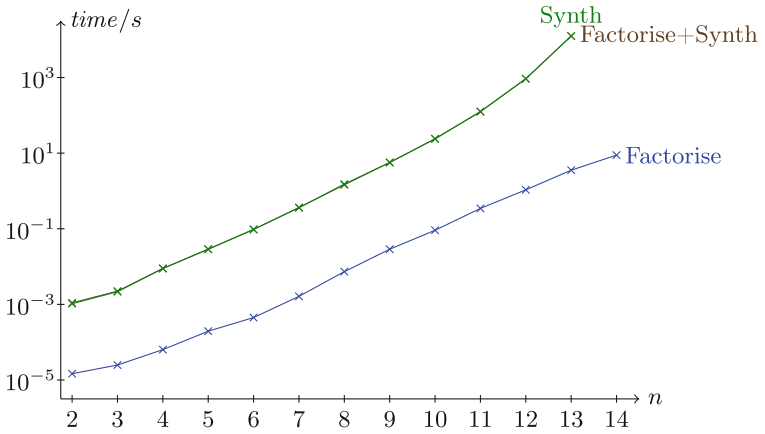


Fig. 8. Runtime for the disjoint sum of two philosophers of size n with an additional *init* transition.

Philosophers Nets. The results for the philosophers nets are shown in Fig. 7. The curve labelled *Synth* shows the time for just Petri net synthesis, *Factorise* is the time just to factorise the input without attempting synthesis, and *Factorise* + *Synth* first factorises the input and then synthesises the factors.

It can be seen that just factorisation is consistently at least ten times faster than just synthesis. Also, the factors are so much smaller than the original input that eventually the curves for Factorise and Factorise + Synth become indistinguishable.

Figure 8 shows the results when making the input non-factorisable by adding an *init* transition. Compared to the previous graph, the Factorise curve is lower, showing that the algorithm notices early on that factorisation is not possible and fails quickly. Synthesis is roughly a 100 times slower than factorisation, but this factor becomes higher with larger inputs. This means that the overhead due to a useless factorisation is low.

Connected Bit Nets. Figure 9 shows the results for connected bit nets. As in the previous example, factorisation is at least ten times faster than synthesis and the factors are a lot easier to synthesise than the original input, making factorisation the slowest part of Synth + Factorise.

Similar to the other example, adding an *init* transition shows that the factorisation algorithm has a very low overhead for non-factorisable inputs. These results are shown in Fig. 10. Factorisation is about a hundred times faster than synthesis and this factor becomes larger for larger inputs. Thus, compared to the time for synthesis, the failed attempt at factorisation takes almost no time. For $n = 9$, one can see that factorisation together with synthesis is noticeably slower than just synthesis. However, measurements varied a lot for this lts with some runs taking twice as long as others. Thus, we believe this is due to the

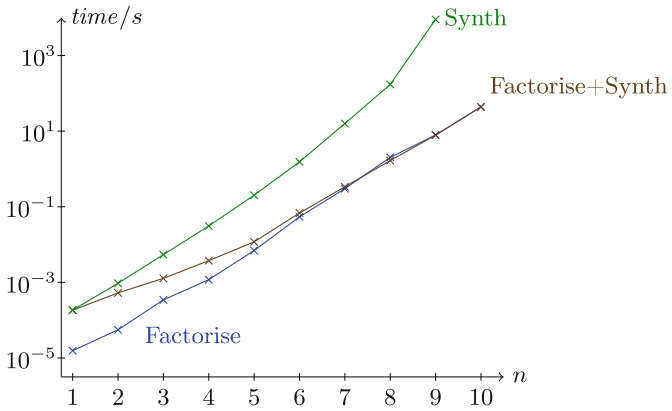


Fig. 9. Runtime for the disjoint sum of connected bit nets of size n.

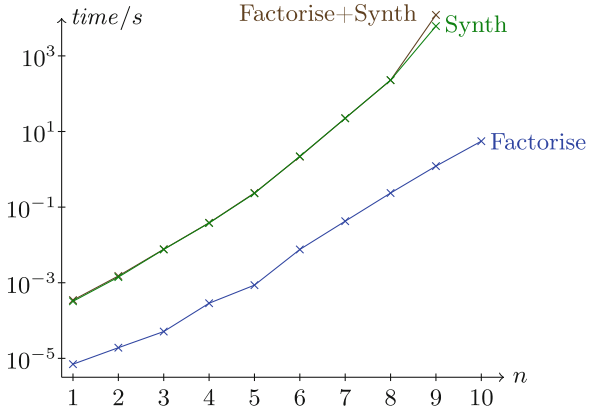


Fig. 10. Runtime for the disjoint sum of connected bit nets of size n with an additional *init* transition.

fact that the available memory was almost exhausted, which means that Java's garbage collector was very active and used a lot of processing time.

7 Concluding Remarks

We have shown how lts factorisation may be rendered very efficient in the context of (bounded) Petri net synthesis. We thus suggest to systematically incorporate it in the pre-synthesis procedures, that first check if needed structural properties like total reachability and f/b determinism (plus others when focussing on a subclass of nets [10,12]) are satisfied. When factorisation succeeds, the speed up is usually noticeable. When it fails, only a negligible time was needed. Anyway, factorisation is needed when using highly optimised synthesis algorithms for specific net subclasses needing connectedness of the result.

Additional works in this direction could be to analyse when and how an lts may be decomposed into a disjoint sequence, allowing to detect cases like the ones used for Figs. 8 and 10, which correspond to a sequence of a single init transition and the original lts used for Figs. 7 and 9, respectively. Other kinds of net and lts operators could also be considered, like loops, choices,... in the spirit of [11].

Acknowledgements. We would like to thank Valentin Spreckels for his help in implementing the factorisation. The anonymous referees made interesting comments, and asked questions that helped improving the presentation.

References

1. Analysis and Synthesis of Weighted Marked Graph Petri Nets (2018)
2. Arnold, A.: Finite Transition Systems - Semantics of Communicating Systems. Prentice Hall International Series in Computer Science. Prentice Hall, Upper Saddle River (1994)
3. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. TTCSAES. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47967-4
4. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Mosses, P.D., Nielsen, M., Schwartzbach, M.I. (eds.) CAAP 1995. LNCS, vol. 915, pp. 364–378. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59293-8_207
5. Badouel, E., Bernardinello, L., Darondeau, P.: The synthesis problem for elementary net systems is NP-complete. Theor. Comput. Sci. **186**(1–2), 107–134 (1997)
6. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_22
7. Best, E., Darondeau, P.: A decomposition theorem for finite persistent transition systems. Acta Inf. **46**(3), 237–254 (2009). http://dx.doi.org/10.1007/s00236-009-0095-6
8. Best, E., Devillers, R.: Characterisation of the state spaces of live and bounded marked graph Petri nets. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) LATA 2014. LNCS, vol. 8370, pp. 161–172. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04921-2_13
9. Best, E., Devillers, R.: Synthesis of bounded choice-free Petri nets. In: Aceto, L., de Frutos-Escrig, D. (eds.) CONCUR 2015. LIPIcs, vol. 42, pp. 128–141. Schloss Dagstuhl (2015). https://doi.org/10.4230/LIPIcs.CONCUR.2015.128
10. Best, E., Devillers, R.: Pre-synthesis of Petri nets based on prime cycles and distance paths. Science of Computer Programming (2017)
11. Best, E., Devillers, R., Koutny, M.: Petri Net Algebra. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2001). https://doi.org/10.1007/978-3-662-04457-5
12. Best, E., Devillers, R., Schlachter, U.: Bounded choice-free Petri net synthesis: algorithmic issues. Acta Informatica (2017). https://doi.org/10.1007/s00236-017-0310-9
13. Best, E., Schlachter, U.: APT: analysis of Petri nets and transition systems. https://github.com/CvO-Theory/apt

14. Best, E., Schlachter, U.: Analysis of Petri nets and transition systems. In: Knight, S., Lanese, I., Lluch-Lafuente, A., Vieira, H.T. (eds.) ICE 2015. EPTCS, vol. 189, pp. 53–67 (2015). https://doi.org/10.4204/EPTCS.189.6
15. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets for finite transition systems. IEEE Trans. Comput. **47**(8), 859–882 (1998). http://dx.doi.org/10.1109/12.707587
16. Devillers, R.: Products of transition systems and additions of Petri nets. In: Desel, J., Yakovlev, A. (eds.) ACSD 2016, pp. 65–73. IEEE Computer Society (2016). https://doi.org/10.1109/ACSD.2016.10
17. Devillers, R.: Factorisation of transition systems. Acta Informatica (2017). https://doi.org/10.1007/s00236-017-0300-y
18. Dijkstra, E.W.: Hierarchical ordering of sequential processes. Acta Informatica **1**, 115–138 (1971). https://doi.org/10.1007/BF00289519
19. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. Part I: basic notions and the representation problem. Acta Informatica **27**(4), 315–342 (1990). http://dx.doi.org/10.1007/BF00264611
20. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. Part II: state spaces of concurrent systems. Acta Informatica **27**(4), 343–368 (1990). http://dx.doi.org/10.1007/BF00264612
21. Keller, R.M.: A fundamental theorem of asynchronous parallel computation. In: Feng, T. (ed.) Parallel Processing. LNCS, vol. 24, pp. 102–112. Springer, Heidelberg (1975). https://doi.org/10.1007/3-540-07135-0_113

A Geometric Characterisation of Event/State Separation

Uli Schlachter(✉) and Harro Wimmel

Department of Computing Science, Carl von Ossietzky Universität, 26111 Oldenburg, Germany
{uli.schlachter,harro.wimmel}@informatik.uni-oldenburg.de

Abstract. Region theory, as initiated by Ehrenfeucht and Rozenberg, allows the characterisation of the class of Petri net synthesisable finite labelled transition systems. Two kinds of problems need to be solved for such a synthesis, state separation problems for distinguishing states and event/state separation problems for preventing unwanted behaviour. In the present paper, the class of finite labelled transition systems in which all event/state separation problems are solvable shall be characterised geometrically, rather than linear-algebraically.

1 Introduction

The linear algebra-based synthesis algorithm described by Badouel et al. [1] allows to check whether a given edge-labelled transition system is isomorphic to the state graph of an unlabelled Petri net, and if so, to construct such a net. However, in case of a negative answer, the explanation for this failure is the unsolvability of some inequality system as a whole (and not some specific inequality). This makes it difficult to pinpoint the failure and it may therefore be interesting to characterise the solvability of separation problems in a linear algebra-independent way with an easier interpretation of failures.

The synthesis algorithm solves two different kinds of *separation problems*. Roughly speaking, a state separation problem (SSP) consists of two different states which must have distinguishable markings in the Petri net solution while an event/state separation problem (ESSP) describes an unwanted edge at some state that must be prevented by the corresponding marking in the solution.

A previous paper [4] provided a graph-theoretical characterisation of state separation (SSP). The present paper deals with the second kind of problems (ESSP) and provides a geometric characterisation of event/state separation. An important role in this characterisation will be played by the convex hull of a set of points, which is a geometric concept with a linear-algebraic foundation.

The structure of the paper is as follows: after recalling some notions about labelled transition systems and about regions in Sects. 2 and 3, Sects. 4 and 5

This work is supported by the German Research Foundation (DFG) projects ARS and ASYST, reference numbers Be 1267/15-1 and Be 1267/16-1, and partially supported by DFG Research Training Group (DFG GRK 1765) SCARE.

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 99–116, 2018.
https://doi.org/10.1007/978-3-319-91268-4_6

present our geometric characterisation for pure[1], respectively general, Petri nets. Examples for this characterisation appear in Sect. 6. In Sect. 7 a restriction to alphabets of size two and a generalisation to infinite lts are considered. Finally some concluding remarks are presented in Sect. 8.

2 Labelled Transition Systems

A labelled transition system can be understood as an edge-labelled directed graph with an initial state.

Definition 1 (lts, walks, reachability). *A labelled transition system with initial state, abbreviated lts, is a quadruple* $TS = (S, T, \to, \imath)$ *where* S *is a set of* states, T *is a set of* labels, $\to \subseteq (S \times T \times S)$ *is the* transition relation, *and* $\imath \in S$ *is an* initial state.

A walk $s[\sigma\rangle s'$ *consists of two states* $s, s' \in S$ *and a word* $\sigma = a_1 \dots a_m \in T^*$ *such that*

$$\exists r_0, r_1, \dots, r_m \in S\colon s = r_0 \wedge r_m = s' \wedge \forall j \in \{1, \dots, m\}\colon (r_{j-1}, a_j, r_j) \in \to$$

Note that $s[\varepsilon\rangle s'$ *is tantamount to* $s = s'$.

A state s enables *a word* σ, *written* $s[\sigma\rangle$, *if there is a state* s' *with* $s[\sigma\rangle s'$. *A state* s' *is* reachable *from a state* s *if* $\exists \sigma \in T^*\colon s[\sigma\rangle s'$.

Some examples for transition systems are depicted in Fig. 1.

Definition 2 (Parikh vectors and cycles). *A* T-vector *is a function* $\Phi\colon T \to \mathbb{Z}$; *in linear algebra, it will usually be considered as a column-vector. For a word* $\sigma \in T^*$, *the* Parikh vector of σ *is a* T*-vector* $\Psi(\sigma)$, *defined inductively as follows:*

$$\Psi(\varepsilon) = \mathbf{0} \text{ (the null vector)}$$

$$\Psi(\sigma a)(t) = \begin{cases} \Psi(\sigma)(t) + 1 & \text{if } t = a \in T \\ \Psi(\sigma)(t) & \text{if } t \neq a \end{cases}$$

A walk $s[\sigma\rangle s'$ *is called a* cycle, *or more precisely a* cycle at (or around) state s, *if* $s = s'$.

Definition 3 (Spanning tree, equivalences). *A transition system* $TS = (S, T, \to, \imath)$ *is called* finite *if* S *and* T *(hence* $\to$*) are finite, and it is called* totally reachable *if every state is reachable from* $\imath$. *A finite lts* TS *is a* tree *(with* root $\imath$*) if it is totally reachable and* $|\to| = |S| - 1$ *(i.e., for each state, there is a single walk from the root to it). A tree* $TS_0 = (S, T, \to_0, \imath)$ *is called a* spanning tree *of* TS *if* $\to_0 \subseteq \to$. *The* language *of* TS *is the set* $L(TS) = \{\sigma \in T^* \mid \imath[\sigma\rangle\}$.

Two lts $TS_1 = (S_1, T, \to_1, \imath_1)$ *and* $TS_2 = (S_2, T, \to_2, \imath_2)$ *are called* language-equivalent *if* $L(TS_1) = L(TS_2)$, *and* isomorphic *if there is a bijection* $\zeta\colon S_1 \to S_2$ *with* $\zeta(\imath_1) = \imath_2$ *and* $(s, t, s') \in \to_1 \Leftrightarrow (\zeta(s), t, \zeta(s')) \in \to_2$, *for all* $s, s' \in S_1$.

[1] Any place/transition pair can be connected by an edge in one direction only.

As an example, consider the labelled transition system TS_1 depicted in Fig. 1. The walk $\imath[aba\rangle$ emanating from state $\imath$ (more precisely, exhibiting all intermediate states, $\imath[a\rangle s_1[b\rangle\imath[a\rangle s_1)$, has a Parikh vector mapping a to 2 and b to 1. TS_1 has a unique spanning tree containing $\imath[a\rangle s_1$ as its only edge. Of the five examples, only TS_5 allows two different spanning trees, both consisting of the edge $\imath[b\rangle s_1$ and exactly one of the a-edges. TS_3 is not isomorphic to TS_1, but it is language-equivalent as both lts have the prefixes of $(ab)^*$ as their language.

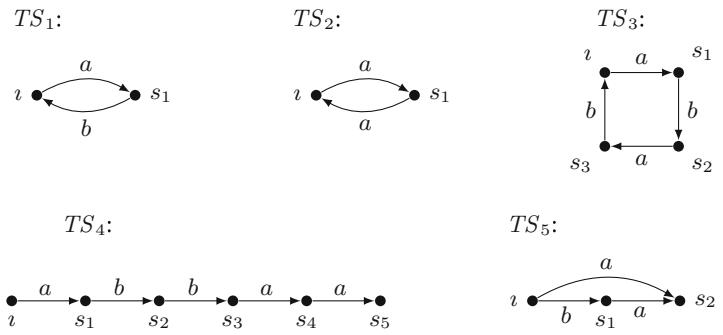


Fig. 1. Five example transition systems.

Definition 4 (π_s, chord, cycle basis). *Let $TS_0 = (S, T, \rightarrow_0, \imath)$ be a spanning tree of $TS = (S, T, \rightarrow, \imath)$. For any state s, let $\imath[\pi_s\rangle s$ denote the (unique) walk from $\imath$ to s in TS_0 (which is also a walk in TS). An edge $(s, t, s') \in (\rightarrow \setminus \rightarrow_0)$ is called a* chord. *Every chord (s, t, s') defines two different walks from $\imath$ to s': $\imath[\pi_s t\rangle s'$ and $\imath[\pi_{s'}\rangle s'$, forming a* generalised cycle[2] *by going from $\imath$ to s' via π_s and t and then backwards through $\pi_{s'}$ until we reach $\imath$ again. With the T-vector $\mathbf{1}_t$ given by $\mathbf{1}_t(t) = 1$ and $\forall u \in T \setminus \{t\}\colon \mathbf{1}_t(u) = 0$, let*

$$\Gamma_{TS,TS_0} = \{\Psi(\pi_s) + \mathbf{1}_t - \Psi(\pi_{s'}) \mid (s, t, s') \text{ is a chord}\}.$$

The set Γ_{TS,TS_0} is called the cycle basis *(of TS, with regard to TS_0).*

The cycle bases for the examples in Fig. 1 are $\{\mathbf{1}_a + \mathbf{1}_b\}$, $\{2{\cdot}\mathbf{1}_a\}$, $\{2{\cdot}(\mathbf{1}_a{+}\mathbf{1}_b)\}$, and $\emptyset$ for the first four lts. They can be computed by the above formula from the chords $s_1[b\rangle\imath$ in TS_1, $s_1[a\rangle\imath$ in TS_2, and $s_3[b\rangle\imath$ in TS_3. TS_4 does not have chords and therefore yields an empty cycle basis. For TS_5 we have either the chord $\imath[a\rangle s_2$ or $s_1[a\rangle s_2$, depending on the chosen spanning tree. This leads to either $\Psi(\pi_\imath) + \mathbf{1}_a - \Psi(\pi_{s_2}) = \mathbf{0} + \mathbf{1}_a - (\mathbf{1}_a + \mathbf{1}_b) = -\mathbf{1}_b$ or $\Psi(\pi_{s_1}) + \mathbf{1}_a - \Psi(\pi_{s_2}) = \mathbf{1}_b + \mathbf{1}_a - \mathbf{1}_a = \mathbf{1}_b$ as the only vector in the cycle basis.

The term *cycle basis* is justified by the following lemma.

[2] A cycle in the underlying undirected graph, i.e. edges can also be followed backwards.

Lemma 1 (Cycle basis). *The Parikh vector $\Psi(\kappa)$ of any cycle $s[\kappa\rangle s$ in $TS = (S, T, \to, \imath)$ with a spanning tree $TS_0 = (S, T, \to_0, \imath)$ can be written as*

$$\Psi(\kappa) = \sum_{\gamma \in \Gamma_{TS,TS_0}} k_\gamma \cdot \gamma$$

for some coefficients $k_\gamma \in \mathbb{Z}$.

Proof. For any non-chord $s'[t\rangle s''$, $\Psi(\pi_{s'}) + \mathbf{1}_t - \Psi(\pi_{s''}) = \Psi(\pi_{s'}t) - \Psi(\pi_{s''}) = \mathbf{0}$ since $\pi_{s'}t = \pi_{s''}$. Now, with $\kappa = a_1 \dots a_m$ and $s = s_0[a_1\rangle s_1 \dots s_{m-1}[a_m\rangle s_m = s$, $\Psi(\kappa) = \sum_{i=0}^{m-1} \mathbf{1}_{a_{i+1}} = \sum_{i=0}^{m-1} (\Psi(\pi_{s_i}) + \mathbf{1}_{a_{i+1}} - \Psi(\pi_{s_{i+1}}))$, where each $\Psi(\pi_{s_i})$ appears twice, once with a negative sign, and $\pi_{s_0} = \pi_{s_n}$. All the elements of the sum are either zero or in Γ_{TS,TS_0}. □

In the remainder of this paper, we always assume that $TS = (S, T, \to, \imath)$ is a finite and totally reachable labelled transition system. Note that by total reachability, TS has at least one spanning tree TS_0 [2]. We pick one such spanning tree and let $\Gamma = \Gamma_{TS,TS_0}$ denote the cycle basis defined by it.

3 Regions

Regions, to be defined next, mimick the properties of Petri net places at the transition system level. Our nomenclature accords with this idea, even though we shall not define Petri nets in the present paper: B and F assign backward and forward weights to labels t, so that these weights can serve as connecting arcs between a transition t (which realises the label t) and a place of a Petri net, while R assigns a token count in each marking to that place.

Definition 5 (Regions of an lts, and the effect of a label). *Let $TS = (S, T, \to, \imath)$ be an lts. A triple $\rho = (R, B, F) \in \mathbb{N}^S \times \mathbb{N}^T \times \mathbb{N}^T$ is a* region *of TS if for all $s, s' \in S$ and $t \in T$, $s[t\rangle s'$ implies $R(s) \geq B(t)$ and $R(s') = R(s) - B(t) + F(t)$. The derived function $E\colon T \to \mathbb{Z}$ defined by $E(t) = F(t) - B(t)$ is called the* effect *of t.*

Definition 6 (State and event/state separation properties [1]). *An lts $TS = (S, T, \to, \imath)$ satisfies SSP (state separation property) iff*

$$\forall s, s' \in S\colon \quad s \neq s' \quad \Rightarrow \quad \exists \text{ region } \rho = (R, B, F) \text{ with } R(s) \neq R(s') \qquad (1)$$

meaning that if all regions agree on two states, then the latter are equal.
TS satisfies ESSP (event/state separation property) iff

$$\forall s \in S\, \forall t \in T\colon \quad (\neg s[t\rangle) \quad \Rightarrow \quad \exists \text{ region } \rho = (R, B, F) \text{ with } R(s) < B(t) \qquad (2)$$

meaning that if all regions satisfy $R(s) \geq B(t)$, then s enables t, i.e. $s[t\rangle$.

In usual parlance, two distinct (unordered) states $s, s' \in S$ yield a *state separation problem*, denoted by $\mathrm{SSP}(s, s')$, and if there is a region ρ as in (1), then ρ is said to *solve* $\mathrm{SSP}(s, s')$.[3] Intuitively, ρ distinguishes s and s'. A state s and a label t not enabled in this state ($\neg s[t\rangle$) yield an *event/state separation problem*, denoted by $\mathrm{ESSP}(s, t)$. A region ρ as in (2) is then said to *solve* $\mathrm{ESSP}(s, t)$. Intuitively, this region prevents t in state s.

For example, in Fig. 1, TS_1 satisfies both SSP and ESSP. Indeed, if we represent the three functions of a region $\rho = (R, B, F)$ by their equivalent multisets, $\rho_1 = (\{s_1\}, \{b\}, \{a\})$ and $\rho_2 = (\{\imath\}, \{a\}, \{b\})$ are two regions such that ρ_1 solves the event/state separation problem $\mathrm{ESSP}(\imath, b)$ and ρ_2 solves the event/state separation problem $\mathrm{ESSP}(s_1, a)$. Either of them solves the state separation problem $\mathrm{SSP}(\imath, s_1)$. TS_2 and TS_3 (in Fig. 1) satisfy ESSP but not SSP ($\mathrm{SSP}(\imath, s_1)$ in TS_2 as well as $\mathrm{SSP}(\imath, s_2)$ and $\mathrm{SSP}(s_1, s_3)$ in TS_3 are unsolvable). TS_4 satisfies SSP but not ESSP, here no region preventing a at s_2 exists. TS_5 satisfies neither SSP nor ESSP, because $\mathrm{SSP}(\imath, s_1)$ and $\mathrm{ESSP}(s_1, b)$ are unsolvable. The significance of these properties is expressed by the following result [1]. (Place/transition Petri nets are defined, e.g., in [8].)

Theorem 1 (Basic region theorems for place/transition nets). *A totally reachable and finite lts is isomorphic to the reachability graph of some place/transition Petri net if, and only if, it satisfies* SSP $\wedge$ ESSP*.*

If a totally reachable, finite lts satisfies ESSP*, then it is language-equivalent to the reachability graph of some Petri net.*

For example, for TS_3 as shown in Fig. 1, even though there is no Petri net with an isomorphic reachability graph, there is still a Petri net with a language-equivalent reachability graph, for instance a net whose reachability graph is isomorphic to TS_1. Interestingly, a totally reachable, finite lts may be language-equivalent to some Petri net without satisfying ESSP. This is the case, for instance, for system TS_5 shown in Fig. 1: it is easy to build a Petri net with two transitions $\{a, b\}$ and the language $\{\varepsilon, a, b, ba\} = L(TS_5)$, but since $\imath[a\rangle s_2$ and $s_1[a\rangle s_2$, any region will assign the same token count to s_1 and $\imath$, so that it is not possible to allow b at $\imath$ and to exclude it from s_1; as a consequence, neither ESSP nor SSP are satisfied by this system.

Here are some observations about regions and walks. Let a finite, totally reachable lts $TS = (S, T, \rightarrow, \imath)$ be given and let TS_0 be any spanning tree of TS with root $\imath$.

(a) Any T-vector $E\colon T \to \mathbb{Z}$ can be extended to words in T^* by defining

$$E(\sigma) = \sum_{t \in T} E(t) \cdot \Psi(\sigma)(t) = E^{\intercal} \cdot \Psi(\sigma), \qquad \text{for any } \sigma \in T^*$$

($^{\intercal}$ means 'transposed', $\cdot$ is scalar product)
If $\rho = (R, B, F)$ is a region with effect E, then for any walk $s_1[\tau\rangle s_2$, we obtain: $R(s_2) = R(s_1) + E(\tau)$. (Proof: by easy induction on the length of τ.)

[3] $\mathrm{SSP}(s, s')$ equals $\mathrm{SSP}(s', s)$, and thus, $\mathrm{SSP}(s, s')$ is solvable iff $\mathrm{SSP}(s', s)$ is solvable.

(b) In particular, by total reachability, for any state $s \in S$, we have $R(s) = R(\imath) + E(\pi_s)$. This implies that knowing $R(\imath)$ (and π_s) is sufficient for knowing $R(s)$, for every state s.
(c) Also, let $(s, t, s') \in \rightarrow$ be an edge of TS. Then $E(\pi_s t) - E(\pi_{s'}) = 0$: Both $\imath[\pi_s t\rangle s'$ and $\imath[\pi_{s'}\rangle s'$ are walks reaching s', so by (b) we have $R(s') = R(\imath) + E(\pi_s t) = R(\imath) + E(\pi_{s'})$, which can be re-arranged into the statement. In particular, for each entry $\gamma \in \Gamma$ of the cycle basis Γ, $E^\intercal \cdot \gamma = 0$ holds.

4 Pure Event/State Separability

For Petri net synthesis, for each separation problem a solving region has to be found. To characterise separation problems geometrically, we will use the following definition.

Definition 7 (Linear Hull and Convex Hull). *Given a set S of points (in some vector space for our purposes), its* linear hull*, or* linear span*, is the set of all linear combinations of elements of S, namely* $\operatorname{span}(S) = \{\sum_{i=1}^{\ell} k_i v_i \mid \ell \in \mathbb{N}, v_i \in S, k_i \in \mathbb{Q}\}$*. A set of points is convex if for each two points in it, also all points on a straight line connecting both are contained. The* convex hull *of S is the smallest convex set enclosing S. The convex hull only contains non-negative linear combinations (of points/vectors in S) with total weight one and is defined as* $\operatorname{convex}(S) = \{\sum_{i=1}^{\ell} k_i v_i \mid \ell \in \mathbb{N}, v_i \in S, k_i \in \mathbb{Q}, k_i \geq 0, \sum_{i=1}^{\ell} k_i = 1\}$.

Two examples for convex hulls are given in Fig. 2.

State separation was previously characterised in a way which we can reformulate as:

Proposition 1 (Characterisation of SSP [4]). *Let TS be a finite, totally reachable lts. A state separation problem SSP(s, s') cannot be solved if and only if $\Psi(\pi_s) - \Psi(\pi_{s'}) \in \operatorname{span} \Gamma$ (with Γ being our cycle basis of TS).*

In other words, a state separation problem is unsolvable exactly if the difference between these states' Parikh vectors corresponds to a cycle in the lts.

We now want to characterise event/state separation. The general case will be examined in the next section while this section concentrates on the *pure* case, to be defined next. Pure Petri nets are a well known class of nets where no side-conditions are allowed, meaning that there are no places and transitions with flows between them in both directions. The equivalent condition for a region is as follows:

Definition 8 (Pure Regions). *Let $TS = (S, T, \rightarrow, \imath)$ be an lts. A region $\rho = (R, B, F)$ of TS is* pure *if it satisfies $B(t) = 0 \vee F(t) = 0$ for each label $t \in T$.*

Clearly, a pure region (R, B, F) is fully described by R and E, so can equivalently be represented as the pair (R, E). This transformation is also possible for a non-pure region, in which case side-conditions are lost. Since SSP(s, s') only depends on R, the resulting region would solve SSP(s, s') if and only if the original region solves it. Thus, Proposition 1 is also valid when only considering pure regions.

However, for event/state separation, the general and pure case are different. In this section, we want to show the following result, where addition of sets is interpreted element-wise:

Theorem 2 (Characterisation of pure ESSP). *Let TS be a finite, totally reachable lts. An event/state separation problem ESSP(s, t) cannot be solved by any pure region if and only if*

$$\Psi(\pi_s) + \mathbf{1}_t \in \operatorname{span} \Gamma + \operatorname{convex} \Theta$$

where $\Theta = \{\Psi(\pi_{s'}) \mid s' \in S\}$ *and* Γ *is a cycle basis.*

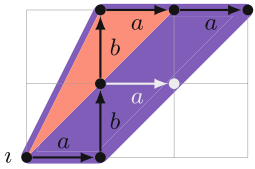


Fig. 2. The lts TS_4 from Fig. 1 drawn in a coordinate system. The triangular area is the convex hull of the Parikh vectors of states enabling a and the trapezoidal area shows the convex hull of the Parikh vectors of all states. To make it clear that the trapezoidal area contains the triangular area, the former one was slightly enlarged. An unsolvable event/state separation problem is visualised by the light a-edge.

As an example for this theorem consider Fig. 2. The lts has no cycles, which means that $\operatorname{span} \Gamma = \{\mathbf{0}\}$. Thus, only the convex hull of Parikh vectors of states can lead to unsolvable separation problems. In the figure, which visualises this hull as the trapezoid containing all arrows, the light state is not part of the lts, but inside the convex hull. By the above theorem, the ESSP instance corresponding to the light edge is unsolvable.

Note that the lts with the light edge and state is still not isomorphic to the reachability graph of any pure Petri net, because, again by the above theorem, the light state needs an outgoing edge with label b.

We will now first show that our condition is sufficient, and then that it is also necessary.

Lemma 2 (Sufficient condition for pure ESSP). *Let TS be a totally reachable lts. Given a state* $s \in S$ *and a label* $t \in T$ *with* $\neg s[t\rangle$ *so that* $\Psi(\pi_s) + \mathbf{1}_t \in \operatorname{span} \Gamma + \operatorname{convex} \Theta$*, the event/state separation problem ESSP(s, t) cannot be solved by any pure region.*

Proof. Assume an arbitrary pure region (R, B, F). Let $k_\gamma, k_s \in \mathbb{Q}$ be the factors showing that $\Psi(\pi_s) + \mathbf{1}_t$ is representable, i.e., $k_s \geq 0$ and $\sum_{s' \in S} k_s = 1$ so that:

$$\Psi(\pi_s) + \mathbf{1}_t = \sum_{\gamma \in \Gamma} k_\gamma \cdot \gamma + \sum_{s' \in S} k_s \cdot \Psi(\pi_{s'}) \tag{3}$$

We now have:

$$
\begin{aligned}
& R(s) + E(t) \\
&= R(\imath) + E^\intercal \cdot (\Psi(\pi_s) + \mathbf{1}_t) && \text{(by Item (b) above)} \\
&= R(\imath) + \sum_{\gamma \in \Gamma} k_\gamma \cdot E^\intercal \cdot \gamma + \sum_{s' \in S} k_s \cdot E^\intercal \cdot \Psi(\pi_{s'}) && \text{(by (3) and linearity)} \\
&= R(\imath) + 0 + \sum_{s' \in S} k_s \cdot E^\intercal \cdot \Psi(\pi_{s'}) && \text{(by Item (c) above)} \\
&= \sum_{s' \in S} k_s \cdot (R(\imath) + E^\intercal \cdot \Psi(\pi_{s'})) && \text{(since } \sum_{s' \in S} k_s = 1\text{)} \\
&= \sum_{s' \in S} k_s \cdot R(s') \geq 0 && \text{(by Item (b) and region)}
\end{aligned}
$$

To summarise, we have shown that for an arbitrary region we have $R(s)+E(t) \geq 0$. For this pure case, we know $B(t) = 0 \vee F(t) = 0$. If $B(t) = 0$, $R(s) \geq B(t)$ by definition of a region. If $F(t) = 0$, $R(s) \geq -E(t) = B(t)$. A region solving ESSP(s,t) would require $R(s) < B(t)$, though. □

In the following proof we will need the well known Farkas' Lemma.

Lemma 3 (Farkas' Lemma [7]). *Let A be a rational (n,m)-matrix and b a rational $(m,1)$-vector. Exactly one of the following alternatives holds. Either there is a rational $(m,1)$-vector x so that $A \cdot x \geq \mathbf{0}$ and $b^\intercal \cdot x < 0$, or there exists a rational $(n,1)$-vector y so that $A^\intercal \cdot y = b$ and $y \geq \mathbf{0}$.*

Lemma 4 (Necessary condition for pure ESSP). *Let TS be a finite, totally reachable lts. Given an event-state separation problem ESSP(s,t) unsolvable by any pure region, we have $\Psi(\pi_s) + \mathbf{1}_t \in \operatorname{span} \Gamma + \operatorname{convex} \Theta$.*

Proof. A pure region is characterised by

$$\forall \gamma \in \Gamma \colon E^\intercal \cdot \gamma = 0 \qquad \forall s' \in S \colon R(\imath) + E^\intercal \cdot \Psi(\pi_{s'}) \geq 0$$

Note that with $R(s') = R(\imath) + E^\intercal \cdot \Psi(\pi_{s'})$ and $E = F - B$, we can uniquely construct a triple $(R,B,F) \in \mathbb{N}^S \times \mathbb{N}^T \times \mathbb{N}^T$ with $F(t') = 0 \vee B(t') = 0$ for all $t' \in T$. Assume $s'[t'\rangle s''$. If this edge is in the spanning tree, we get $R(s'') - R(s') = R(\imath) + E^\intercal \cdot \Psi(\pi_{s''}) - R(\imath) - E^\intercal \cdot \Psi(\pi_{s'}) = E^\intercal \cdot (\Psi(\pi_{s''}) - \Psi(\pi_{s'})) = E^\intercal \cdot (\Psi(\pi_{s'}t') - \Psi(\pi_{s'})) = E(t') = F(t') - B(t')$ as expected. If the edge is a chord, we find $R(s'') - R(s') = E^\intercal \cdot (\Psi(\pi_{s''}) - \Psi(\pi_{s'})) = E^\intercal \cdot (\Psi(\pi_{s''}) - \mathbf{1}_{t'} - \Psi(\pi_{s'}) + \mathbf{1}_{t'}) = E^\intercal \cdot \mathbf{1}_{t'} = E(t')$ again, by $\Psi(\pi_{s'}) + \mathbf{1}_{t'} - \Psi(\pi_{s''}) \in \Gamma$ and Item (c). In both cases $R(s') - B(t') = R(s'') - F(t')$, so $R(s') \geq B(t')$ holds as $B(t') = 0 \vee F(t') = 0$. Overall, (R,B,F) is a pure region.

The equality in the first condition (for the cycles γ) can also be expressed as two inequalities. Written via matrix multiplication, this inequality system

looks as follows, where $T = \{t_1, t_2, \ldots, t_n\}$ and multiple rows are abbreviated via quantification:

$$\begin{pmatrix} \forall \gamma \in \Gamma\colon 0 & (\gamma)^\intercal \\ \forall \gamma \in \Gamma\colon 0 & (-\gamma)^\intercal \\ \forall s' \in S\colon 1 & (\Psi(\pi_{s'}))^\intercal \end{pmatrix} \begin{pmatrix} R(\iota) \\ E(t_1) \\ \vdots \\ E(t_n) \end{pmatrix} \geq \mathbf{0}$$

So each row of the matrix on the left (which we will name A from now) consists of a constant (0 or 1) followed by the row vector for one of the cycles (γ, also in the negated form) or the row form of a Parikh vector $\Psi(\pi_{s'})$ for a walk in the spanning tree. The solution vector with the initial value and the effects of a region will be called x, so that the above can be written as $Ax \geq \mathbf{0}$.

Note that this system has a solution in the natural numbers whenever it has a rational solution. If $Ax \geq \mathbf{0}$, then for $y \in \mathbb{Q}_{>0}$ also $Ayx = y \cdot (Ax) \geq \mathbf{0}$, showing that if x is a solution, yx also is. Thus, without loss of generality, we can assume x to be rational.

Now that we have a linear-algebraic definition of a pure region, let us turn back to the event/state separation problem. We are assuming that $\mathrm{ESSP}(s, t)$ is unsolvable, which means that there is no pure region with $R(s) < B(t)$, which is, for pure regions, equivalent to $R(s) + E(t) < 0$ and can, by Item (b) above, be written as $b^\intercal x < 0$ with $b^\intercal = \big(1\ (\Psi(\pi_s) + \mathbf{1}_t)^\intercal\big)$. Again, if a solution is multiplied with a positive rational number, this inequality is still fulfilled.

We can now use Farkas' lemma (Lemma 3): Since the system $Ax \geq \mathbf{0}$ with $b^\intercal x < 0$ has no solution, this means there is a value $y \geq \mathbf{0}$ so that $b = A^\intercal y$, specifically:

$$\begin{pmatrix} 1 \\ (\Psi(\pi_s) + \mathbf{1}_t) \end{pmatrix} = \begin{pmatrix} \forall \gamma \in \Gamma\colon & \forall \gamma \in \Gamma\colon & \forall s' \in S\colon \\ 0 & 0 & 1 \\ (\gamma) & (-\gamma) & (\Psi(\pi_{s'})) \end{pmatrix} \cdot y$$

We index y by $y_{s'}$, y_γ, and $y_{-\gamma}$ for $s' \in S$ and $\gamma \in \Gamma$, and write the above as:

$$\begin{aligned} 1 &= \sum_{s' \in S} y_{s'} \\ \Psi(\pi_s) + \mathbf{1}_t &= \sum_{\gamma \in \Gamma} (y_\gamma - y_{-\gamma}) \cdot \gamma + \sum_{s' \in S} y_{s'} \cdot \Psi(\pi_{s'}) \end{aligned}$$

Defining $z_\gamma = y_\gamma - y_{-\gamma}$, the second equation can be written as $\Psi(\pi_s) + \mathbf{1}_t = \sum_{\gamma \in \Gamma} z_\gamma \cdot \gamma + \sum_{s' \in S} y_{s'} \cdot \Psi(\pi_{s'})$. We now see that this describes an element of the linear hull of Γ plus an element of the convex hull of $\Theta = \{\Psi(\pi_{s'}) \mid s' \in S\}$. Thus, our inequality system is equivalent to:

$$\Psi(\pi_s) + \mathbf{1}_t \in \operatorname{span} \Gamma + \operatorname{convex} \Theta \qquad \square$$

5 General Event/State Separability

The previous section provided a characterisation of event/state separability for pure Petri nets. In this section, the more complicated general case will be investigated. The aim is to show the following theorem:

Theorem 3 (Characterisation of general ESSP). *Let TS be a finite, totally reachable lts. An event/state separation problem ESSP(s, t) cannot be solved by any region if and only if*

$$\Psi(\pi_s) \in \operatorname{span} \Gamma + \operatorname{convex} \Theta_t,$$

where $\Theta_t = \{\Psi(\pi_{s'}) \mid s' \in S, s'[t\rangle\}$ *(which is* $\subseteq \Theta$*) and* Γ *is a cycle basis.*

Consider the example in Fig. 2. The triangular area shows the convex hull of the Parikh vectors of states enabling a. The state from which the light a-edge emanates is inside this area. By the above theorem, the missing light edge corresponds to an unsolvable event/state separation problem.

When adding the light edge to the lts, the resulting lts is isomorphic to the reachability graph of a Petri net. While in the pure case the light state needed an outgoing b-edge, in the general setting the corresponding event/state separation problem is solvable since the convex hull of states enabling b is just a vertical line segment that does not include the light state.

As in the previous section, each direction of this theorem will be shown separately.

Lemma 5 (Sufficient condition for general ESSP). *Let TS be a totally reachable lts. Given a state* $s \in S$ *and a label* $t \in T$ *with* $\neg s[t\rangle$ *so that* $\Psi(\pi_s) \in \operatorname{span} \Gamma + \operatorname{convex} \Theta_t$*, the event/state separation problem ESSP(s, t) is unsolvable.*

Proof. Let $S_t = \{s \in S \mid s[t\rangle\}$ be the set of all states enabling t and let $k_\gamma, k_s \in \mathbb{Q}$ be the factors showing that $\Psi(\pi_s)$ is representable, i.e., $k_s \geq 0$ and $\sum_{s' \in S_t} k_s = 1$ so that:

$$\Psi(\pi_s) = \sum_{\gamma \in \Gamma} k_\gamma \cdot \gamma + \sum_{s' \in S_t} k_s \cdot \Psi(\pi_{s'}) \tag{4}$$

Assume an arbitrary region (R, B, F). We now have the following:

$$\begin{aligned}
& R(s) \\
&= R(\imath) + E^\intercal \cdot \Psi(\pi_s) && \text{(by Item (b) above)} \\
&= R(\imath) + \sum_{\gamma \in \Gamma} k_\gamma \cdot E^\intercal \cdot \gamma + \sum_{s' \in S_t} k_{s'} \cdot E^\intercal \cdot \Psi(\pi_{s'}) && \text{(by (4) and linearity)} \\
&= R(\imath) + 0 + \sum_{s' \in S_t} k_{s'} \cdot E^\intercal \cdot \Psi(\pi_{s'}) && \text{(by Item (c) above)}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{s' \in S_t} k_{s'} \cdot (R(\imath) + E^\intercal \cdot \Psi(\pi_{s'})) && (\text{since } \sum_{s' \in S_t} k_{s'} = 1)\\
&= \sum_{s' \in S_t} k_{s'} \cdot R(s') && (\text{by Item (b) above})\\
&\geq \sum_{s' \in S_t} k_{s'} \cdot B(t) && (\text{since } s'[t\rangle \text{ and region})\\
&= B(t) && (\text{since } \sum_{s' \in S_t} k_{s'} = 1)
\end{aligned}
$$

This shows that the region (R, B, F) does not solve $\mathrm{ESSP}(s, t)$. Because the region was arbitrary, no region solves $\mathrm{ESSP}(s, t)$ and it is unsolvable. □

Lemma 6 (Necessary condition for general ESSP). *Let TS be a finite and totally reachable lts. Given an unsolvable event-state separation problem ESSP(s, t), we have $\Psi(\pi_s) \in \operatorname{span} \Gamma + \operatorname{convex} \Theta_t$.*

Proof. The definition of a region (R, B, F) is expressed by the following inequalities:

$$
\begin{aligned}
& & R(\imath) \geq 0 \qquad B \geq \mathbf{0} \qquad F \geq \mathbf{0}\\
\forall \gamma \in \Gamma\colon & & (F - B)^\intercal \cdot \gamma = 0\\
\forall s'[t'\rangle s'' \in \to\colon & & R(\imath) + (F - B)^\intercal \cdot \Psi(\pi_{s'}) - B^\intercal \cdot \mathbf{1}_{t'} \geq 0
\end{aligned}
$$

With the same reasoning as in the proof of Lemma 4 we find that for $s'[t'\rangle s''$, be it an edge of the spanning tree or a chord, holds $R(s'') - R(s') = F(t') - B(t')$. The last line of inequations then ensures that, if $s'[t\rangle s''$, then $R(s') = R(\imath) + (F - B)^\intercal \cdot \Psi(\pi_{s'}) \geq B^\intercal \cdot \mathbf{1}_{t'} = B(t')$, which makes (R, B, F) a region.

Replacing the equality in the middle line with two inequalities, the system can be written as a matrix multiplication $Ax \geq \mathbf{0}$ in the following way (where U represents the unit matrix which expresses the first three inequalities):

$$
\begin{pmatrix}
 & & U & & \\
\forall \gamma \in \Gamma\colon & 0 & (\gamma)^\intercal & (-\gamma)^\intercal & \\
\forall \gamma \in \Gamma\colon & 0 & (-\gamma)^\intercal & (\gamma)^\intercal & \\
\forall s'[t'\rangle s'' \in \to\colon & 1 & (\Psi(\pi_{s'}))^\intercal & (-\Psi(\pi_{s'}) - \mathbf{1}_{t'})^\intercal &
\end{pmatrix}
\begin{pmatrix}
R(\imath)\\ F(t_1)\\ \vdots\\ F(t_n)\\ B(t_1)\\ \vdots\\ B(t_n)
\end{pmatrix}
\geq \mathbf{0}
$$

As before, the system has a solution in the natural numbers whenever it has a rational solution.

Let us turn back to the event/state separation problem. We are assuming that $\mathrm{ESSP}(s, t)$ is unsolvable, which means that there is no region with $R(s) < B(t)$, which can be written by Item (b) above as $R(\imath) + E(\pi_s) < B(t)$ and thus results in $b^\intercal \cdot x < 0$ with $b^\intercal = \big(1\ (\Psi(\pi_s))^\intercal\ (-\Psi(\pi_s) - \mathbf{1}_t)^\intercal\big)$.

We can now use Farkas' lemma (Lemma 3): Since the system $Ax \geq \mathbf{0}$ with $b^\intercal x < 0$ has no solution, this means there is a value $y \geq \mathbf{0}$ so that $b = A^\intercal y$. The unit matrix that is part of $A^\intercal$ will be interpreted as introducing slack variables: Instead of $b = A^\intercal y$ with $A^\intercal = (UC^\intercal)$ we just aim for $b \geq C^\intercal y$. We obtain

$$\begin{pmatrix} 1 \\ (\Psi(\pi_s)) \\ (-\Psi(\pi_s) - \mathbf{1}_t) \end{pmatrix} \geq \begin{pmatrix} \forall \gamma \in \Gamma \colon \forall \gamma \in \Gamma \colon & \forall s'[t'\rangle s'' \in \to \colon \\ 0 \quad 0 & 1 \\ (\gamma) \quad (-\gamma) & (\Psi(\pi_{s'})) \\ (-\gamma) \quad (\gamma) & (-\Psi(\pi_{s'}) - \mathbf{1}_{t'}) \end{pmatrix} \cdot y.$$

Thus, by Farkas' Lemma we have a value $y \geq \mathbf{0}$, which we will index by $y_{s'[t'\rangle s''}$, y_γ, and $y_{-\gamma}$ for $s'[t'\rangle s'' \in \to$ and $\gamma \in \Gamma$, so that:

$$\begin{aligned} 1 &\geq \sum_{s'[t'\rangle s'' \in \to} y_{s'[t'\rangle s''} \\ \Psi(\pi_s) &\geq \sum_{\gamma \in \Gamma} (y_\gamma - y_{-\gamma}) \cdot \gamma + \sum_{s'[t'\rangle s'' \in \to} y_{s'[t'\rangle s''} \cdot \Psi(\pi_{s'}) \\ -\Psi(\pi_s) - \mathbf{1}_t &\geq \sum_{\gamma \in \Gamma} (y_\gamma - y_{-\gamma}) \cdot (-\gamma) + \sum_{s'[t'\rangle s'' \in \to} y_{s'[t'\rangle s''} \cdot (-\Psi(\pi_{s'}) - \mathbf{1}_{t'}) \end{aligned}$$

We begin by defining $z_\gamma = y_\gamma - y_{-\gamma}$, which thus is no longer necessarily non-negative. Next, we negate the last inequality, bring $\mathbf{1}_t$ to the other side, and distribute the second sum in it to arrive at:

$$\Psi(\pi_s) \leq \sum_{\gamma \in \Gamma} z_\gamma \cdot \gamma + \sum_{s'[t'\rangle s'' \in \to} y_{s'[t'\rangle s''} \cdot \Psi(\pi_{s'}) + \sum_{s'[t'\rangle s'' \in \to} y_{s'[t'\rangle s''} \cdot \mathbf{1}_{t'} - \mathbf{1}_t$$

The second inequality gives a lower bound to $\Psi(\pi_s)$ while this new, last inequality provides an upper bound. Relating these two inequalities, dropping $\Psi(\pi_s)$, and eliminating common summands results in $\mathbf{1}_t \leq \sum y_{s'[t'\rangle s''} \cdot \mathbf{1}_{t'}$. We can consider only component t of this vector to get $1 \leq \sum y_{s'[t'\rangle s''} \cdot \mathbf{1}_{t'}(t) \leq \sum y_{s'[t'\rangle s''} \leq 1$, where the second inequality is due to $\mathbf{1}_{t'}(t) \leq 1$ and the last one is part of our inequality system. Thus $1 = \sum y_{s'[t'\rangle s''} \cdot \mathbf{1}_{t'}(t)$, i.e. already the variables $y_{s'[t'\rangle s''}$ with $t' = t$ sum up to one, and so with $\sum y_{s'[t'\rangle s''} \leq 1$ we can conclude now that $y_{s'[t'\rangle s''} = 0$ for $t' \neq t$ and also $1 = \sum y_{s'[t'\rangle s''}$, which is stronger than the original inequality and can replace it. To eliminate all the $y_{s'[t'\rangle s''}$ that are zero, define $S_t = \{s' \in S \mid s'[t\rangle\}$ to be the set of all states enabling t and $y_{s'} = \sum y_{s'[t\rangle s''}$ to be the weight of all edges emanating from state s' with label t. This allows us to simplify our inequality system as follows:

$$\begin{aligned} 1 &= \sum_{s' \in S_t} y_{s'} \\ \Psi(\pi_s) &\geq \sum_{\gamma \in \Gamma} z_\gamma \cdot \gamma + \sum_{s' \in S_t} y_{s'} \cdot \Psi(\pi_{s'}) \\ \Psi(\pi_s) &\leq \sum_{\gamma \in \Gamma} z_\gamma \cdot \gamma + \sum_{s' \in S_t} y_{s'} \cdot \Psi(\pi_{s'}) + \sum_{s' \in S_t} y_{s'} \cdot \mathbf{1}_t - \mathbf{1}_t \end{aligned}$$

By the equality that we just computed, $\sum_{s' \in S_t} y_{s'} \cdot \mathbf{1}_t = \mathbf{1}_t$. Now, after simplification, the last two inequalities are the same except for opposite comparison operators. Thus, we can combine them into an equality to arrive at the following system with $y_{s'}, z_\gamma \in \mathbb{Q}$ and $y_{s'} \geq 0$.

$$\Psi(\pi_s) = \sum_{\gamma \in \Gamma} z_\gamma \cdot \gamma + \sum_{s' \in S_t} y_{s'} \cdot \Psi(\pi_{s'}) \qquad 1 = \sum_{s' \in S_t} y_{s'}$$

Finally, we see that $\Psi(\pi_s)$ is the sum of an element of the linear hull of Γ and the convex hull of $\Theta_t = \{\Psi(\pi_{s'}) \mid s' \in S_t\}$. Thus, we can write it as such:

$$\Psi(\pi_s) \in \operatorname{span} \Gamma + \operatorname{convex} \Theta_t$$ □

To better illustrate the close connection to the pure case, Theorem 3 can be reformulated. The set Θ_t of Parikh vectors of states which enable t can be replaced with the set Θ^t containing Parikh vectors of states that are reached by t:

Theorem 4. *Let TS be a finite, totally reachable lts. An event/state separation problem ESSP(s, t) cannot be solved by any region if and only if*

$$\Psi(\pi_s) + \mathbf{1}_t \in \operatorname{span} \Gamma + \operatorname{convex} \Theta^t,$$

where $\Theta^t = \{\Psi(\pi_{s''}) \mid s', s'' \in S, s'[t\rangle s''\}$ $(\subseteq \Theta)$ *and* Γ *is a cycle basis.*

Proof. We have to show that $v \in \operatorname{span} \Gamma + \operatorname{convex} \Theta_t \Leftrightarrow v + \mathbf{1}_t \in \operatorname{span} \Gamma + \operatorname{convex} \Theta^t$. The rest follows from Theorem 3.

($\Rightarrow$) Let $v \in \operatorname{span} \Gamma + \operatorname{convex} \Theta_t$ and $S_t = \{s' \in S \mid s'[t\rangle\}$. By definition, there are $y_\gamma, y_{s'} \in \mathbb{Q}$ for $\gamma \in \Gamma$ and $s' \in S_t$ with $y_{s'} \geq 0$ for all $s' \in S_t$, and $\sum_{s' \in S_t} y_{s'} = 1$ so that

$$v = \sum_{\gamma \in \Gamma} y_\gamma \cdot \gamma + \sum_{s' \in S_t} y_{s'} \cdot \Psi(\pi_{s'})$$

Let now $S^t = \{s'' \in S \mid s' \in S, s'[t\rangle s''\}$. Choose some mapping ϱ: $S_t \to S^t$ such that $\varrho(s') = s''$ implies $s'[t\rangle s''$.[4] If (s', t, s'') is a chord, then $\Psi(\pi_{s'}) + \mathbf{1}_t - \Psi(\pi_{s''})$ is in Γ, otherwise this expression is $\mathbf{0}$ since $\pi_{s''} = \pi_{s'} t$. Thus, $\Psi(\pi_{s'}) + \mathbf{1}_t = \Psi(\pi_{s'}) + \mathbf{1}_t - \Psi(\pi_{s''}) + \Psi(\pi_{s''}) \in (\Gamma \cup \{\mathbf{0}\}) + \Theta^t$. With this and $\sum y_{s'} = 1$, we can now write

$$\begin{aligned} v + \mathbf{1}_t &= \sum_{\gamma \in \Gamma} y_\gamma \cdot \gamma + \sum_{s' \in S_t} y_{s'} \cdot (\Psi(\pi_{s'}) + \mathbf{1}_t) \\ &= \sum_{\gamma \in \Gamma} (y_\gamma + z_\gamma) \cdot \gamma + \sum_{s'' \in S^t} y_{s''} \cdot \Psi(\pi_{s''}) \in \operatorname{span} \Gamma + \operatorname{convex} \Theta^t \end{aligned}$$

where $y_{s''} = \sum_{s' \in \varrho^{-1}(s'')} y_{s'}$ and z_γ counts how often each individual chord was added.

[4] ϱ is unique if TS is deterministic.

($\Leftarrow$) This case is analogous to the previous one. For $v + \mathbf{1}_t \in \operatorname{span} \Gamma + \operatorname{convex} \Theta^t$, we have $v = \sum_{\gamma \in \Gamma} y_\gamma \cdot \gamma + \sum_{s'' \in S^t} y_{s''} \cdot (\Psi(\pi_{s''}) - \mathbf{1}_t)$. By adding suitable $\Psi(\pi_{s'}) + \mathbf{1}_t - \Psi(\pi_{s''})$ and compensating for this in the first sum, we see that $v \in \operatorname{span} \Gamma + \operatorname{convex} \Theta_t$. □

6 Further Examples

The unsolvable event/state separation problem from the example in Fig. 2 was already discussed after Theorems 2 and 3. This section provides some further examples.

In TS_5 from Fig. 1, ESSP(s_1, b) is unsolvable. The cycle basis contains a Parikh vector with a single b, $\Gamma = \{\mathbf{1}_b\}$, since the two a's cancel each other out. Now, the Parikh vector of s_1, which is reached by a single b, i.e. $\Psi(\pi_{s_1}) = \mathbf{1}_b$, can be represented as the only element of the cycle basis plus the Parikh vector of the initial state $\imath$, which is the zero vector $\mathbf{0}$. Since $\Psi(\pi_{s_1}) \in \operatorname{span} \Gamma + \operatorname{convex} \Theta_b$, ESSP$(s_1, b)$ is unsolvable by Theorem 3.

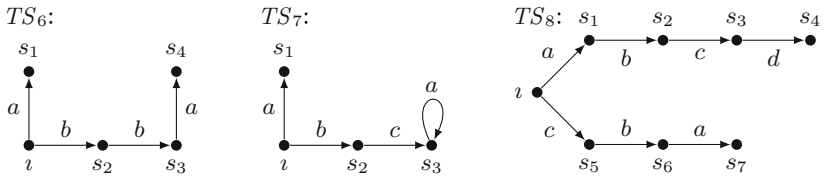


Fig. 3. Three transition systems with unsolvable ESSP instances.

Some more examples are shown in Fig. 3. TS_6 has a comb structure with a shaft of b's and a's as teeth, where at least one tooth is missing (here at s_2). This makes ESSP(s_2, a) unsolvable. Quite obviously, $\Psi(\pi_{s_2}) = \mathbf{1}_b = \frac{1}{2}\Psi(\pi_\imath) + \frac{1}{2}\Psi(\pi_{s_3}) \in \operatorname{convex}\{\Psi(\pi_\imath), \Psi(\pi_{s_3})\} = \operatorname{convex} \Theta_a$ and $\operatorname{span} \Gamma = \{\mathbf{0}\}$ (a linear hull always contains $\mathbf{0}$). Comb structures are special cases of unsolvable ESSP, but slight changes like replacing the edge $s_2[b\rangle s_3$ by $s_2[c\rangle s_3$, i.e. just changing one label, will make the whole system solvable, even to a pure Petri net (with a region $R(\imath) = 3$, $E(a) = -2$, $E(b) = -3$, and $E(c) = 2$ preventing a at s_2).

A very similar lts is shown in TS_7. Despite the different labels on the shaft, it has three unsolvable separation problems: SSP$(\imath, s_1)$, ESSP(s_1, a), and ESSP(s_1, b). All of these are unsolvable because there is a chord with a Parikh vector that contains just label a once (namely the loop at s_3). Thus, state s_1, which is reached from $\imath$ via $\pi_{s_1} = a$, has a Parikh vector that appears in the cycle basis and is the sum of the Parikh vector of the initial state and an entry from the cycle basis. By Proposition 1, SSP$(\imath, s_1)$ is unsolvable. Since $\imath$ enables labels a and b, this also means that these labels cannot be prevented in s_1.

When considering pure regions, ESSP(s_2, a) becomes unsolvable, too: Since the cycle basis contains a Parikh vector with just event a (for the loop), $\Psi(\pi_s) +$

$\mathbf{1}_a \in \operatorname{span} \Gamma + \operatorname{convex} \Theta$ for any state s via $\Psi(\pi_s) \in \operatorname{convex} \Theta$ and $\mathbf{1}_a \in \operatorname{span} \Gamma$. Thus, a must be enabled in every state.

As the last example, TS_8 has two states, s_3 and s_7, which both have the same Parikh vector consisting of one a, one b and one c. Thus, $\Psi(\pi_{s_3}) - \Psi(\pi_{s_7}) = \mathbf{0} \in \{\mathbf{0}\} = \operatorname{span} \emptyset = \operatorname{span} \Gamma$ and $\mathrm{SSP}(s_3, s_7)$ is unsolvable. Furthermore, since s_3 enables d, but s_7 does not, $\mathrm{ESSP}(s_7, d)$ is unsolvable. This can be seen from $\Psi(\pi_{s_7}) = \Psi(\pi_{s_3}) \in \operatorname{convex} \Theta_d$. If the attention is restricted to pure Petri nets for this lts, then $\mathrm{ESSP}(s_1, c)$ is also unsolvable. Note that this does not represent an unsolvable comb structure. As seen before, the shaft with different labels $\iota[a\rangle s_1[b\rangle s_2$ does not necessarily lead to an unsolvable ESSP.

7 Restrictions and Extensions

In this section, the restriction to lts over a two-letter alphabet and corresponding to words is investigated. In this setting, Theorem 3 can be used as an alternative proof for an already known result.

We also consider the extension of our characterisation to infinite lts, but we will obtain only a partial result, i.e. we do not get a characterisation but only a sufficient condition.

7.1 Two-Letter Words

We start this section by deriving some already known results as corollaries to our characterisation to underline the generality of our new result.

In [5], conditions for Petri net solvability of words over a two letter alphabet are investigated. A word is identified with a transition system in the obvious way. For example, TS_4 from Fig. 1 corresponds to the word *abbaa*. The main result for words is the following theorem.

Theorem 5. (Theorem 2 of [5]). *A word $w \in \{a, b\}^*$ is solvable if and only if the following formula holds for $x = a \wedge y = b$ as well as for $x = b \wedge y = a$:*

$$\forall \alpha, \beta, \gamma, \delta \colon (w = \alpha y \beta x \gamma y \delta \Rightarrow \#_y(y\beta) \cdot \#_x(x\gamma) > \#_x(y\beta) \cdot \#_y(x\gamma))$$

Proof (Idea). We decompose $\gamma = x^n y \gamma'$ into a prefix x^n for some $n \in \mathbb{N}$ and the remaining part $y\gamma'$. (If γ does not contain y, the inequation above is trivially true.) Let s, t, u, v be states so that $s[y\beta\rangle t[xx^n\rangle u[y\gamma'\rangle v[y\rangle$. This situation is illustrated in Fig. 4. We can rearrange the formula from the theorem into $\#_y(y\beta)/\#_x(y\beta) > \#_y(x\gamma)/\#_x(x\gamma)$. This formula compares the slope of the line connecting states s and t with the slope of the line connecting t and v, requiring the first line to have a higher slope than the second. We can see in the picture that this corresponds to state t being outside of the convex hull of s, u and v. Because these are the extremal states where y is enabled, by Theorem 3 this is equivalent to $\mathrm{ESSP}(t, y)$ being solvable. □

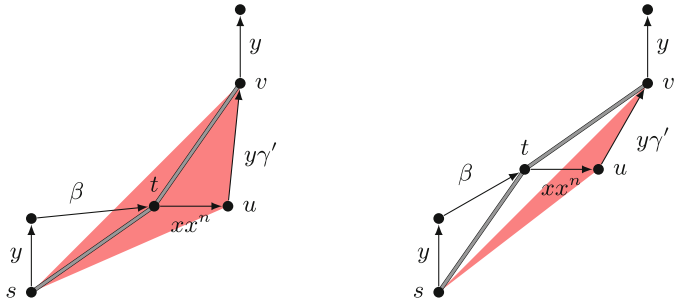


Fig. 4. Illustration for the proof of Theorem 5. Label x goes to the right while y goes up. On the left side, the slope of the line connecting s and t is lower than the slope of the line connecting t and v, putting t into the convex hull and yielding an unsolvable ESSP(t, y), in contrast to the right picture. For simplicity, we do not show any y's in β; this would complicate the convex hull, but the result still holds

Another related paper is [6] which investigates the possible shapes of reachability graphs of Petri nets with just two transitions. Its Theorem 2 states that an lts without non-trivial cycles[5] can be solved by a pure Petri net if its states form a convex set and all interior edges are present. In the present paper, Proposition 1 and Theorem 2 show the same statement, but also for the case of more than just two transitions: Because only trivial cycles exist, Proposition 1 guarantees that all SSP instances are solvable, and convexity and the presence of interior edges guarantee by Theorem 2 that all ESSP instances are solvable.

Theorem 3 in [6] handles the general case and is again a special case of Theorem 3, similar to the pure case.

7.2 Infinite Labelled Transition Systems

Our characterisations in Theorems 2 and 3 are only valid for finite lts. However, the sufficient conditions in Lemmas 2 and 5 do not have this precondition and also hold for infinite lts.

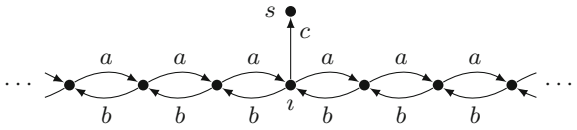


Fig. 5. An infinite lts where all separation problems not involving the state s are unsolvable.

The example in Fig. 5 shows that the theorems are not easily generalisable to infinite lts. The lts consists of an infinite chain of states going in two directions [3].

[5] Meaning that span $\Gamma = \{\mathbf{0}\}$.

The label a proceeds to the right while b advances to the left. Additionally, there is an extra state s reachable from the initial state via label c. This label occurs only once.

Let $\rho = (R, B, F)$ be an arbitrary region of this lts. The initial state has $R(\imath)$ tokens and there are walks $s'[a^{R(\imath)+1}\rangle\imath$ and $\imath[a^{R(\imath)+1}\rangle s''$ with suitable states s' and s''. Thus, $E(a) = 0$ since otherwise either $R(s')$ or $R(s'')$ would be negative, which is not allowed. By the same reasoning, $E(b) = 0$.

This shows that all state separation problems that do not involve the state s are unsolvable, because all such states are assigned the same region value, and also that all event/state separation problems not involving state s are unsolvable, since such a separation problem necessarily involves event c, which is enabled in a state with the same number of tokens.

However, there are no states s' and s'' in this lts so that $\Psi(\pi_{s'}) - \Psi(\pi_{s''}) \in \operatorname{span} \Gamma$, i.e. Proposition 1 does not hold for infinite lts. Each chord has a Parikh vector with one a and one b, i.e. $\Gamma = \{\mathbf{1}_a + \mathbf{1}_b\}$. Each state s'' to the right of $\imath$ has $\pi_{s''} = a^\ell$ for a suitable ℓ, while each state s' to the left of $\imath$ has $\pi_{s'} = b^\ell$. Thus, the only way to satisfy $\Psi(\pi_{s'}) - \Psi(\pi_{s''}) \in \operatorname{span} \Gamma$ is with $s' = s''$.

For event/state separation, we can see that all instances involving state s are solvable. ESSP instances not involving state s only exist with label c, so we only have to consider this label. Since $\Theta_c = \{\mathbf{0}\}$, Theorem 3 would simplify to $\Psi(\pi_{s'}) \in \operatorname{span} \Gamma = \{\ell \cdot (\mathbf{1}_a + \mathbf{1}_b) \mid \ell \in \mathbb{Q}\}$. No such state $s' \neq \imath$ exists even though there are many unsolvable event/state separation problems.

To summarise, our characterisation is a sufficient condition even for infinite lts, but the example shows that it is not a necessary condition in this case.

8 Conclusion

The main results of this paper are the following characterisations of event/state separability, as defined in Petri net synthesis [1]:

- For pure Petri nets, the convex hull of all states describes reachable states (Theorem 2).
- For general Petri nets, the convex hull of states enabling some event t contains states that must enable t (Theorem 3).

Together with a previous characterisation of state separation (Proposition 1), this amounts to a full geometric characterisation of lts that are reachability graphs of Petri nets.

We have seen that this characterisation can be used to simplify the proof of some earlier results, and also that it is only valid for finite lts and does not generalise to infinite transition systems.

While this characterisation can be used to produce better diagnostics in case a given lts cannot be synthesised into a Petri net, actually doing so is still an open problem. It is not obvious how our characterisation can be checked and it only allows to conclude that a Petri net solution exists, but cannot actually produce it.

Instead, we want to use the presented characterisation in future work for the idea of label-splitting: Given an event/state separation problem ESSP(s, t), our characterisation allows to pinpoint states that enable transition t and cause ESSP(s, t) to be unsolvable. One of the states $s' \in S_t$ with $y_{s'} > 0$ is chosen and $s'[t\rangle$ replaced with $s'[t'\rangle$, which means that the label t is split up into the labels t and t'. This is repeated until ESSP(s, t) becomes solvable. In the synthesised Petri net, both transitions, t and t', will later obtain the same label t, yielding a hopefully small, labelled Petri net with the sought behaviour.

References

1. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Texts in Theoretical Computer Science. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47967-4
2. Berge, C.: Graphs and Hypergraphs, vol. 6. Elsevier, New York (1973). North-Holland mathematical library
3. Best, E., Devillers, R.: Characterisation of the state spaces of marked graph Petri nets. Inf. Comput. **253**, 399–410 (2017). https://doi.org/10.1016/j.ic.2016.06.006
4. Best, E., Devillers, R., Schlachter, U.: A graph-theoretical characterisation of state separation. In: Steffen, B., Baier, C., van den Brand, M., Eder, J., Hinchey, M., Margaria, T. (eds.) SOFSEM 2017. LNCS, vol. 10139, pp. 163–175. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51963-0_13
5. Best, E., Erofeev, E., Schlachter, U., Wimmel, H.: Characterising Petri net solvable binary words. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016. LNCS, vol. 9698, pp. 39–58. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39086-4_4
6. Erofeev, E., Wimmel, H.: Reachability graphs of two-transition Petri nets. In: van der Aalst, W.M.P., Bergenthum, R., Carmona, J. (eds.) ATAED 2017. CEUR Workshop Proceedings, vol. 1847, pp. 39–54. CEUR-WS.org (2017). http://ceur-ws.org/Vol-1847/paper03.pdf
7. Farkas, J.: Theorie der einfachen Ungleichungen. J. für die reine und Angew. Math. **124**, 1–27 (1902). https://doi.org/10.1515/crll.1902.124.1
8. Reisig, W.: Petri Nets: An Introduction. EATCS Monographs in Theoretical Computer Science, vol. 4. Springer, Heidelberg (1985). https://doi.org/10.1007/978-3-642-69968-9

From Event-Oriented Models to Transition Systems

Eike Best[1], Nataliya Gribovskaya[2], and Irina Virbitskaite[2,3](✉)

[1] Department of Computing Science,
Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany
eike.best@informatik.uni-oldenburg.de
[2] A.P. Ershov Institute of Informatics Systems, SB RAS,
6, Acad. Lavrentiev av., 630090 Novosibirsk, Russia
{gribovskaya,virb}@iis.nsk.su
[3] Novosibirsk State University, 2, Pirogov av., 630090 Novosibirsk, Russia

Abstract. Two structurally different methods of associating transition system semantics to event-oriented models of distributed systems are distinguished in the literature. One of them is based on configurations (event sets), the other on residuals (model fragments). In this paper, a variety of models is investigated, ranging from extended prime event structures to configuration structures, and it is shown that the two semantics lead to isomorphic results. This strengthens prior work where bisimilarity (but not necessarily isomorphism) is achieved for a smaller range of models. Thanks to the isomorphisms obtained here, a wide range of facts known from the literature on configuration-based transition systems can be extended to residual-based ones.

Keywords: Event structures · Configuration structures
Transition systems

1 Introduction

Two methods of associating a labeled transition system [18] with an event-oriented model of a distributed system, such as an event structure [27] or a configuration structure [12], can be distinguished: a *configuration-based* and a *residual-based* method. In the first case,[1] states are understood as sets of events, called *configurations*, and state transitions are built by starting with the empty configuration and enlarging configurations by already executed events. In the second approach,[2] states are understood as event structures, and transitions are built by starting with the given event structure as an initial state and removing already executed (or conflicting) parts thereof in the course of an execution.

E. Best, N. Gribovskaya and I. Virbitskaite—Supported by German Research Foundation through grant Be 1267/14-1.

[1] E.g., see [1,2,10,11,13–16,24,28].

[2] E.g., see [3,6,7,16,17,19,22,25].

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 117–139, 2018.
https://doi.org/10.1007/978-3-319-91268-4_7

In the literature, configuration-based transition systems seem to be predominantly used as the semantics of event structures, but residual-based transition systems are actively used in providing operational semantics of process calculi and in demonstrating the consistency of operational and denotational semantics. The two kinds of transition systems have occasionally been used alongside each other (see [16] as an example), but their general relationship has not been studied for a wide range of existing models. In a seminal paper, viz. [23], bisimulations between configuration-based and residual-based transition systems have been proved to exist for prime event structures [29]. The result of [23][3] has been extended in [5] to more complex event structure models with asymmetric conflict. Counterexamples[4] illustrate that an isomorphism cannot be achieved with the various removal operators defined in [5,23].

The present paper demonstrates that the operators can be tightened in such a way that *isomorphisms*, rather than just bisimulations, between the two types of transition systems belonging to a single event structure can be obtained. A key idea is to employ *non-executable* events[5] if the model allows them (and to introduce a special non-executable event otherwise), in order to turn *model fragments* into parts of states.[6] This idea has been applied by the authors on a wide variety of event structure models, and for a full spectrum of semantics (interleaving, step, pomset, multiset). Some of the resulting combinations present substantial formal difficulties due to model and semantical overhead. Nevertheless, as will be reported in the present paper, the definitions can be achieved successfully, and the corresponding isomorphism results proved, in all of the considered cases.

In Sect. 2 of this paper, we shall thus define removal operators which are tightened in this way, for the spectrum of event-oriented models summarised in Sect. 1.2 below, and demonstrate the correctness of the operators in different semantics. Section 3 contains the definition of the two types of transition systems and describes the main isomorphism results of this paper. Section 4 concludes. The paper also contains Appendix A with proofs.

1.1 Prime Event Structures, and Motivating Remarks

All removal operators that are introduced in Sects. 2.1–2.5 use emulations of non-executable events. Such events cannot be expressed in the basic prime event structure model of [29], for which [23] has been designed. In this subsection, we show how such an event can be added usefully for the purpose of proving isomorphism, rather than bisimulation, between the resulting transition systems.

A labeled prime event structure (P-structure) over a set L of actions is a tuple $\mathcal{E} = (E, \sharp, \leq, L, l)$, where E is a set of events; $\leq \subseteq E \times E$ is a partial order (the *causality relation*), satisfying the *principle of finite causes*:

[3] Recalled below in Sect. 1.1.

[4] One of which is also shown in Sect. 1.1.

[5] In an event structure, an event is called non-executable or impossible if it does not occur in any configuration of the structure, i.e. the event is never executed.

[6] As explained in Sect. 1.1 on the example described there.

$\forall e \in E\colon \lfloor e \rfloor = \{e' \in E \mid e' \leq e\}$ is finite; $\sharp \subseteq E \times E$ is an irreflexive and symmetric relation (the *conflict relation*), satisfying the *principle of hereditary conflict*: $\forall e, e', e'' \in E\colon e \leq e'$ and $e \sharp e''$ then $e' \sharp e''$; and $l : E \to L$ is a labeling function. So, a P-structure over L is a simple event based model of (concurrent) computations where events labeled over L are considered as atomic, indivisible and instantaneous action occurrences, some of which can only be executed after another (i.e. there is a causal dependency represented by a partial order $\leq$ between the events) and some of which might not be executed together (i.e. there is a binary conflict $\sharp$ between the events). Two events that are neither in causal dependency nor in conflict are considered to be concurrent. In addition, the principle of finite causes and the principle of conflict inheritance are required. Therefore, non-executable events cannot be expressed in the model. For example, the left-hand side of Fig. 1 shows a P-structure $\mathcal{E}^p$ with an identity labeling function, where pairs of events related by causality are connected by arrows (for the pairs derivable from the transitivity property, the arrows are not shown), and pairs of the events included in conflict are marked by a symbol $\sharp$ (for the pairs derivable from the hereditary conflict principle, symbols $\sharp$ are not depicted). The states of a computation of P-structure is called configurations—finite subsets of conflict-free events left-closed with respect to the causality relation.

We can represent the behavior of P-structure as an LTS $TC(.)$ where states are configurations and transitions between configurations are set inclusions; the initial state is the empty configuration. In [23,25], for a prime event structure $\mathcal{E}$ and its configuration X, a removal operator for the construction of residuals has been defined as follows: $\mathcal{E} \setminus X = (E', \leq \cap (E' \times E'), \sharp \cap (E' \times E'), L, l \mid_{E'})$, with $E' = E \setminus (X \cup \sharp(X))$, where $\sharp(X)$ denotes the events conflicting with the events in X. With this, an LTS $TR(.)$ can be defined as follows: states are understood as event structures, and transitions are built by starting with the given event structure as an initial state and removing the events from already executed configurations and the events conflicting with the executed ones. For our example, Fig. 1 presents $TC(\mathcal{E}^p)$ (in the middle) and $TR(\mathcal{E}^p)$ (on

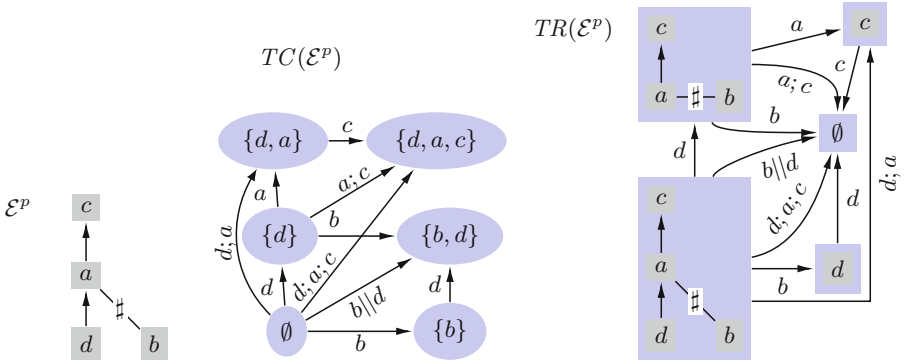


Fig. 1. A prime event structure and two bisimilar LTS derived from it as in [23]

the right-hand side), in their pomset semantics. The main result of [23] states that the two transition systems are bisimilar. Observe that this is true in our example, but also, that $TC(\mathcal{E}^p)$ and $TR(\mathcal{E}^p)$ are not isomorphic. The culprits are the maximal states, of which there are two in $TC(\mathcal{E}^p)$, obtained by executing, respectively, $d; a; c$ and $d \| b$, but in $TR(\mathcal{E}^p)$ only one—the empty event structure obtained as the residual by applying the removal operator with the corresponding configurations.

Suppose that we extend prime event structures just slightly by allowing a non-executable event, say denoted by $*$. On the left-hand side of Fig. 2, an event structure $\mathcal{E}^{p*}$ extending the previous one in this way is shown. Using $*$, we can specify parts of an event structure as non-executable, rather than deleting them. For instance, consider $TR(\mathcal{E}^{p*})$ shown on the right-hand side of Fig. 2, with the initial state being the middle one on the lower line. After executing b as a first event, a and c are no longer possible, but instead of being deleted as in $TR(\mathcal{E}^p)$, they were positioned after the (unique) $*$ event.[7] Doing this systematically, we obtain the residual transition system $TR(\mathcal{E}^{p*})$ shown in full on the right hand side of Fig. 2. Observe that $TR(\mathcal{E}^{p*})$ is isomorphic – rather than just bisimilar – to the original configuration-based transition system $TC(\mathcal{E}^p)$, which is, of course, the same as $TC(\mathcal{E}^{p*})$.

All event structure models investigated in Sect. 2 can simulate an $*$ event. Therefore, we were able to define a suitable removal operator[8] which leads to the desired (and announced, and proved) transition system isomorphism.[9]

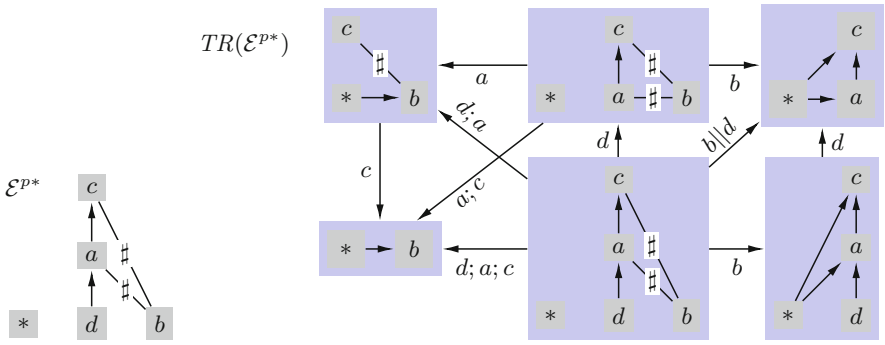


Fig. 2. An (extended) prime event structure with an $*$ event, and a novel residual LTS

[7] If the transitivity of $\rightarrow$ is assumed, the arrow from $*$ to c could be omitted, of course.

[8] Even if this was technically not always straightforward.

[9] The general idea of retaining an appropriate amount of structure during residual semantics (or, for that matter, unfolding semantics, when applied, e.g., to a process algebra) is not novel, nor claimed to be so. However, its application to the transition semantics of event structures is hoped to shed some new light onto this general idea.

1.2 Event-Oriented Models to Be Considered in This Paper

In Sect. 2, the following models of varying complexity will be considered:

- Section 2.1: *(extended [1]) prime event structures* [29] with conjunctive[10] binary causality, unique enabling, and symmetric irreflexive binary conflict;
- Section 2.2: *bundle event structures* [19] with non-binary conjunctive causality, alternative enablings, and a stability constraint (each causal predecessor set for an event contains only conflicting events) ensuring unique enabling within a configuration; and *dual event structures* [19] with disjunctive causality[11] allowing causal ambiguity even in a single configuration;
- Section 2.3: *flow event structures* [6] with binary conjunctive causality, allowing for alternative enablings, and self-conflicting events;
- Section 2.4: *stable event structures* [29] with non-binary conjunctive causality, alternative enablings, and a stability constraint (the intersection of non-conflicting causal predecessor sets for an event is a causal predecessors set for it), resulting in unique enabling within a configuration; and, dropping the stability constraint, *general event structures* [29] with disjunctive causality;
- Section 2.5: *configuration structures* [11–14] that represent the behaviours of event-oriented models as the collections of their configurations.

When comparing the families of (finite reachable) configurations of the models they can express, configuration structures are more expressive than general/stable event structures; which are more expressive than flow event structures; which are more expressive than bundle event structures; which are, in turn, more expressive than (extended) prime event structures.

2 Removal Operators for Some Event-Oriented Models

In Sects. 2.1–2.5, we define configurations, as well as removal operators, for some variants of Winskel's original event structures [27]. Section 2.5 differs slightly, in that (finitary) configuration structures described in [11,14] are considered. They can be thought of as the "essence" of configuration semantics appearing in all other models, and thus as a virtual bracket between them. In Sect. 2.6, we demonstrate the correctness of the removal operators in different (interleaving, step, pomset, multiset) semantics.

2.1 Extended Prime Event Structures

For reasons of flexibility, the authors of [1] propose to generalise ordinary prime event structures [29] by dropping the transitivity and acyclicity of causality, as well as the principles of finite causes and conflict inheritance.[12] As opposed to prime event structures, the extended version allows for non-executable (impossible) events. We will use those in order to construct residual event structures.

[10] An event is enabled once all of its causal predecessors have occurred.

[11] An event is enabled once at least one of its causal predecessors have occurred.

[12] It was noted in [1] that, as far as finite configurations are concerned, this does not lead to an increase in expressive power.

Definition 1. *An* extended prime event structure (*EP*-structure) over L *is a triple* $\mathcal{E} = (E, \sharp, \rightarrow, L, l)$*, where* E *is a set of events;* $\sharp \subseteq E \times E$ *is an irreflexive symmetric relation (the* conflict relation*);* $\rightarrow \subseteq E \times E$ *is the* enabling relation*;* L *is a set of labels;* $l : E \rightarrow L$ *is a labeling function. Let* $\mathbb{E}_L^{ep}$ *denote the class of EP-structures over* L*.*

Let $\mathcal{E}$ be an *EP*-structure. For $X \subseteq E$, let $\sharp(X) = \{e' \in E \mid \exists e \in X : e \sharp e'\}$.[13] We call a set $X \subseteq E$ a *configuration* of $\mathcal{E}$ if X is finite, conflict-free (i.e. $\forall e, e' \in X : \neg(e \sharp e')$), left-closed (i.e. $\forall e, e' \in E : e \rightarrow e' \wedge e' \in X \Rightarrow e \in X$), and does not contain enabling cycles (i.e., $\nexists e_1, \ldots, e_n \in X : e_1 \rightarrow \ldots \rightarrow e_n$ and $e_n = e_1$ $(n > 1)$). The set of configurations of $\mathcal{E}$ is denoted by *Conf*$(\mathcal{E})$.

In an *EP*-structure, an event e is called *impossible* if it does not occur in any of the configurations. Events can be impossible because of enabling cycles, or infinite causes, or an overlapping between the enabling and the conflict relation, or because of impossible predecessors.

In the graphical representation of an *EP*-structure, pairs of events related by the enabling relation are connected by arrows; pairs of the events included in the conflict relation are marked by the symbol $\sharp$.

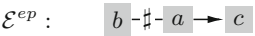


Fig. 3. An extended prime event structure $\mathcal{E}^{ep}$

Example 1. Figure 3 depicts the *EP*-structure $\mathcal{E}^{ep}$ over $L = \{a, b, c\}$, with $E^{ep} = L; \sharp^{ep} = \{(a, b), (b, a)\}$; $\rightarrow^{ep} = \{(a, c)\}$; and the identity labeling function l^{ep}. Observe that the principle of conflict inheritance is violated. The set of configurations of $\mathcal{E}^{ep}$ is $\{\emptyset, \{a\}, \{b\}, \{a, c\}\}$.

Definition 2. *For* $\mathcal{E} \in \mathbb{E}_L^{ep}$ *and* $X \in$ *Conf*$(\mathcal{E})$*, a removal operator is defined as follows:* $\mathcal{E} \setminus X = (E', \rightarrow', \sharp', L, l')$*, with*

$$
\begin{aligned}
E' &= E \setminus X \\
\sharp' &= \sharp \cap (E' \times E') \\
\rightarrow' &= (\rightarrow \cap (E' \times E')) \ \cup \ \{(e, e) \mid e \in \sharp(X)\} \\
l' &= l \mid_{E'}
\end{aligned}
$$

We see that the events in X are removed, yielding a reduction of the enabling and conflict relations. At the same time, any event conflicting with some event in X is retained, equipping it with an enabling cycle, thereby making the conflicting event impossible.

Consider auxiliary

Lemma 1. *Given* $\mathcal{E} \in \mathbb{E}_L^{ep}$ *and* $X \in$ *Conf*$(\mathcal{E})$*, (i)* $\mathcal{E} \setminus X \in \mathbb{E}_L^{ep}$*; (ii)* $X \cup X'$ *is conflict-free and* $\sharp(X \cup X') = \sharp(X) \cup \sharp'(X')$*, for any* $X' \in$ *Conf*$(\mathcal{E} \setminus X)$*.*

[13] We shall use the notation for the other models through the paper.

2.2 Bundle and Dual Event Structures

Bundle event structures were introduced in [19] for the description of formal semantics of the specification language LOTOS for parallel systems and the corresponding algebra of processes PA [16]. Unlike events in prime structures, those in bundle structures can be initiated by different sets of events. Causality is not a binary relation anymore; instead, it is represented by the bundle relation $\rightarrowtail$ between a finite set W of events and event e. A pair (W, e) such that $W \rightarrowtail e$ is called a bundle, and W is called a bundle set. The bundle set contains only pairwisely conflicting events (which is known as the stability principle). This causality relation can be interpreted as follows: in the system's functioning, an event e can occur only if one of the events from the set W has already occurred. Dual event structures [20] extend bundle structures by dropping the stability principle. This leads to causal ambiguity; i.e., given a configuration and one of its events, it is not always possible to determine what caused this event.

Definition 3. *A* dual event structure (D-structure) over L *is a tuple* $\mathcal{E} = (E, \sharp, \rightarrowtail, L, l)$*, where* E *is a set of events;* $\sharp \subseteq E \times E$ *is an irreflexive and symmetric relation (the* conflict relation*);* $\rightarrowtail \subseteq 2^E \times E$ *is the* bundle relation*;* L *is a set of labels;* $l : E \to L$ *is a labeling function.* $\mathcal{E}$ *is a* bundle event structure (B-structure) over L *if the stability principle holds:* $\forall X \subseteq E, e \in E : X \rightarrowtail e \Rightarrow \forall e_1, e_2 \in X$ *if* $e_1 \neq e_2$ *then* $e_1 \sharp e_2$*. Let* $\mathbb{E}_L^{b/d}$ *denote the class of* B/D*-structures over* L*.*

The behavior of a B/D-structure is described by explaining which subsets of events constitute possible (partial) runs of the represented system (thus formalising the interpretation of the bundle sets and the conflict relation). These subsets are called *configurations*. In other words, a set $X \subseteq E$ is a *configuration* of a B/D-structure $\mathcal{E}$ iff X is a finite set, conflict-free (i.e., $\neg(e \sharp e')$, for all $e, e' \in X$) and secured (i.e., there exist events $e_1, \ldots, e_n$ $(n \geq 0)$ such that $X = \{e_1, \ldots, e_n\}$, and for all $i < n$, if $Y \rightarrowtail e_{i+1}$, then $\{e_1, \ldots, e_i\} \cap Y \neq \emptyset$). The set of configurations of $\mathcal{E}$ is denoted by $Conf(\mathcal{E})$. The causal relation between events in a configuration X of a B-structure can be represented by the partial order $\leq_X = \{(d, e) \in X \times X \mid \exists Y : d \in Y \rightarrowtail e\}^*$. In D-structures, a configuration cannot be described by a single poset anymore, because of the causal ambiguity—a configuration may contain events whose causes are not determined uniquely.

The definition of the syntax of a B/D-structure allows an empty bundle, $\emptyset \rightarrowtail e$, to be defined. The behavioral interpretation of such a bundle is that e cannot occur in any configuration, i.e. e is an impossible event. Notice that there are alternative ways to specify impossible events, for instance $\{e\} \rightarrowtail e$ or $\{e'\} \rightarrowtail e \sharp e'$. It is known from [16,19], all the bundles with impossible events can always be eliminated while preserving the behaviour (in terms of configurations).

In the graphical representation of a B/D-structure, the bundles (W, e) are indicated by drawing an arrow from each element of W to e and connecting all the arrows by small lines. The pairs of the events included in the conflict relation are marked by the symbol $\sharp$.

Fig. 4. A bundle event structure $\mathcal{E}^b$ (l.h.s.) and dual event structure $\mathcal{E}^d$ (r.h.s.)

Example 2. Figure 4 (l.h.s.) shows the B-structure $\mathcal{E}^b$ over $L = \{a, b, c\}$, with $E^b = L; \sharp^b = \{(a, b), (b, a)\}; \rightarrowtail^b = \{(\{a, b\}, c)\}$; and the identity labeling function l^b. The set of configurations of $\mathcal{E}^b$ consists of the sets: $\emptyset, \{a\}, \{b\}, \{a, c\}, \{b, c\}$. Unlike conflicting causes of an event in an EP-structure, the conflicting causes a and b for the event c of $\mathcal{E}^b$ may appear in the configurations containing the event c.

Consider the D-structure $\mathcal{E}^d$ over $L = \{a, b, c\}$ depicted in Fig. 4 (r.h.s.). The components of $\mathcal{E}^d$ are the sets: $E^d = \{a, b, c\}$; $\sharp^d = \emptyset$; $\rightarrowtail^d = \{(\{a, b\}, c)\}$; and the identity labeling function l^d. Notice that $\mathcal{E}^d$ is not a B-structure because it has a bundle set with non-conflicting causes for the event c. It is easy to see that $Conf(\mathcal{E}^d) = \{\emptyset, \{a\}, \{b\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$. The last configuration contains the event c whose cause is not determined uniquely.

In [19], a removal operator for B-structures was defined and treated within the specification language LOTOS. It has turned out that the operator can be used for D-structures as well.

Definition 4. *For $\mathcal{E} \in \mathbb{E}_L^{b/d}$ and $X \in Conf(\mathcal{E})$, a removal operator is defined as follows: $\mathcal{E}' = \mathcal{E} \setminus X = (E', \rightarrowtail', \sharp', L, l')$, with*

$$\begin{aligned} E' &= E \setminus X \\ \sharp' &= \sharp \cap (E' \times E') \\ \rightarrowtail' &= \big(\rightarrowtail \setminus \{(W, e) \in \rightarrowtail \mid W \cap X \neq \emptyset\}\big) \cup \{(\emptyset, e) \mid e \in \sharp(X)\} \\ l' &= l \mid_{E'} \end{aligned}$$

So, all the events in X are removed from E; the conflict relation $\sharp'$ is defined on the remaining conflicting events; bundles $W \mapsto e$ such that $W \cap X \neq \emptyset$ are eliminated from $\rightarrowtail$, because some cause of e already occurs in X; hence, e can be executed any next moment in $\mathcal{E}'$. The events conflicting with some event in X are retained in E', making them impossible by adding empty bundles.

The lemma below seems identical to Lemma 1 but it should be stressed that the residuals obtained by the removal operators are B/D-structures.

Lemma 2. *Given, $\mathcal{E} \in \mathbb{E}_L^{b/d}$ and $X \in Conf(\mathcal{E})$, (i) $\mathcal{E} \setminus X \in \mathbb{E}_L^{b/d}$; (ii) $X \cup X'$ is conflict-free and $\sharp(X \cup X') = \sharp(X) \cup \sharp'(X')$, for any $X' \in Conf(\mathcal{E} \setminus X)$.*

2.3 Flow Event Structures

Flow event structures introduced in [6] are another kind of event structures having a similar representation as prime event structures [29] but being far more

relaxed. First, the causality ordering in flow event structures is represented by an irreflexive flow relation that is not necessarily transitive and acyclic. Second, the symmetric conflict relation is not assumed to be irreflexive; this means that self-conflicting events are allowed. Such events cannot in general be removed from a flow structure without affecting its set of configurations. Third, there is no requirement on the relationships between the flow and the conflict relations. Fourth, the principles of finite causes and conflict inheritance are dropped. Boudol [6] provided translations between 1-reachable occurrence nets, flow nets, and flow event structures that have expressive power strictly between the bundle event structures of [19] and the stable event structures of [29].

Definition 5. *A* flow event structure (*F*-structure) over L *is a tuple* $\mathcal{E} = (E, \sharp, \prec, L, l)$*, where* E *is a set of events;* $\sharp \subseteq E \times E$ *is a symmetric relation (the* conflict relation*);* $\prec \subseteq E \times E$ *is an irreflexive relation (the* flow relation*);* L *is a set of labels;* $l : E \to L$ *is a labeling function. Let* $\mathbb{E}_L^f$ *denote the class of* F*-structures over* L*.*

Consider the notion of a configuration of F-structures. First, configurations must be finite and, moreover, conflict-free (hence, self-conflicting events will never occur in any configuration, i.e. they are impossible). Second, for an event to occur it is necessary that a complete non-conflicting set of its immediate causes has occurred. Here, we say that d is a *possible immediate cause* for e iff $d \prec e$, and a set of causes is *complete* if for any cause which is not contained there is a conflicting cause which is included. Third, no cycles with respect to causal dependence may occur. A set $X \subseteq E$ is a *configuration* of an F-structure $\mathcal{E}$ iff X is a finite set, conflict-free (i.e., for all $e, e' \in X$ $\neg(e \sharp e')$), left-closed up to conflicts (i.e., for all $d, e \in E$ if $e \in X, d \prec e$ and $d \notin X$ then there is $f \in X$ such that $d \sharp f \prec e$), and does not contain flow cycles (i.e., $\leq_X := \left(\prec \cap (X \times X)\right)^*$ is an ordering). The set of configurations of $\mathcal{E}$ is denoted $\mathit{Conf}(\mathcal{E})$.

In the graphical representation of an F-structure, $e \prec e'$ is drawn as an arrow from e to e'; the pairs of the events included in the conflict relation are marked by the symbol $\sharp$; and the self-conflicts are pictured as dotted circles around the events.

Example 3. Figure 5 presents the F-structure $\mathcal{E}^f$ over $L = \{a, b, c, d, e, f\}$, with $E^f = L$; $\sharp^f = \{(a,b),(b,a),(b,b),(b,c),(c,b),(a,d),\ (d,a),(c,f),(f,c)\}$; $<^f = \{(d,e),(a,e),(b,e),(c,e),(f,e)\}$; and the identity labeling function l^f. The set of configurations $\mathit{Conf}(\mathcal{E}^f)$ consists of the sets: $\emptyset, \{a\}, \{c\}, \{d\}, \{f\}, \{a,c\}$, $\{a,f\}, \{c,d\}, \{d,f\},\ \{a,c,e\}, \{a,f,e\}, \{c,d,e\}$. The F-structure $\mathcal{E}^f$ is not a

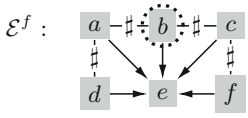


Fig. 5. A flow event structure $\mathcal{E}^f$

B/D-structure because $\mathcal{E}^f$ has the self-conflicting event b being a cause of the event e, b cannot be removed without affecting $\mathit{Conf}(\mathcal{E}^f)$—$\{d, f, e\}$ would be a configuration of $\mathcal{E}^f$.

Definition 6. *For $\mathcal{E} = (E, \prec, \sharp, L, l) \in \mathbb{E}_L^f$ and $X \in \mathit{Conf}(\mathcal{E})$, a removal operator is defined as follows: $\mathcal{E} \setminus X = (E', \sharp', \prec', L, l')$, with*

$$
\begin{aligned}
E' &= E \setminus X \\
\sharp' &= (\sharp \cap (E' \times E')) \cup \{(e, e) \mid e \in \sharp(X)\} \\
\prec' &= (\prec \cap (E' \times E')) \setminus \{(e, f) \in \prec \mid \exists e' \in X \ : \ e \sharp e' \prec f\} \\
l' &= l \mid_{E'}
\end{aligned}
$$

The intuitive interpretation of the above definition is as follows. All the events in X are removed from E; the conflict relation $\sharp'$ contains the pairs of remaining conflicting events and newly-added self-conflicting events being in conflict with some events in X; and the flow relation $\prec'$ includes the pairs of remaining events related by $\prec$ without the pairs (e, f) with e conflicting with some e' in X and f having immediate causes e and e'. We remove the pairs (e, f) because e and e' being in conflict belong to different complete sets of causes of f and the self-conflicting event e not being in conflict with the events from the intersection of the complete sets would prohibit a possible execution of f.

Lemma 3. *Given $\mathcal{E} \in \mathbb{E}_L^f$ and $X \in \mathit{Conf}(\mathcal{E})$,*

(i) $\mathcal{E} \setminus X \in \mathbb{E}_L^f$;
(ii) for any $X' \in \mathit{Conf}(\mathcal{E} \setminus X)$,
  (a) whenever $b \prec a$: if $a \in X \cup X'$ and $b \in X'$, then $a \in X'$; if $a \in X$ and $b \in X \cup X'$, then $b \in X$; if $a, b \in X'$, then $b \prec' a$;
  (b) $X \cup X'$ is conflict-free and $\sharp(X \cup X') = \sharp(X) \cup \sharp'(X')$.

2.4 Stable and General Event Structures

Stable and non-stable event structures, introduced in the work of Winskel [28] in order to overcome the unique enabling problem of prime event structures, have an enabling relation indicating which (usually finite) sets X of events are possible prerequisites of a single event e, written $X \vdash e$. This enables one to model disjunctive causality as in D-structures, the phenomenon that an event is causally dependent on a disjunction of other events occurring in the same system's run. We consider versions of stable and general event structures of [29] where the conflict relation is specified for sets with two events. The versions generalise the above event structure models, except for D-structures which generalise stable event structures, as shown in [16].

Definition 7. *A* general event structure (G-structure) over L *is a tuple $\mathcal{E} = (E, \sharp, \vdash, L, l)$, where*

- *E is a set of events;*
- *$\sharp \subseteq E \times E$ is an irreflexive, symmetric relation (the* conflict relation*). We shall write Con for the set of finite conflict-free subsets of E, i.e. those finite subsets $X \subseteq E$ for which $\forall e, e' \in X \ : \ \neg(e \sharp e')$. $X \in Con$ means that the events in X could happen in the same run, i.e. they are consistent;*
- *$\vdash \subseteq Con \times E$ is the* enabling relation *which satisfies $X \vdash e$ and $X \subseteq Y \in Con \Rightarrow Y \vdash e$. $\vdash$ indicates possible causes: an event e can occur whenever for some X with $X \vdash e$ all events in X have occurred before. The minimal enabling relation $\vdash_{min}$ is defined as follows: $X \vdash_{min} e$ iff $X \vdash e$ and for all $Y \subseteq X$ if $Y \vdash e$ then $Y = X$;*
- *L is a set of actions;*
- *$l : E \to L$ is a labeling function.*

$\mathcal{E}$ is a stable event structure (S-structure) over L *if the stability principle holds: $X \vdash e, Y \vdash e$, and $X \cup Y \cup \{e\} \in Con \Rightarrow X \cap Y \vdash e$. Let $\mathbb{E}_L^{s/g}$ denote the class of S/G-structures over L.*

A set $X \subseteq E$ is a *configuration* of an S/G-structure $\mathcal{E}$ iff X is finite, conflict-free (i.e., $X \in Con$), and secured (i.e., there are $e_1, \ldots, e_n$ such that $X = \{e_1, \ldots, e_n\}$ and $\{e_1, \ldots, e_i\} \vdash e_{i+1}$, for all $i < n$). The set of configurations of $\mathcal{E}$ is denoted $Conf(\mathcal{E})$. For an S-structure $\mathcal{E}, X \in Conf(\mathcal{E})$, and $e, e' \in X$, let $e' \prec_X e$ iff e' belongs to the smallest subset Y of X with $Y \vdash e$. Then, $\leq_X$, the causality relation on X, is defined as the reflexive transitive closure of $\prec_X$.

Example 4. Consider the S-structure $\mathcal{E}^s$ over $L = \{a, b, c, d\}$, with $E^s = L$; $\sharp^s = \{(a, b), (b, a)\}$; $\vdash^s_{min} = \{(\emptyset, a), (\emptyset, b), (\emptyset, c), (\{a\}, d), (\{b, c\}, d)\}$; and the identity labeling function l^s. The set of configurations of $\mathcal{E}^s$ is $\{\emptyset, \{a\}, \{b\}, \{c\}, \{a, c\}, \{b, c\}, \{a, d\}, \{a, c, d\}, \{b, c, d\}\}$.

Next, contemplate the G-structure $\mathcal{E}^g$ over $L = \{a, b, c, d\}$, with $E^g = L$; $\sharp^g = \{(a, b), (b, a), (b, c), (c, b)\}$; $\vdash^g_{min} = \{(\emptyset, a), (\emptyset, b), (\emptyset, c), (\{a\}, d), (\{b\}, d), (\{c\}, d)\}$; and the identity labeling function l^g. Clearly, $\mathcal{E}^g$ is not an S-structure because $(\{a\} \cup \{c\} \cup \{d\}) \in Con$ and $(\{a\} \cap \{c\}) \not\vdash_{\mathcal{E}^g} d$. The set of configurations of $\mathcal{E}^g$ is $\{\emptyset, \{a\}, \{b\}, \{c\}, \{a, c\}, \{a, d\}, \{b, d\}, \{c, d\}, \{a, c, d\}\}$.

Notice that neither $\mathcal{E}^s$ nor $\mathcal{E}^g$ is a flow event structure because the event c not conflicting with the event a may be a cause for d or may not.

Definition 8. *For $\mathcal{E} = (E, \sharp, \vdash, L, l) \in \mathbb{E}_L^{s/g}$ and $X \in Conf(\mathcal{E})$, a removal operator is defined as follows: $\mathcal{E} \setminus X = (E', \sharp', \vdash', L, l')$, with*

$$
\begin{array}{lcl}
E' & = & E \setminus X \\
\sharp' & = & \sharp \cap (E' \times E') \\
\vdash' & = & \{(W', e) \mid W' \in Con',\ \exists (W'', e) \in \vdash'_{min} \ \ s.t.\ W'' \subseteq W'\} \ \textit{where} \\
 & & \quad \vdash'_{min} = \{(W'', e) \mid \exists (W, e) \in \vdash_{min} \ \ s.t.\ W'' = W \cap E',\ e \in E', \\
 & & \qquad\qquad W'' \cup X \in Con,\ \{e\} \cup X \in Con\} \\
l' & = & l \mid_{E'}
\end{array}
$$

We see that all the events in X are deleted; the conflict relation $\sharp'$ contains the pairs of remaining conflicting events; the definition of $\vdash'$ is based on that of $\vdash'_{min}$, which consists of the pairs from $\vdash$ without the pairs whose events conflict with some event in X, thereby making them impossible.

Lemma 4. *Given $\mathcal{E} \in \mathbb{E}_L^{s/g}, X \in Conf(\mathcal{E})$, and $\mathcal{E}' = \mathcal{E} \setminus X$,*

(i) $A \cap E' \in Con'$, for any $A \in Con$, and $A' \in Con$, for any $A' \in Con'$;
(ii) $\mathcal{E}' \in \mathbb{E}_L^{s/g}$;
(iii) $X \cup X' \in Con$, and $W \cup X \cup X' \in Con$ iff $(W \setminus X) \cup X' \in Con'$ and $W \cup X \in Con$, for any $X' \in Conf(\mathcal{E}')$ and $W \subseteq E$.

2.5 Configuration Structures

In this section, we present and study the model of (finitary) configuration structures from [11,14] that generalises the families of configurations of event structures and also resembles the Chu spaces of [26]. Furthermore, the connections between configuration structures and Scott domains are established in [9].

Definition 9. *A* configuration structure (C-structure) over L *is a tuple $\mathcal{E} = (E, C, L, l)$, where E is a set of events, $C \subseteq \mathcal{P}(E)$ is a collection of subsets of events, L is a set of actions, $l : E \to L$ is a labeling function.*

The elements of E are events and the elements of C are configurations. An event is considered as an occurrence of an action the system may execute. A configuration X denotes a state of the system in which the events of X have occurred. From now on we will restrict our attention to *finitary* C-structures over L, meaning that all configurations are finite and the empty set of events is a configuration. Let $\mathbb{E}_L^c$ be the class of finitary C-structures over L. For technical convenience, we shall use $Conf(\mathcal{E})$ instead of C, in Sect. 2.6.

Example 5. Consider the C-structure $\mathcal{E}^c$ over $L = \{a, b, c\}$, with $E^c = L$; $C^c = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, b, c\}\}$; and the identity labeling function l^c. Note that $\mathcal{E}^c$ is not a G-structure because the events a and b are not causes for c and are not in the conflict relation with c, however, the sets $\{a, c\}$ and $\{b, c\}$ are not configurations of $\mathcal{E}^c$.

Definition 10. *For $\mathcal{E} \in \mathbb{E}_L^c$ and $X \in C$, a removal operator is defined as follows: $\mathcal{E} \setminus X = (E', C', L, l')$, where*

$$E' = E \setminus X$$
$$C' = \{Y \subseteq E' \mid X \cup Y \in C\}$$
$$l' = l \mid_{E'}$$

That is, all events in X are deleted, and the configurations of $\mathcal{E}'$ are simply the subsets of E' whose unions with X give configurations of $\mathcal{E}$. Note that this definition is much less complex than the ones in the previous sections. Nevertheless, as we will see in Sect. 3, it also guarantees the desired isomorphism result. This can be interpreted as a formal vindication in favour of preferring the removal operators defined above (rather than those described in [5,23]).

Lemma 5. *Given $\mathcal{E} \in \mathbb{E}_L^c$ and $X \in C$, $\mathcal{E} \setminus X \in \mathbb{E}_L^c$.*

2.6 Correctness of the Removal Operators in Different Semantics

We first introduce auxiliary notations. From now on, we consider a structure $\mathcal{E}$ over L, as defined in the various previous subsections of Sect. 2. For configurations $X, X' \in \mathit{Conf}(\mathcal{E})$, we write:

- $X \rightarrow_{int} X'$ iff $X \subseteq X'$ and $X' \setminus X = \{e\}$;
- $X \rightarrow_{step} X'$ iff $X \subseteq X'$ and $X'' \in \mathit{Conf}(\mathcal{E})$, for all $X \subseteq X'' \subseteq X'$;
- $X \rightarrow_{pom} X'$ iff $X \subseteq X'$ and $\leq_{X' \setminus X}$ is defined;
- $X \rightarrow_{mset} X'$ iff $X \subseteq X'$.

From now on, we specify $\star$ as follows, if not defined otherwise:

$$\star \in \begin{cases} \{int, step, mset\}, & \text{if } \mathcal{E} \in \mathbb{E}_L^d \cup \mathbb{E}_L^g \cup \mathbb{E}_L^c, \\ \{int, step, mset, pom\}, & \text{otherwise.} \end{cases}$$

A configuration $X \in \mathit{Conf}(\mathcal{E})$ is a *configuration in $\star$-semantics* of $\mathcal{E}$ iff $\emptyset \rightarrow_\star^* X$, where $\rightarrow_\star^*$ is the reflexive and transitive closure of $\rightarrow_\star$. Let $\mathit{Conf}_\star(\mathcal{E})$ denote the set of configurations in $\star$-semantics of $\mathcal{E}$.

We state some correctness criteria for the removal operators introduced in the previous subsections. The meaning of the correctness properties is that the obtained residuals do not allow configurations that are disallowed by original structures. Also, in some sense, this signifies some compositionality properties of the removal operators.

Proposition 1. *Let $\mathcal{E}$ be a structure over L. Then,*

(i) for any $\mathcal{E}' = \mathcal{E} \setminus X$, with $X \in \mathit{Conf}_\star(\mathcal{E})$, and $\mathcal{E}'' = \mathcal{E}' \setminus X'$, with $X' \in \mathit{Conf}_\star(\mathcal{E}')$, $X \cup X' \in \mathit{Conf}_\star(\mathcal{E})$ and $\mathcal{E}'' = \mathcal{E} \setminus (X \cup X')$;

(ii) for any $X, X'' \in \mathit{Conf}_\star(\mathcal{E})$ such that $X \rightarrow_\star X''$, $X'' \setminus X \in \mathit{Conf}_\star(\mathcal{E} \setminus X)$ and, moreover, $\emptyset \rightarrow_\star X'' \setminus X$ in $\mathcal{E} \setminus X$.

3 Transition Systems $TC(.)$ and $TR(.)$, and Main Results

In this section, we first give some basic definitions concerning labeled transition systems. Then, we define the mappings $TC(\mathcal{E})$ and $TR(\mathcal{E})$, which associate two distinct kinds of transition systems – one whose states are configurations and one whose states are residual event structures – with the structures $\mathcal{E}$ over L.

A transition system $T = (S, \rightarrow, i)$ over a set $\mathcal{L}$ of labels consists of a set of states S, a transition relation $\rightarrow \subseteq S \times \mathcal{L} \times S$, and an initial state $i \in S$. Two transition systems over $\mathcal{L}$ are *isomorphic* if their states can be mapped one-to-one to each other, preserving transitions and initial states.

Let L be a fixed set of labels (of event structures). Let $\mathbb{L}_{int} := L$, and $\mathbb{L}_{step/mset} := \mathbb{N}_0^L$ (the set of multisets over L, or functions from L to the non-negative integers), and $L_{pom} := Pom_L$ (the set of isomorphic classes of partial

orders labeled over L) be another sets of labels. (The sets will be used as the set of labels of the transition systems.)

We need an additional auxiliary notation. For a structure $\mathcal{E}$ over L and configurations $X, X' \in Conf(\mathcal{E})$ such that $X \rightarrow_\star X'$ ($\star \in \{int, step, pom, mset\}$), we write:

- $l_{int}(X' \setminus X) = a \in \mathbb{L}_{int}$ iff $X' \setminus X = \{e\}$ and $l(e) = a$, if $\star = int$;
- $l_{step/mset}(X' \setminus X) = M \in \mathbb{L}_{step/mset}$ iff $M(a) = |\{e \in X' \setminus X \mid l(e) = a\}|$, for all $a \in L$, if $\star \in \{step, mset\}$;
- $l_{pom}(X' \setminus X) = \mathcal{Y} \in \mathbb{L}_{pom}$ iff $\mathcal{Y} = [(X' \setminus X, \leq_{X'} \cap (X' \setminus X \times X' \setminus X), L, l\,|_{X' \setminus X})]$, if $\star = pom$.

We are ready to construct two kinds of labeled transition systems.

Definition 11. *For a structure $\mathcal{E}$ over L,*

- *$TC_\star(\mathcal{E})$ is the transition system $(Conf_\star(\mathcal{E}), \rightarrow_\star, \emptyset)$ over $\mathbb{L}_\star$, where $X \xrightarrow{p}_\star X'$ for $p \in \mathbb{L}_\star$ iff $X \rightarrow_\star X'$ and $p = l_\star(X' \setminus X)$;*
- *$TR_\star(\mathcal{E})$ is the transition system $(Reach_\star(\mathcal{E}), \rightarrow_\star, \mathcal{E})$ over $\mathbb{L}_\star$, where $\mathcal{F} \xrightarrow{p}_\star \mathcal{F}'$ for some $p \in \mathbb{L}_\star$ iff $\mathcal{F}' = \mathcal{F} \setminus X$ and $\emptyset \rightarrow_\star X$ in $\mathcal{F}$, for some $X \in Conf_\star(\mathcal{F})$ with $p = l_\star(X)$, and $Reach_\star(\mathcal{E}) = \{\mathcal{F} \mid \exists \mathcal{E}_0, \ldots, \mathcal{E}_k\ (k \geq 0)$ s.t. $\mathcal{E}_0 = \mathcal{E}, \mathcal{E}_k = \mathcal{F}$, and $\mathcal{E}_i \xrightarrow{p}_\star \mathcal{E}_{i+1}\ (i < k)\}$.*

For instance, Figs. 6, 7, 8, 9, 10 and 11 indicate the transition systems $TR_\star(\cdot)$ with states—the residuals of the structures considered in Examples 1–5, respectively. Here, $\star = step$, if $\mathcal{E} \in \mathbb{E}^c_L$, $\star = mset$, if $\mathcal{E} \in \mathbb{E}^d_L \cup \mathbb{E}^g_L$, and $\star = pom$, otherwise. This implies that a single transition system arrow may represent, respectively, the occurrence of labeled concurrent events, the occurrence of a multiset of labeled non-conflicting enabled events, and the occurrence of partially ordered labeled enabled events.

Proposition 2. *Given a structure $\mathcal{E}$ over L,*

(i) for any $X \in Conf_\star(\mathcal{E}), \mathcal{E} \setminus X \in Reach_\star(\mathcal{E})$;
(ii) for any $\mathcal{E}' \in Reach_\star(\mathcal{E})$, there exists $X \in Conf_\star(\mathcal{E})$ such that $\mathcal{E}' = \mathcal{E} \setminus X$;
(iii) for any $X', X'' \in Conf_\star(\mathcal{E})$, if $X' \xrightarrow{p}_\star X''$, then $\mathcal{E} \setminus X' \xrightarrow{p}_\star \mathcal{E} \setminus X''$;
(iv) for any $\mathcal{E}', \mathcal{E}'' \in Reach_\star(\mathcal{E})$, if $\mathcal{E}' \xrightarrow{p}_\star \mathcal{E}''$, then there are $X', X'' \in Conf_\star(\mathcal{E})$ such that $\mathcal{E}' = \mathcal{E} \setminus X', \mathcal{E}'' = \mathcal{E} \setminus X''$, $X' \xrightarrow{p}_\star X''$.

We state the main result of the paper.

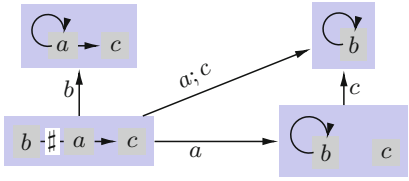


Fig. 6. The residual transition system $TR_{pom}(\mathcal{E}^{ep})$

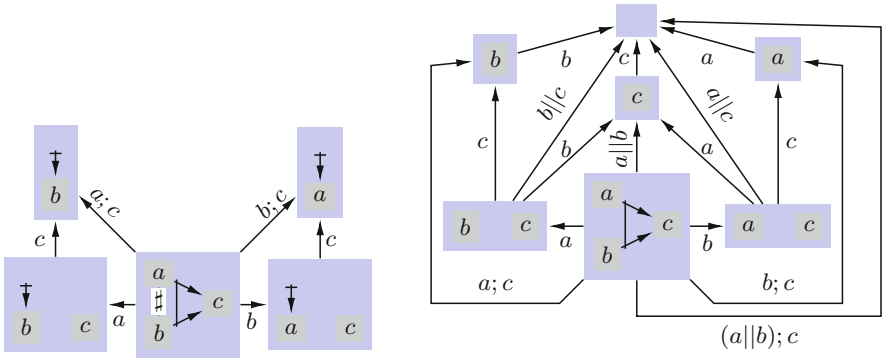


Fig. 7. The residual transition systems $TR_{pom}(\mathcal{E}^b)$ (l.h.s.) and $TR_{mset}(\mathcal{E}^d)$ (r.h.s.)

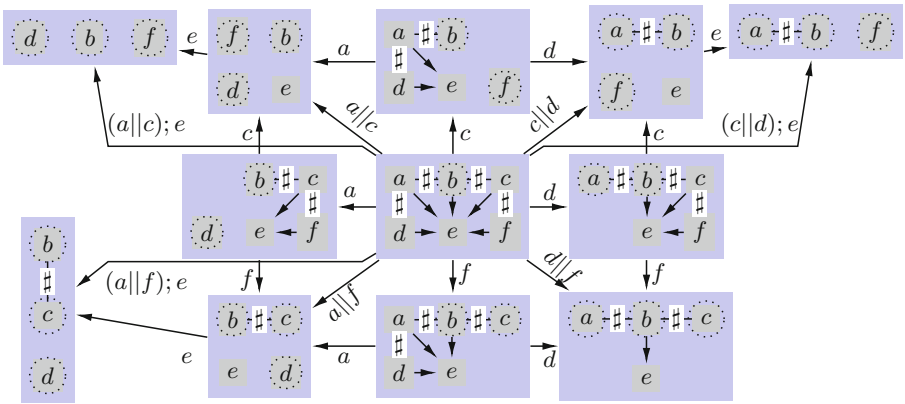


Fig. 8. The residual transition system $TR_{pom}(\mathcal{E}^f)$

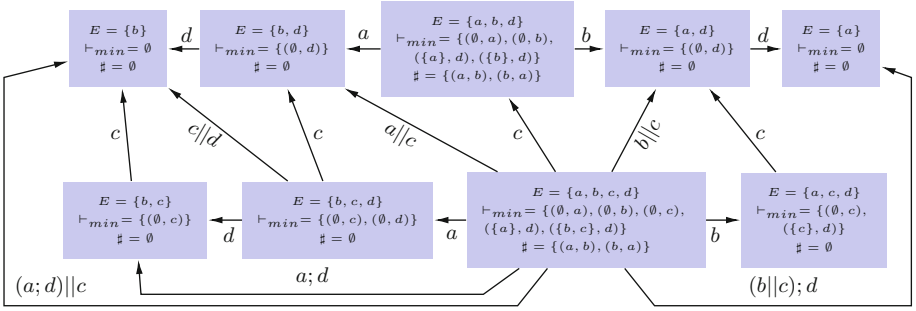


Fig. 9. The residual transition system $TR_{pom}(\mathcal{E}^s)$

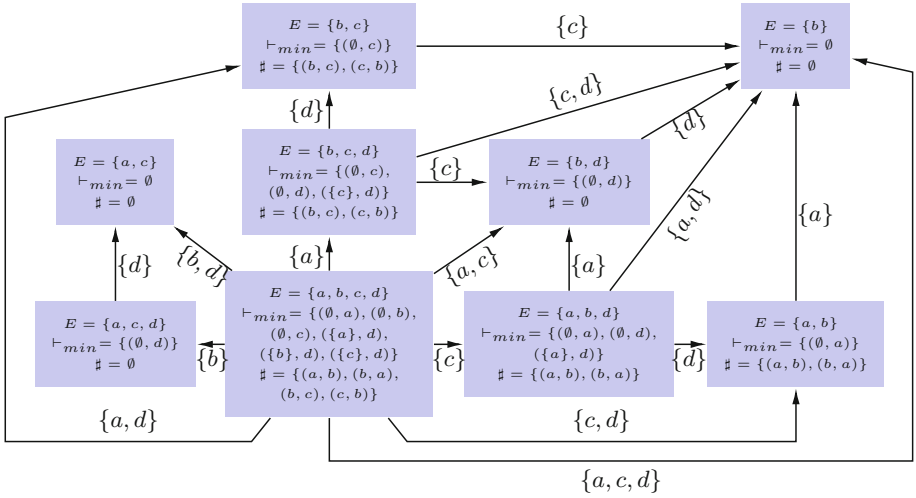


Fig. 10. The residual transition system $TR_{mset}(\mathcal{E}^g)$

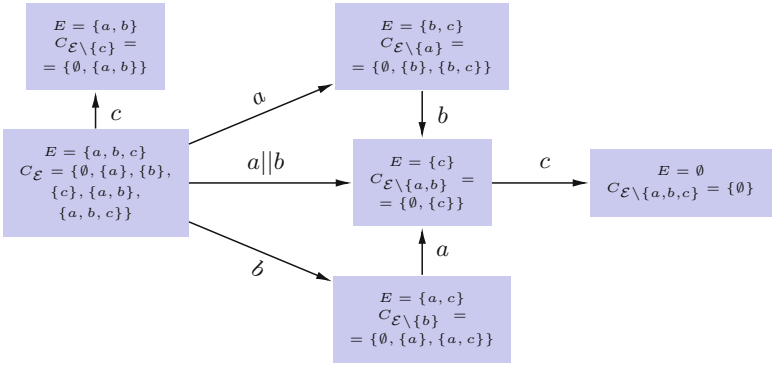


Fig. 11. The residual transition system $TR_{step}(\mathcal{E}^c)$

Theorem 1. *Given a structure $\mathcal{E}$ over L, $TC_\star(\mathcal{E})$ and $TR_\star(\mathcal{E})$ are isomorphic.*

It is well-known that the families of (finite and infinite) configurations of any structure $\mathcal{E}$ from $(\mathbb{E}^{ep}_L \cup \mathbb{E}^b_L \cup \mathbb{E}^d_L \cup \mathbb{E}^f_L \cup \mathbb{E}^s_L \cup \mathbb{E}^g_L)$ are completely determined by its finite configurations defined in Sects. 2.1–2.4. Specify the configuration structure $\mathcal{C}(\mathcal{E}) = (E, \mathit{Conf}(\mathcal{E}), L, l)$, where E is the set of events and l is the labeling function of $\mathcal{E}$. Then, it is easy to check the truth of the equality $TC_\star(\mathcal{C}(\mathcal{E})) = TC_\star(\mathcal{E})$. So, we get the following corollary, which provides the formal vindication announced at the end of Sect. 2.5.

Corollary 1. *Given a structure $\mathcal{E}$ over L and $\star \in \{int, step, mset\}$, $TR_\star(\mathcal{E})$ and $TR_\star(\mathcal{C}(\mathcal{E}))$ are isomorphic.*

4 Concluding Remarks

In this paper, we have defined two structurally different ways of giving various (interleaving, step, pomset, and multiset) transition system semantics for a wide spectrum of event-oriented distributed system models. As our main result, we have obtained an isomorphism between the corresponding transition systems of each model. As a consequence, we have been able to argue that the various removal operators defined for this purpose are related to a simple form defined for an encompassing model, that of configuration structures.

The main technical contribution of this paper consists in defining appropriate formal concepts underlying the removal operators which are necessary for residual semantics. They are based on the notion of non-executable events and have to be defined meticulously, and individually for each of the models.

In a sense, our paper can be seen as unifying the use of such a technique which has already had plenty of impact (albeit scattered) in the literature, but has not, so far, been emphasised quite so explicitly. For example, the use of non-executable events in removal operators allows the author of [16] to develop algebraic calculi for (extended) bundle, (extended) dual, and various real-time and probabilistic extensions of event structures. The author of [6] needed to restrict their attention to a subclass (at least with inherited conflict) of flow event structures when deleting events conflicting with an executed configuration in a removal operator (but not using non-executable events). The presence of non-executable events allowed the authors of [1] to establish a close relationships between event structures with shrinking causality and dual event structures. In our experience, removal operators based on deleting already executed configurations and events conflicting with them are likely to become cumbersome, up to the point of unreadability, for models that are more complex than prime event structures. In sum, this device facilitates the elimination of non-fundamental inconsistencies between models and is therefore, we feel, useful in comparative semantics. Other papers confirm that non-executable events are technically convenient in expressing operations on concurrent models in a concise way (see, e.g., [21] where they are used in the definition of unfoldings).

Isomorphisms such as the ones obtained in this paper are expected to allow one to relate transition systems constructed on configurations and transition systems derived from denotational semantics of process calculi in a tight way. Moreover, thanks to our results, a variety of facts known from the literature on configuration-based transition systems (e.g., [4,9,12,29]) can be extended to residual-based ones. For instance, it is known from [12] that every stable (general) event structure with non-binary conflict is history preserving bisimulation equivalent to a stable (general) event structure with binary conflict. A similar statement is likely to be true with residuals playing the role of configurations in the bisimulation. As another example, consider [9] where van Glabbeek proposed transition systems as alternative presentations of domains. Due to our results, the classic duality between prime algebraic domains and partial orders of configurations of stable event structures, established in [29], can be extended to residual-based transition systems of the models. Furthermore, thanks to the

paper [4], we can expect a duality between weak prime domains and residual-based transition systems of a well-defined subclass of general event structures.

Another benefit of having isomorphism instead of bisimulation arises in the context of prime event structures. With a bisimulation between $TC(.)$ and $TR(.)$, [23] shows that the operator TC is a functor from a category of labeled prime event structures into a category of labeled transition systems, in interleaving and step semantics, whereas the operator TR is not, for any semantics. This is at variance with a postulate by Winskel and Nielsen [30] that any semantic model should form a category and its semantic operations should possess a categorical characterization. Isomorphisms between the two kinds of transition systems conform better to this postulate.

Work on extending our approach (e.g., to precursor [8], probabilistic [31], and local [15] event structures, to event structures with dynamic causality [1] and to labeled event structures with invisible actions) is presently under way and has yielded promising intermediate results. Another future line of research is to extend our results to the non-pure case of resolvable conflict event structures [13] and to the multiset transition relation. We also plan to develop some translations of event structures from the classes under consideration into resolvable conflict structures, so as to compare residual-based transition systems constructed from the original structures with the ones obtained after translation, extending the nice corresponding suggestion in [14]. Furthermore, it would be interesting to see if the $TR(.)$ operators proposed here preserve behavioural (trace, testing, bisimulation) equivalences of the event-oriented models under consideration.

A Proofs

This appendix contains the proofs of the results obtained for $\mathcal{E} \in \mathbb{E}_L^g$ and $\star = mset$. All the proofs can be found at http://www.iis.nsk.su/virb/BGV-proofsketches-2018.pdf.

Proof of Lemma 4. (i) Take an arbitrary $A \in Con$. This implies that $A \subseteq E$ and $\neg(x \sharp x')$, for all $x, x' \in A$. Check that $A \cap E' \in Con'$. Suppose a contrary, i.e. there is $a, a' \in A \cap E'$ such that $a \sharp' a'$. By the definition of $\sharp'$, this means that $a, a' \in A$ and $a \sharp a'$, contradicting $A \in Con$. So, $A \cap E' \in Con'$. Next, take an arbitrary $A' \in Con'$. We shall show that $A' \in Con$. Assume a contrary, i.e. there is $a, a' \in A'$ such that $a \sharp a'$. Since $A' \in Con'$, it holds that $A' \subseteq E'$. By the definition of $\sharp'$, we get $a \sharp' a'$, contradicting $A' \in Con'$. Hence, $A' \in Con$.

(ii) Follows from the definitions of the components of $\mathcal{E} \setminus X$.

(iii) Next, we show that $X \cup X' \in Con$. As $X \in Conf(\mathcal{E})$ $(X' \in Conf(\mathcal{E}'))$, we get $X \in Con$ $(X' \in Con')$. By item (i), it holds that $X' \in Con$. Suppose a contrary, i.e. $X \cup X' \notin Con$. Then, we can find $e' \in X'$ and $e \in X$ such that $e' \sharp e$. Since $X' \in Conf(\mathcal{E}')$, there are $e'_1, \ldots, e'_m$ $(m \geq 0)$ such that $X' = \{e'_1, \ldots, e'_m\}$ and $\{e'_1, \ldots, e'_i\} \vdash' e'_{i+1}$, for all $i < m$. W.l.o.g., assume $e' = e'_j$ for some $1 \leq j \leq m$. By the definition of $\vdash'$, there is

$W' \subseteq \{e'_1, \ldots, e'_{j-1}\}$ such that $(W', e'_j) \in \vdash'_{min}$. Then, due to the definition of $\vdash'_{min}$, there is $(W, e'_j) \in \vdash_{min}$ such that $W' = W \cap E'$ and $\{e'_j\} \cup X \in Con$, contradicting $e' \sharp e \in X$.

Check that $W \cup X \cup X' \in Con$ iff $(W \setminus X) \cup X' \in Con'$ and $W \cup X \in Con$. Assume $W \cup X \cup X' \in Con$. Clearly, it holds that $W \cup X \in Con$. Next, it easy to see that $(W \cup X \cup X') \cap E' \in Con'$, due to item (i). This means $(W \setminus X) \cup X' \in Con'$. Conversely, suppose $(W \setminus X) \cup X' \in Con'$ and $W \cup X \in Con$. By item (i), we have $(W \setminus X) \cup X' \in Con$. Moreover, we know that $X \cup X' \in Con$. Hence, it holds that $W \cup X \cup X' \in Con$. □

Proof of Proposition 1. (i) Let $\mathcal{E}' = \mathcal{E} \setminus X$ with $X \in Conf_\star(\mathcal{E})$ and $\mathcal{E}'' = \mathcal{E}' \setminus X'$ with $X' \in Conf_\star(\mathcal{E}')$.

Since $X \in Conf_\star(\mathcal{E})$ ($X' \in Conf_\star(\mathcal{E}')$), X (X') is a finite, conflict-free, and secured subset of E (E') and $\emptyset \rightarrow^*_\star X$ in $\mathcal{E}$ ($\emptyset \rightarrow^*_\star X'$ in $\mathcal{E}'$). Obviously, $X \cup X'$ is a finite subset of E, by the definition of E'. Next, due to Lemma 4(iii), we have that $X \cup X' \in Con$. Further, we check that $X \cup X'$ is secured. As X is secured, there is a sequence $e_1 \ldots e_n$ ($n \geq 0$) such that $X = \{e_1, \ldots, e_n\}$, and $\{e_1, \ldots, e_i\} \vdash e_{i+1}$, for all $i < n$. Moreover, there exists a sequence $e'_1 \ldots e'_m$ ($m \geq 0$) such that $X' = \{e'_1, \ldots, e'_m\}$, and $\{e'_1, \ldots, e'_j\} \vdash' e'_{j+1}$, for all $j < m$, because X' is secured in $\mathcal{E}'$. Consider a sequence $e_1 \ldots e_n\, e_{n+1} = e'_1 \ldots e_{n+m} = e'_m$. Clearly, $\{e_1, \ldots, e_n, e_{n+1}, \ldots, e_{n+m}\} = X \cup X'$. Check that $\{e_1, \ldots, e_l\} \vdash e_{l+1}$, for all $l < n + m$. We know that $\{e_1, \ldots, e_l\} \vdash e_{l+1}$, for all $l < n$. Consider the case when $l = n$. Obviously, $\emptyset \vdash' e_{n+1} = e'_1$, i.e. $\emptyset \vdash'_{min} e_{n+1} = e'_1$. This means that there is $(W, e'_1) \in \vdash_{min}$ such that $\emptyset = W \cap E'$ and $\{e'_1\} \cup X \in Con$. Since $W \subseteq E' \cup X$ and $W \cap E' = \emptyset$, we get $W \subseteq X \in Con$. Then, $X \vdash e_{n+1}$, i.e. $\{e_1, \ldots, e_n\} \vdash e_{n+1}$. Finally, we verify the case when $n + 1 \leq l < n + m$. We know that $\{e'_1, \ldots, e'_{l-n}\} \vdash' e'_{l-n+1}$. Then, $W' \vdash'_{min} e'_{l-n+1}$, for some $W' \subseteq \{e'_1, \ldots, e'_{l-n}\}$. This means that there is $(W, e'_{l-n+1}) \in \vdash_{min}$ such that $W' = W \cap E', \{e'_{l-n+1}\} \cup X \in Con$, and $W' \cup X \in Con$. So, $W' \cup X \vdash e'_{l-n+1}$. Since $X \cup X' \in Con$ and $W' \subseteq \{e'_1, \ldots, e'_{l-n}\} \subseteq X'$, $X \cup \{e'_1, \ldots, e'_{l-n}\} \in Con$. Then, $\{e_1, \ldots, e_n, e_{n+1}, \ldots, e_l\} \vdash e_{l+1}$. Hence, $X \cup X'$ is a configuration of $\mathcal{E}$. Since $\emptyset \subseteq X \cup X'$ we get $\emptyset \rightarrow_\star X \cup X'$ in $\mathcal{E}$. Thus, $X \cup X' \in Conf_\star(\mathcal{E})$.

Set $\widetilde{\mathcal{E}} = \mathcal{E} \setminus (X \cup X')$. Check that $\mathcal{E}'' = \widetilde{\mathcal{E}}$. By definition, $\widetilde{E} = E \setminus (X \cup X') = (E \setminus X) \setminus X' = E' \setminus X' = E''$. Then, again by definition, $\sharp'' = \sharp' \cap (E'' \times E'') = (\sharp \cap (E' \times E')) \cap (E'' \times E'') = \sharp \cap (E'' \times E'') = \sharp \bigcap \widetilde{E} \times \widetilde{E} = \widetilde{\sharp}$, and $l'' = l \mid_{E''} = l \mid_{\widetilde{E}} = \widetilde{l}$. Using that $\widetilde{E} = E''$ and $\widetilde{\sharp} = \sharp''$, it is straightforward to verify that $\widetilde{Con} = Con''$. It remains to show that $\widetilde{\vdash} = \vdash''$. We know that $\widetilde{\vdash}_{min} = \{(\widetilde{W}, e)\} \mid \exists (W, e) \in \vdash_{min}$ s.t. $e \in \widetilde{E}, \widetilde{W} = W \cap \widetilde{E}, \{e\} \cup (X \cup X') \in Con, \widetilde{W} \cup (X \cup X') \in Con\}$. On the other hand, we have that $\vdash''_{min} = \{(W'', e) \mid \exists (W', e) \in \vdash'_{min}$ s.t. $e \in E'', W'' = W' \cap E'', \{e\} \cup X' \in Con', W'' \cup X' \in Con'\}$. Due to the definition of $\vdash'_{min}$, $\vdash''_{min} = \{(W'', e) \mid \exists (W, e) \in \vdash_{min}$ s.t. $e \in E' \cap E'', W'' = (W' = W \cap E') \cap E'', \{e\} \cup X \in Con$, $\{e\} \cup X' \in Con'$, $W'' \cup X' \in Con'$, $W' \cup X \in Con\} = \{(W'', e) \mid \exists (W, e) \in \vdash_{min}$ s.t. $e \in E'', W'' = W \cap E'', \{e\} \cup X \in Con, \{e\} \cup X' \in Con'$, $W'' \cup X' \in Con'$, $W' \cup X \in Con\}$. As $\widetilde{W} = W \setminus (X \cup X'), \widetilde{W} \cup X \cup X' = W \cup X \cup X'$, as $W' = W \setminus X, W' \cup X = W \cup X$, and

as $W'' = W' \setminus X', W'' \cup X' = W' \cup X' = (W \setminus X) \cup X'$. Hence, $\widetilde{W} \cup X \cup X' \in Con$ iff $W'' \cup X' \in Con'$ and $W' \cup X \in Con$, due to Lemma 4(iii). Notice that $e \notin X \cup X'$, because $e \in E''$. We also have that $\{e\} \cup X \cup X' \in Con$ iff $\{e\} \cup X' \in Con'$ and $\{e\} \cup X \in Con$, again by Lemma 4(iii). Due to $\widetilde{E} = E''$, we get that $\widetilde{\vdash}_{min} = \vdash''_{min}$. Moreover, it holds that $\widetilde{\vdash} = \{(\widetilde{W}, e)\} \mid \widetilde{W} \in \widetilde{Con}, \exists (W, e) \in \widetilde{\vdash}_{min}$ s.t. $W \subseteq \widetilde{W}\} = \{(W'', e)\} \mid W'' \in Con'', \exists (W, e) \in \vdash''_{min}$ s.t. $W \subseteq W''\} = \vdash''$, because $\widetilde{Con} = Con''$ and $\widetilde{\vdash}_{min} = \vdash''_{min}$.

(ii) Assume $X, X'' \in Conf_\star(\mathcal{E})$ and $X \rightarrow_\star X''$. Since $X'' \in Conf_\star(\mathcal{E})$, we have $X'' \in Conf(\mathcal{E})$, i.e. X'' is finite, conflict-free (i.e., $X'' \in Con$), and secured (i.e., there exists a sequence $t = e''_1 \dots e''_n$ of events from E ($n \geq 0$) such that $X'' = \{e''_1, \dots, e''_n\}$, and $\{e''_1, \dots, e''_i\} \vdash e''_{i+1}$, for all $i < n$. Using the sequence t and the fact that $X \rightarrow_\star X''$, i.e. $X \subseteq X''$, construct a sequence t' as follows: $t'_0 = \epsilon$, $t'_{i+1} = t'_i e''_{i+1}$, if $e''_{i+1} \notin X$, and $t'_{i+1} = t'_i$, otherwise. W.l.o.g. assume $t' = e'_1 \dots e'_k$ ($k \geq 0$) and $X' = \{e'_1, \dots, e'_k\}$. Clearly, X' is a finite subset of E'. As $X'' \in Con$, then $X'' \setminus X = X' = X'' \cap E' \in Con'$, by Lemma 4(i). We shall show that $\{e'_1, \dots, e'_j\} \vdash' e'_{j+1}$, for all $j < k$. Take an arbitrary $j < k$. Clearly, $e'_1, \dots, e'_j, e'_{j+1} \in X' \subseteq E'$ and $\{e'_1, \dots, e'_j\} \in Con'$. W.l.o.g., assume $e'_{j+1} = e''_{i+1}$, for some $e''_{i+1} \in X''$. As $X'' \in Conf(\mathcal{E})$, we get that $W'' = \{e''_1, \dots, e''_i\} \vdash e''_{i+1}$. Then, $W \vdash_{min} e''_{i+1}$, for some $W \subseteq W''$. Obviously, $W'' \cap E' = \{e'_1, \dots, e'_j\}$. Set $W' = W \cap E'$. Clearly, $W' \cup \{e''_{i+1}\} \cup X \subseteq X''$. As $X'' \in Con$, we get that $W' \cup X \in Con$ and $\{e''_{i+1}\} \cup X \in Con$. So, $W' \vdash'_{min} e'_{j+1}$. Since $W' \subseteq \{e'_1, \dots, e'_j\} \in Con'$, $\{e'_1, \dots, e'_j\} \vdash' e'_{j+1}$. This implies, X' is a configuration of $\mathcal{E} \setminus X$. Since $\emptyset \subseteq X'$, we have $\emptyset \rightarrow_\star X' = X'' \setminus X$ in $\mathcal{E} \setminus X$. Hence, $X' \in Conf_\star(\mathcal{E} \setminus X)$. □

Proof of Proposition 2. (i) Take an arbitrary $X \in Conf_\star(\mathcal{E})$. This means that $X_0 = \emptyset \rightarrow_\star X_1 \dots X_{n-1} \rightarrow_\star X_n = X$ ($n \geq$)) in $\mathcal{E}$. Then, we get that $X_i \in Conf_\star(\mathcal{E})$ ($0 \leq i \leq n$) and $X_{i-1} \subseteq X_i$ ($1 \leq i \leq n$). We shall proceed by induction on n.

$n = 0$. $\mathcal{E} \setminus X_0 = \mathcal{E} \in Reach_\star(\mathcal{E})$.

$n = 1$. By the induction hypothesis, $\mathcal{E} \setminus X_0 \in Reach_\star(\mathcal{E})$. Check that $\mathcal{E} \setminus X_0 \xrightarrow{p_1}_\star \mathcal{E} \setminus X_1$. Since $X_0 \rightarrow_\star X_1$ in $\mathcal{E}$, it holds $X_1 \setminus X_0 \in Conf_\star(\mathcal{E} \setminus X_0)$ and $\emptyset \rightarrow_\star X_1 \setminus X_0 = A_1$ in $\mathcal{E} \setminus X_0$, by Proposition 1(ii). Next, due to Proposition 1(i), we have $(\mathcal{E} \setminus X_0) \setminus A_1 = \mathcal{E} \setminus (X_0 \cup (X_1 \setminus X_0)) = \mathcal{E} \setminus X_1$. This implies that $\mathcal{E} \setminus X_0 \xrightarrow{p_1}_\star \mathcal{E} \setminus X_1$, where $p_1 = l_\star(A_1)$. So, $\mathcal{E} \setminus X_1 \in Reach_\star(\mathcal{E})$.

$n > 1$. By the induction hypothesis, $\mathcal{E} \setminus X_{n-2} \xrightarrow{p_{n-1}}_\star \mathcal{E} \setminus X_{n-1}$. Reasoning as in the previous item, we get that $\mathcal{E} \setminus X_{n-1} \xrightarrow{p_n}_\star \mathcal{E} \setminus X_n$, where $p_n = l_\star(A_n)$. Thus, $\mathcal{E} \setminus (X_n = X) \in Reach_\star(\mathcal{E})$.

(ii) Take an arbitrary $\mathcal{E}' \in Reach_\star(\mathcal{E})$. This means that $\mathcal{E} = \mathcal{E}_0 \xrightarrow{p_1}_\star \mathcal{E}_1 \dots \mathcal{E}_{n-1} \xrightarrow{p_n}_\star \mathcal{E}_n = \mathcal{E}'$ ($n \geq 0$). By the definition of $\xrightarrow{p_{i+1}}_\star$, it holds that $\mathcal{E}_{i+1} = \mathcal{E}_i \setminus X_{i+1}$, for some $X_{i+1} \in Conf_\star(\mathcal{E}_i)$ such that $\emptyset \rightarrow_\star X_{i+1}$ in $\mathcal{E}_i$ and $p_{i+1} = l_\star(X_{i+1})$ ($i < n$). Verify that $Y_{i+1} = \bigcup_{j=1}^{i+1} X_j \in Conf_\star(\mathcal{E})$ and $\mathcal{E}_{i+1} = \mathcal{E} \setminus Y_{i+1}$, for all $i < n$.

We shall proceed by induction on n.

$n = 0$. Obvious.
$n = 1$. Then, $Y_1 = X_1 \in Conf_\star(\mathcal{E})$ and $\mathcal{E}_1 = \mathcal{E}_0 \setminus X_1 = \mathcal{E} \setminus Y_1$.
$n > 1$. By the induction hypothesis, $Y_{n-1} = \bigcup_{j=1}^{n-1} X_j \in Conf_\star(\mathcal{E})$ and $\mathcal{E}_{n-1} = \mathcal{E} \setminus Y_{n-1}$. Check that $Y_n = \bigcup_{j=1}^{n} X_j \in Conf_\star(\mathcal{E})$ and $\mathcal{E}_n = \mathcal{E} \setminus Y_n$. As $\mathcal{E}_n = \mathcal{E}_{n-1} \setminus X_n$, it holds that $\mathcal{E}_n = (\mathcal{E} \setminus Y_{n-1}) \setminus X_n$. According to Proposition 1(i), we have that $Y_{n-1} \cup X_n \in Conf_\star(\mathcal{E})$ and $\mathcal{E}_n = \mathcal{E} \setminus (Y_{n-1} \cup X_n) = \mathcal{E} \setminus Y_n$.

Thus, $\mathcal{E}' = \mathcal{E} \setminus Y_n$.

(iii) It follows from the definitions of the transition relations and Proposition 1(i),(ii).

(iv) Due to item (ii), there is $X' \in Conf_\star(\mathcal{E})$ such that $\mathcal{E}' = \mathcal{E} \setminus X'$. According to the definition of $\xrightarrow{p}_\star$, there is $\widetilde{X}' \in Conf_\star(\mathcal{E}')$ such that $\mathcal{E}'' = \mathcal{E}' \setminus \widetilde{X}'$, $\emptyset \rightarrow_\star \widetilde{X}'$ in $\mathcal{E}'$, and $p = l_\star(\widetilde{X}')$. Then, $X'' = X' \cup \widetilde{X}' \in Conf_\star(\mathcal{E})$ and $\mathcal{E}'' = \mathcal{E} \setminus X''$, by Proposition 1(i). Clearly, $X', X'' \in Conf(\mathcal{E})$ and $X' \subseteq X' \cup \widetilde{X}' = X''$. Hence, $X' \rightarrow_\star X''$ in $\mathcal{E}$. Obviously, $l_\star(X'' \setminus X') = p$. Thus, $X' \xrightarrow{p}_\star X''$ in $\mathcal{E}$. □

Proof of Theorem 1. Define a mapping $g : Conf_\star(\mathcal{E}) \rightarrow Reach_\star(\mathcal{E})$ as follows: $g(X) = \mathcal{E} \setminus X$, for all $X \in Conf_\star\mathcal{E})$. Clearly, $g(\emptyset) = \mathcal{E}$. By the definition of $\mathcal{E} \setminus X$ and Proposition 2(i), g is well-defined. Check that g is a bijective mapping.

Suppose that $g(X) = g(X')$, for some $X, X' \in Conf_\star(\mathcal{E})$. This means that $\mathcal{E} \setminus X = \mathcal{E} \setminus X'$. By the definition of $\mathcal{E} \setminus X$ and $\mathcal{E} \setminus X'$, we get that $E \setminus X = E \setminus X'$. Since $X, X' \subseteq E$, we have that $X = X'$. Thus, g is an injective mapping.

Take an arbitrary $\mathcal{E}' \in Reach_\star(\mathcal{E})$. Due to Proposition 2(ii), we get that $\mathcal{E}' = \mathcal{E} \setminus X$, for some $X \in Conf_\star(\mathcal{E})$. So, g is a surjective mapping.

We finally show that $X \xrightarrow{p}_\star X'$ in $TC_\star(\mathcal{E})$ iff $g(X) \xrightarrow{p}_\star g(X')$ in $TR_\star(\mathcal{E})$. It follows from Proposition 2(iii) and (iv) and the fact that g is a bijective mapping.

Thus, g is indeed an isomorphism. □

References

1. Arbach, Y., Karcher, D., Peters, K., Nestmann, U.: Dynamic causality in event structures. In: Graf, S., Viswanathan, M. (eds.) FORTE 2015. LNCS, vol. 9039, pp. 83–97. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19195-9_6
2. Armas-Cervantes, A., Baldan, B., Garcia-Banuelos, L.: Reduction of event structures under history preserving bisimulation. J. Log. Algebr. Methods Program. **85**(6), 1110–1130 (2016)
3. Baier, C., Majster-Cederbaum, M.: The connection between event structure semantics and operational semantics for TCSP. Acta Inform. **31**, 81–104 (1994)
4. Baldan, P., Corradini, A., Gadducci, F.: Domains and event structures for fusions. In: LICS, pp. 1–12 (2017)

5. Best, E., Gribovskaya, N., Virbitskaite, I.: Configuration- and residual-based transition systems for event structures with asymmetric conflict. In: Steffen, B., Baier, C., van den Brand, M., Eder, J., Hinchey, M., Margaria, T. (eds.) SOFSEM 2017. LNCS, vol. 10139, pp. 132–146. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51963-0_11
6. Boudol, G.: Flow event structures and flow nets. In: Guessarian, I. (ed.) LITP 1990. LNCS, vol. 469, pp. 62–95. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-53479-2_4
7. Crafa, S., Varacca, D., Yoshida, N.: Event structure semantics of parallel extrusion in the pi-calculus. In: Birkedal, L. (ed.) FoSSaCS 2012. LNCS, vol. 7213, pp. 225–239. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28729-9_15
8. Fecher, H., Majster-Cederbaum, M.: Event structures for arbitrary disruption. Fundam. Inform. **68**(1–2), 103–130 (2005)
9. van Glabbeek, R.J.: History preserving process graphs. Report, Stanford University. http://boole.stanford.edu/rvg/pub/abstracts/history
10. van Glabbeek, R.J.: On the expressiveness of higher dimensional automata. Theor. Comput. Sci. **356**(3), 265–290 (2006)
11. van Glabbeek, R.J., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Inform. **37**, 229–327 (2001)
12. van Glabbeek, R.J., Plotkin, G.D.: Configuration structures. In: Proceedings of the LICS, pp. 199–209 (1995)
13. van Glabbeek, R., Plotkin, G.: Event structures for resolvable conflict. In: Fiala, J., Koubek, V., Kratochvíl, J. (eds.) MFCS 2004. LNCS, vol. 3153, pp. 550–561. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28629-5_42
14. van Glabbeek, R.J., Plotkin, G.D.: Configuration structures, event structures and Petri nets. Theor. Comput. Sci. **410**(41), 4111–4159 (2009)
15. Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: An event structure semantics for general Petri nets. Theor. Comput. Sci. **153**, 129–170 (1996)
16. Katoen, J.-P.: Quantitative and qualitative extensions of event structures. Ph.D. thesis, Twente University (1996)
17. Katoen, J.-P., Langerak, R., Latella, D.: Modeling systems by probabilistic process algebra: an event structures approach. In: IFIP Transactions, vol. C-2, pp. 253–268 (1993)
18. Keller, R.M.: Formal verification of parallel programs. Commun. ACM **19**(7), 371–384 (1976)
19. Langerak, R.: Bundle event structures: a non-interleaving semantics for LOTOS. In: Formal Description Techniques V. IFIP Transactions, vol. C-10, pp. 331–346 (1993)
20. Langerak, R., Brinksma, E., Katoen, J.-P.: Causal ambiguity and partial orders in event structures. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 317–331. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63141-0_22
21. Li, B., Koutny, M.: Unfolding CSPT-nets. In: Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2015), Brussels, Belgium, pp. 207–226, June 2015
22. Loogen, R., Goltz, U.: Modelling nondeterministic concurrent processes with event structures. Fundam. Inform. **14**(1), 39–74 (1991)
23. Majster-Cederbaum, M., Roggenbach, M.: Transition systems from event structures revisited. Inf. Process. Lett. **67**(3), 119–124 (1998)
24. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains. Theor. Comput. Sci. **13**(1), 85–108 (1981)

25. Nielsen, M., Thiagarajan, P.S.: Regular event structures and finite Petri nets: the conflict-free case. In: Esparza, J., Lakos, C. (eds.) ICATPN 2002. LNCS, vol. 2360, pp. 335–351. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-48068-4_20
26. Pratt, V.: Chu spaces and their interpretation as concurrent objects. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 392–405. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0015256
27. Winskel G.: Events in computation. Ph.D. thesis, University of Edinburgh (1980)
28. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) ACPN 1986. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-17906-2_31
29. Winskel, G.: An introduction to event structures. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) REX 1988. LNCS, vol. 354, pp. 364–397. Springer, Heidelberg (1989). https://doi.org/10.1007/BFb0013026
30. Winskel, G., Nielsen, M.: Models for concurrency. In: Handbook of Logic in Computer Science, vol. 4 (1995)
31. Winskel, G.: Distributed probabilistic and quantum strategies. Electron. Notes Theor. Comput. Sci. **298**, 403–425 (2013)

Analysis and Model Checking

Simplification of CTL Formulae for Efficient Model Checking of Petri Nets

Frederik Bønneland, Jakob Dyhr, Peter G. Jensen, Mads Johannsen, and Jiří Srba(✉)

Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark
srba@cs.aau.dk

Abstract. We study techniques to overcome the state space explosion problem in CTL model checking of Petri nets. Classical state space pruning approaches like partial order reductions and structural reductions become less efficient with the growing size of the CTL formula. The reason is that the more places and transitions are used as atomic propositions in a given formula, the more of the behaviour (interleaving) becomes relevant for the validity of the formula. We suggest several methods to reduce the size of CTL formulae, while preserving their validity. By these methods, we significantly increase the benefits of structural and partial order reductions, as the combination of our techniques can achive up to 60% average reduction in formulae sizes. The algorithms are implemented in the open-source verification tool TAPAAL and we document the efficiency of our approach on a large benchmark of Petri net models and queries from the Model Checking Contest 2017.

1 Introduction

Model checking [6] of distributed systems, described in high-level formalisms like Petri nets, is often a time and resource consuming task—attributed mainly to the state space explosion problem. Several techniques like partial order and symmetry reductions [16,20,21,23,24] and structural reductions [14,17,18] were suggested for reducing the size of the state space of a given Petri net in need of exploration to verify different logical specifications. These techniques try to prune the searchable state space and their efficiency is to a high degree influenced by the type and size of the logical formula in question. The larger the formula is and the more atomic propositions (querying the number of tokens in places or the fireability of certain transitions) it has, the less can be pruned away when exploring the state space and hence the effect of these techniques is reduced. It is therefore desirable to design techniques that can reduce the size of a given logical formula, while preserving the model checking answer. For practical applicability, it is important that such formula reduction techniques are computationally less demanding than the actual state space search.

In this paper, we focus on the well-known logic CTL [5] and describe three methods for CTL formula simplification, each preserving the logical equivalence

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 143–163, 2018.
https://doi.org/10.1007/978-3-319-91268-4_8

w.r.t. a the given Petri net model. The first two methods rely on standard logical equivalences of formulae, while the third one uses state equations of Petri nets and linear programming to recursively traverse the structure of a given CTL formula. During this process, we identify subformulae that are either trivially satisfied or impossible to satisfy, and we replace them with easier to verify alternatives. We provide an algorithm for performing such a formula simplification, including the traversal though temporal CTL operators, and prove the correctness of our approach.

The formula simplification methods are implemented and fully integrated into an open-source model checker TAPAAL [10] and its untimed verification engine `verifypn` [14]. We document the performance of our tool on the large benchmark of Petri net models and CTL queries from the Model Checking Contest 2017 (MCC'17) [15]. The data show that for CTL cardinality queries, we are able to achieve on average 60% of reduction of the query size and about 34% of queries are simplified into trivial queries *true* or *false*, hence avoiding completely the state space exploration. For CTL fireability queries, we achieved 50% reduction of the query size and about 10% of queries are simplified into *true* or *false*. Finally, we compare our simplification algorithm with the one implemented in the tool LoLA [26], the winner of MCC'17 in the several categories including the CTL category, documenting a noticeable performance margin in favour of our approach, both in the number of solved queries purely by the CTL simplification as well as when CTL verification follows the simplification process. For completeness, we also present the data for pure reachability queries where the tool Sara [25] (run parallel with LoLA during MCC'17) performs counterexample guided abstraction refinement and contributes to a high number (about twice as high as our tool) of solved reachability queries without the need to run LoLA's state space exploration. Nevertheless, if we also include the actual verification after the formula simplification, TAPAAL now moves 0.4% ahead of the combined performance of LoLA and Sara.

Related Work. Traditionally, the conditions generated by the state equation technique [17] express linear constraints on the number of times the events can occur relative to other events of the system, and form a necessary condition for marking reachability. State equations were used in [14] as an over-approximation technique for preprocessing of reachability formulae in earlier editions of the model checking contest. As the technique can be often inconclusive, extensions of state equations were studied e.g. in [11] where the authors use traps to increase precision of the method, or in [8] where the state equation technique is extended to liveness properties. State equations, as a necessary condition for reachability, were also used in other application domains like concurrent programming [1,2]. Our work further extends state equations to full CTL logic and improves the precision of the method by a recursive evaluation of integer linear programs for all subformulae, while employing state equations for each subformula and its negation. State equations were also exploited in [22] in order to guide the state space search based on a minimal solution to the equations. This approach is orthogonal with ours as it essentially defines a heuristic search strat-

egy that in the worst case must explore the whole state space. More recently, the state equation technique was also applied to the coverability problem for Petri nets [4,12].

Formula rewriting techniques (in order to reduce the size of CTL formulae) are implemented in the tool LoLA [26]. The tool performs formula simplification by employing subformula rewriting rules that include a subset of the rules described in Sect. 3. LoLA also employs the model checking tool Sara [25] that uses state equations in combination with Counter Example Abstraction Refinement (CEGAR) to perform an exact reachability analysis, being able to answer both reachability and non-reachability questions and hence it is close to being a complete model checker. Sara shows a very convincing performance on reachability queries, however, in the CTL category, we are able to simplify to *true* or *false* almost twice as many formulae, compared to the combined performance of Sara and LoLA.

2 Preliminaries

A *labelled transition system* (LTS) is a tuple $TS = (\mathcal{S}, A, \rightarrow)$ where $\mathcal{S}$ is a set of states, A is a set of actions (or labels), and $\rightarrow \subseteq \mathcal{S} \times A \times \mathcal{S}$ is a transition relation. We write $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \rightarrow$ and say that a is *enabled* in s. The set of all enabled actions in a state s is denoted $en(s)$. A state s is a *deadlock* if $en(s) = \emptyset$. We write $s \rightarrow s'$ whenever there is an action a such that $s \xrightarrow{a} s'$.

A *run* starting at s_0 is any finite or infinite sequence $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$ where $s_0, s_1, s_2, \ldots \in \mathcal{S}$, $a_0, a_1, a_2 \cdots \in A$ and $(s_i, a_i, s_{i+1}) \in \rightarrow$ for all respective i. We use $\Pi(s)$ to denote the set of all runs starting at the state s. A run is *maximal* if it is either infinite or ends in a state that is a deadlock. Let $\Pi^{max}(s)$ denote the set of all maximal runs starting at the state s. A *position* i in a run $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$ refers to the state s_i in the path and is written as π_i. If π is infinite then any i, $0 \leq i$, is a position in π. Otherwise $0 \leq i \leq n$ where s_n is the last state in π.

We now define the syntax and semantics of a *computation tree logic* (CTL) [7] as used in the Model Checking Contest [15]. Let AP be a set of atomic propositions. We evaluate atomic propositions on a given LTS $TS = (\mathcal{S}, A, \rightarrow)$ by the function $v : \mathcal{S} \rightarrow 2^{AP}$ so that $v(s)$ is the set of atomic propositions satisfied in the state $s \in \mathcal{S}$.

The CTL syntax is given as follows (where $\alpha \in AP$ ranges over atomic propositions):

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid \alpha \mid \mathit{deadlock} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A(\varphi_1 U \varphi_2) \mid E(\varphi_1 U \varphi_2).$$

We use Φ_{CTL} to denote the set of all CTL formulae. The semantics of a CTL formula φ in a state $s \in \mathcal{S}$ is given in Table 1. We do not use only the minimal

Table 1. Semantics of CTL formulae

$s \models \mathit{true}$	
$s \not\models \mathit{false}$	
$s \models \alpha$	iff $\alpha \in v(s)$
$s \models \mathit{deadlock}$	iff $en(s) = \emptyset$
$s \models \varphi_1 \wedge \varphi_2$	iff $s \models \varphi_1$ and $s \models \varphi_2$
$s \models \varphi_1 \vee \varphi_2$	iff $s \models \varphi_1$ or $s \models \varphi_2$
$s \models \neg\varphi$	iff $s \not\models \varphi$
$s \models AX\varphi$	iff $s' \models \varphi$ for all $s' \in \mathcal{S}$ s.t. $s \rightarrow s'$
$s \models EX\varphi$	iff there is $s' \in \mathcal{S}$ s.t.$s \rightarrow s'$ and $s' \models \varphi$
$s \models AF\varphi$	iff for all $\pi \in \Pi^{max}(s)$ there is a position i in π s.t. $\pi_i \models \varphi$
$s \models EF\varphi$	iff there is $\pi \in \Pi^{max}(s)$ and a position i in π s.t. $\pi_i \models \varphi$
$s \models AG\varphi$	iff for all $\pi \in \Pi^{max}(s)$ and for all positions i in π we have $\pi_i \models \varphi$
$s \models EG\varphi$	iff there is $\pi \in \Pi^{max}(s)$ s.t. for all positions i in π we have $\pi_i \models \varphi$
$s \models A(\varphi_1 U \varphi_2)$	iff for all $\pi \in \Pi^{max}(s)$ there is a position i in π s.t.
	$\pi_i \models \varphi_2$ and for all j, $0 \leq j < i$, we have $\pi_j \models \varphi_1$
$s \models E(\varphi_1 U \varphi_2)$	iff there is $\pi \in \Pi^{max}(s)$ and there is a position i in π s.t.
	$\pi_i \models \varphi_2$ and for all j, $0 \leq j < i$, we have $\pi_j \models \varphi_1$

set of CTL operators because the query simplification tries to push the negation as far as possible to the atomic predicates. This significantly improves the performance of our on-the-fly CTL model checking algorithm and allows for a more refined query rewriting.

We can now define weighted Petri nets with inhibitor arcs. Let $\mathbb{N}^0 = \mathbb{N} \cup \{0\}$ be the set of natural numbers including 0 and let $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$ be the set of natural numbers including infinity.

Definition 1 (Petri net). *A Petri net is a tuple $N = (P, T, W, I)$ where P and T are finite disjoint sets of places and transitions, $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}^0$ is the weight function for regular arcs, and $I : (P \times T) \rightarrow \mathbb{N}^\infty$ is the weight function for inhibitor arcs.*

A *marking* M on N is a function $M : P \rightarrow \mathbb{N}^0$ where $M(p)$ denotes the number of tokens in the place p. The set of all markings of a Petri net N is written as $\mathcal{M}(N)$. Let $M_0 \in \mathcal{M}(N)$ be a given *initial marking* of N.

A Petri net $N = (P, T, W, I)$ defines an LTS $TS(N) = (\mathcal{S}, A, \rightarrow)$ where $\mathcal{S} = \mathcal{M}(N)$ is the set of all markings, $A = T$ is the set of labels, and $M \xrightarrow{t} M'$ whenever for all $p \in P$ we have $M(p) < I((p,t))$ and $M(p) \geq W((p,t))$ such that $M'(p) = M(p) - W((p,t)) + W((t,p))$. We inductively extend the relation $\xrightarrow{t}$ to sequences of transitions $w \in T^*$ such that $M \xrightarrow{\epsilon} M$ and $M \xrightarrow{wt} M'$ if $M \xrightarrow{w} M''$ and $M'' \xrightarrow{t} M'$. We write $M \rightarrow^* M'$ if there is $w \in T^*$ such that $M \xrightarrow{w} M'$. By $\mathit{reach}(M) = \{M' \in \mathcal{M}(N) \mid M \rightarrow^* M'\}$ we denote the set of all markings reachable from M.

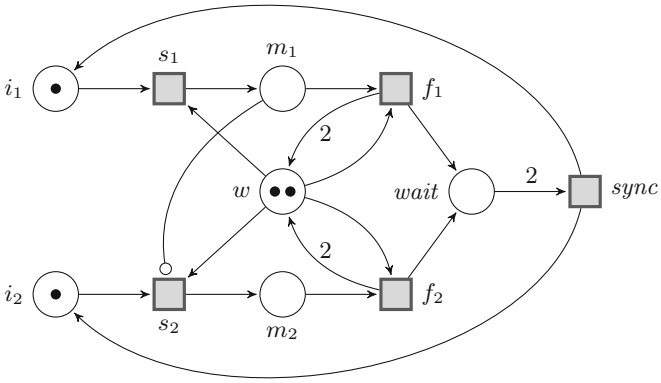


Fig. 1. A Petri net modelling two synchronizing processes

Example 1. Figure 1 illustrates an example of a Petri net where places are drawn as circles, transitions as rectangles, regular arcs as arrows with the weight as labels (default weight is 1 and arcs with weight 0 are not depicted) and inhibitor arcs are shown as circle-headed arrows (again the default weight is 1 and arcs with weight ∞ are not depicted). The dots inside places represent the number of tokens (marking). The initial marking in the net can be written by $i_1 i_3 2w$ denoting one token in i_1, one token in i_3 and two tokens in the place w. The net attempts to model two processes that aim to get exclusive access to firing either the transition f_1 or f_2 (making sure that they cannot be enabled concurrently). Once the first process decides to enable transition f_1 by moving the token from i_1 to m_1, the second process is not allowed to place a token into m_2 due to the inhibitor arc connection m_1 to s_2. However, as there is no inhibitor arc in the order direction, it is possible to reach a deadlock in the net by performing $i_1 i_2 2w \xrightarrow{s_2} i_1 m_2 w \xrightarrow{s_1} m_1 m_2$.

Finally, we fix the set of atomic propositions α ($\alpha \in AP$) for Petri nets as used in the MCC Property Language [15]:
$\alpha ::= t \mid e_1 \bowtie e_2$
$e ::= c \mid p \mid e_1 \oplus e_2$
where $t \in T$, $c \in \mathbb{N}^0$, $\bowtie \in \{<, \leq, =, \neq, >, \geq\}$, $p \in P$, and $\oplus \in \{+, -, *\}$. The evaluation function v for a marking M is given as $v(M) = \{t \in T \mid t \in en(M)\} \cup \{e_1 \bowtie e_2 \mid eval_M(e_1) \bowtie eval_M(e_2)\}$ where $eval_M(c) = c, eval_M(p) = M(p)$ and $eval_M(e_1 \oplus e_2) = eval_M(e_1) \oplus eval_M(e_2)$.

Formulae that do not use any atomic predicate t for transition firing and *deadlock* are called *CTL cardinality formulae* and formulae that avoid the use of $e_1 \bowtie e_2$ and *deadlock* are called *CTL firability formulate*. Formulae of the form $EF\varphi$ or $AG\varphi$ where φ does not contain any other temporal operator are called *reachability formulae*, and as for CTL can be subdivided into the *reachability cardinality* and *reachability fireability* category.

Example 2. Consider the Petri net in Fig. 1 and the reachability fireability formula $EF(f_1 \wedge f_2)$ asking whether there is a reachable marking that enables both f_1 and f_2. By exploring the (finite) part of the LTS reachable from the initial marking $i_1 i_2 2w$, we can conclude that $i_1 i_2 2w \not\models EF(f_1 \wedge f_2)$. However, the slightly modified query $EFAX(f_1 \wedge f_2)$ holds in the initial marking as a deadlocked marking $m_1 m_2$ can be reached, and due to the definition of the universal next modality we have $m_1 m_2 \models AX(f_1 \wedge f_2)$. Another example of a cardinality formula is $E(w \geq 2 \ U \ m_2 = 1)$ asking if there is a computation that marks the place m_2 and before it happens, w must contain at least two tokens. This formula holds in the initial marking by firing the transition s_2.

We shall finish the preliminaries by recalling the basics of linear programming. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables and let $\overline{x} = (x_1, x_2, \ldots, x_n)^T$ be a column vector of the variables. A *linear equation* is of the form $\overline{c} \cdot \overline{x} \bowtie k$ where $\bowtie \in \{=, <, \leq, >, \geq\}$, $k \in \mathbb{Z}$ is an integer, and $\overline{c} = (c_1, c_2, \ldots, c_n)$ is a row vector of integer constants. An *integer linear program LP* is a finite set of linear equations. An (integer) *solution* to LP is a mapping $u : X \to \mathbb{N}^0$ from variables to natural numbers such that for every linear equation $(\overline{c} \cdot \overline{x} \bowtie k) \in LP$, the column vector $\overline{u} = (u(x_1)\ u(x_2) \cdots u(x_n))^T$ satisfies the equation $\overline{c} \cdot \overline{u} \bowtie k$. We use $\mathcal{E}^X_{lin}$ to denote the set of all integer linear programs over the variables X.

An integer linear program with a solution is said to be *feasible*. For our purpose, we only consider feasibility and we are not interested in the optimality of the solution. The feasibility problem of integer linear programs is NP-complete [11,19], however, there exists a number of efficient linear program solvers (we use `lp_solve` in our implementation [3]).

3 Logical Equivalence of Formulae

Before we give our method for recursive simplification of CTL formulae via the use of state equations in Sect. 4, we first introduce two other formula simplification techniques. The first method utilizes the initial marking and the second method uses universally valid formulae equivalences. For the rest of this section, we assume a fixed Petri net $N = (P, T, W, I)$ with the initial marking M_0.

For the first simplification, let us define in Table 2 the function $\Omega : \Phi_{CTL} \to \{true, false, ?\}$ that checks if a given formula is trivially satisfiable in the initial marking M_0. Note that we generalize the binary conjunctions and disjunctions to n-ary operations as it corresponds to the implementation in our tool. The correctness of this simplification is expressed in the following theorem.

Theorem 1 (Initial Rewrite). *Let φ be a CTL formula such that $\Omega(\varphi) \neq ?$. Then $M_0 \models \varphi$ if and only if $\Omega(\varphi) = true$.*

For the second simplification, we establish a recursively defined rewrite-function $\rho : \Phi_{CTL} \to \Phi_{CTL}$ given in Table 3 that is based on logical equivalences for the CTL quantifiers. In the definition of ρ, we assume that the n-ary operators $\vee$ and $\wedge$ are associative and commutative. The correctness is captured in the following theorem.

Table 2. Simplification rules for a given initial marking M_0

$$\Omega(\mathit{true}) = \mathit{true} \qquad \Omega(\mathit{false}) = \mathit{false}$$

$$\Omega(\alpha) = M_0 \models \alpha \qquad \Omega(\mathit{deadlock}) = M_0 \models \mathit{deadlock}$$

$$\Omega(AX\varphi) = \begin{cases} \mathit{true} & \text{if } M_0 \models \mathit{deadlock} \\ ? & \text{otherwise} \end{cases} \qquad \Omega(EX\varphi) = \begin{cases} \mathit{false} & \text{if } M_0 \models \mathit{deadlock} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(\neg\varphi) = \begin{cases} \mathit{true} & \text{if } \Omega(\varphi) = \mathit{false} \\ \mathit{false} & \text{if } \Omega(\varphi) = \mathit{true} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(\varphi_1 \wedge \cdots \wedge \varphi_n) = \begin{cases} \mathit{true} & \text{if for all } i,\ 1 \le i \le n, \text{ we have } \Omega(\varphi_i) = \mathit{true} \\ \mathit{false} & \text{if there exists } i,\ 1 \le i \le n, \text{ s.t. } \Omega(\varphi_i) = \mathit{false} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(\varphi_1 \vee \cdots \vee \varphi_n) = \begin{cases} \mathit{true} & \text{if there exists } i,\ 1 \le i \le n, \text{ s.t. } \Omega(\varphi_i) = \mathit{true} \\ \mathit{false} & \text{if for all } i,\ 1 \le i \le n, \text{ we have } \Omega(\varphi_i) = \mathit{false} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(EG\varphi) = \Omega(AG\varphi) = \begin{cases} \mathit{false} & \text{if } \Omega(\varphi) = \mathit{false} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(EF\varphi) = \Omega(AF\varphi) = \begin{cases} \mathit{true} & \text{if } \Omega(\varphi) = \mathit{true} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(E(\varphi_1 U \varphi_2)) = \Omega(A(\varphi_1 U \varphi_2)) = \begin{cases} \mathit{true} & \text{if } \Omega(\varphi_2) = \mathit{true} \\ \mathit{false} & \text{if } \Omega(\varphi_1) = \Omega(\varphi_2) = \mathit{false} \\ ? & \text{otherwise} \end{cases}$$

Theorem 2 (Equivalence Rewriting). *Let $M \in \mathcal{M}(N)$ be a marking on N. Then $M \models \varphi$ if and only if $M \models \rho(\varphi)$.*

4 Formula Simplification via State Equations

We will now describe the main ingredients of our formula simplification algorithm. It is based on a recursive decent on the structure of the formula, checking whether its subformulae and their negations can possibly hold in some reachable marking (here we use the state equation [11,17] approach) and then propagating back this information through the Boolean and temporal operators.

We use state equations to identify universally true or false subformulae, similarly as e.g. in [14]. The main novelty is that we extend the approach to deal with arbitrary arithmetical expressions and repeatedly solve linear programs for subformulae of the given property so that more significant simplifications can

Table 3. Equivalence rewriting of CTL formulae

$$\rho(\alpha) = \alpha \qquad \rho(\mathit{deadlock}) = \mathit{deadlock}$$

$$\rho(EG\varphi) = \rho(\neg AF\rho(\neg\varphi)) \qquad \rho(AG\varphi) = \rho(\neg EF\rho(\neg\varphi))$$

$$\rho(EX\varphi) = EX\rho(\varphi) \qquad \rho(AX\varphi) = AX\rho(\varphi)$$

$$\rho(\varphi_1 \wedge \cdots \wedge \varphi_n) = \rho(\varphi_1) \wedge \cdots \wedge \rho(\varphi_n) \qquad \rho(\varphi_1 \vee \cdots \vee \varphi_n) = \rho(\varphi_1) \vee \cdots \vee \rho(\varphi_n)$$

$$\rho(\neg\varphi) = \begin{cases} \varphi' & \text{if } \rho(\varphi) = \neg\varphi' \\ AX\rho(\neg\varphi') & \text{if } \rho(\varphi) = EX\varphi' \\ EX\rho(\neg\varphi') & \text{if } \rho(\varphi) = AX\varphi' \\ \rho((\neg\varphi_1) \wedge \cdots \wedge (\neg\varphi_n)) & \text{if } \varphi = \varphi_1 \vee \cdots \vee \varphi_n \\ \rho((\neg\varphi_1) \vee \cdots \vee (\neg\varphi_n)) & \text{if } \varphi = \varphi_1 \wedge \cdots \wedge \varphi_n \\ \neg\rho(\varphi) & \text{otherwise} \end{cases}$$

$$\rho(EF\varphi) = \begin{cases} \neg\mathit{deadlock} & \text{if } \rho(\varphi) = \neg\mathit{deadlock} \\ EF\varphi' & \text{if } \rho(\varphi) = EF\varphi' \\ \rho(EF\varphi') & \text{if } \rho(\varphi) = AF\varphi' \\ \rho(EF\varphi_2) & \text{if } \rho(\varphi) = E(\varphi_1 U \varphi_2) \\ \rho(EF\varphi_2) & \text{if } \rho(\varphi) = A(\varphi_1 U \varphi_2) \\ \rho(EF\varphi_1 \vee \cdots \vee EF\varphi_n) & \text{if } \rho(\varphi) = \varphi_1 \vee \cdots \vee \varphi_n \\ EF\rho(\varphi) & \text{otherwise} \end{cases}$$

$$\rho(AF\varphi) = \begin{cases} \neg\mathit{deadlock} & \text{if } \rho(\varphi) = \neg\mathit{deadlock} \\ EF\varphi' & \text{if } \rho(\varphi) = EF\varphi' \\ AF\varphi' & \text{if } \rho(\varphi) = AF\varphi' \\ \rho(AF\varphi_2) & \text{if } \rho(\varphi) = A(\varphi_1 U \varphi_2) \\ \rho((EF\varphi_2) \vee (AF\varphi_1)) & \text{if } \rho(\varphi) = \varphi_1 \vee EF\varphi_2 \\ AF\rho(\varphi) & \text{otherwise} \end{cases}$$

$$\rho(A(\varphi_1 U \varphi_2)) = \begin{cases} \neg\mathit{deadlock} & \text{if } \rho(\varphi_2) = \neg\mathit{deadlock} \\ \rho(\varphi_2) & \text{if } \rho(\varphi_1) = \mathit{deadlock} \\ \rho(AF\varphi_2) & \text{if } \rho(\varphi_1) = \neg\mathit{deadlock} \\ EF\varphi_3 & \text{if } \rho(\varphi_2) = EF\varphi_3 \\ AF\varphi_3 & \text{if } \rho(\varphi_2) = AF\varphi_3 \\ \rho((EF\varphi_4) \vee A(\varphi_1 U \varphi_3)) & \text{if } \rho(\varphi_2) = \varphi_3 \vee EF\varphi_4 \\ A(\rho(\varphi_1) U \rho(\varphi_2)) & \text{otherwise} \end{cases}$$

$$\rho(E(\varphi_1 U \varphi_2)) = \begin{cases} \neg\mathit{deadlock} & \text{if } \rho(\varphi_2) = \neg\mathit{deadlock} \\ \rho(\varphi_2) & \text{if } \rho(\varphi_1) = \mathit{deadlock} \\ \rho(EF\varphi_2) & \text{if } \rho(\varphi_1) = \neg\mathit{deadlock} \\ EF\varphi_3 & \text{if } \rho(\varphi_2) = EF\varphi_3 \\ \rho((EF\varphi_4) \vee E(\varphi_1 U \varphi_3)) & \text{if } \rho(\varphi_2) = \varphi_3 \vee EF\varphi_4 \\ E(\rho(\varphi_1) U \rho(\varphi_2)) & \text{otherwise} \end{cases}$$

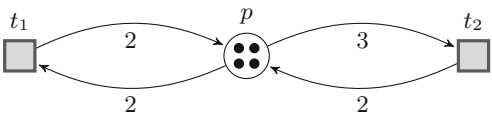


Fig. 2. Example Petri net and initial marking for formula simplification

be achieved (we try to solve the state equations both for the subformula and its negation). As a result, we can simplify more formulae into the trivially valid ones (*true*) or invalid ones (*false*) or we can significantly reduce the size of the formulae which can then speed up the state space exploration.

Consider the Petri net in Fig. 2 with the initial marking M_0, where $M_0(p) = 4$. The state equation for the reachability formula $EF\ p \geq 5$ (can the place p be marked with at least five tokens) over the variables x_{t_1} and x_{t_2} (representing the number of transition firings of t_1 and t_2 respectively) looks as

$$M_0(p) + \sum_{t \in T} (W(t,p) - W(p,t)) x_t \geq 5$$

which in our example translates to $4 + 0 \cdot x_{t_1} - 1 \cdot x_{t_2} \geq 5$. The inequality clearly does not have a solution in nonnegative integers, hence we can conclude without exploring the state space that $EF\ p \geq 5$ does not hold in the initial marking. Moreover, consider now the formula $EF\ (p \geq 5) \vee (p = 2 \wedge p \leq 7)$. By recursively analyzing the subformulae, we can conclude using the state equations that $p \geq 5$ cannot be satisfied in any reachable marking, hence the formula simplifies to $EF\ (p = 2 \wedge p \leq 7)$. Moreover, by continuing the recursive decent and looking at the subformula $p \leq 7$, we can determine by using state equations, that its negation $p > 7$ cannot be satisfied in any reachable marking. Hence $p \leq 7$ is universally true and the formula further simplifies to an equivalent formula $EF\ p = 2$ for which we have to apply conventional verification techniques.

In what follows, we formally define our formula simplification procedure and extend it to the full CTL logic so that e.g. the formula $EF\ AX\ p \geq 5$ simplifies to the reachability formula $EF\ deadlock$ for which we can use specialized algorithms for deadlock detection (e.g. using the siphon-trap property [13]) instead of the more expensive CTL verification algorithms. Even if a CTL formula does not simplify to a pure reachability property, the reduction in the size of the CTL formula has still a positive effect on the efficiency of the CTL verification algorithms as the state space grows with the number of different subformulae.

4.1 Simplification Procedure

Let $N = (P, T, W, I)$ be a fixed Petri net with the initial marking M_0 and φ a given CTL formula. Before we start, we assume that the formula φ has been rewritten into an equivalent one by recursively applying the rewriting rules in Table 4. Clearly, these rules preserve logical equivalence and they push the negation down to either the atomic propositions or in front of the existential or

Table 4. Rewriting rules

φ	Rewritten φ
t	$p_1 \geq W(p_1,t) \wedge \cdots \wedge p_n \geq W(p_n,t) \wedge$ $p_1 < I(p_1,t) \wedge \cdots \wedge p_n < I(p_n,t)$ where $P = \{p_1, p_2, \ldots, p_n\}$
$e_1 \neq e_2$	$e_1 > e_2 \vee e_1 < e_2$
$e_1 = e_2$	$e_1 \leq e_2 \wedge e_1 \geq e_2$
$\neg(\varphi_1 \wedge \varphi_2)$	$\neg\varphi_1 \vee \neg\varphi_2$
$\neg(\varphi_1 \vee \varphi_2)$	$\neg\varphi_1 \wedge \neg\varphi_2$
$\neg AX\varphi$	$EX\neg\varphi$
$\neg EX\varphi$	$AX\neg\varphi$
$\neg AF\varphi$	$EG\neg\varphi$
$\neg EF\varphi$	$AG\neg\varphi$
$\neg AG\varphi$	$EF\neg\varphi$
$\neg EG\varphi$	$AF\neg\varphi$

universal until operators. Moreover, the fireability predicate for a transition t is rewritten to the equivalent cardinality formula.

Let $\mathcal{E}^X_{lin}$ be the set of all integer linear programs over the set of variables $X = \{x_t \mid t \in T\}$. Let $LPS \subseteq \mathcal{E}^X_{lin}$ be a finite set of integer linear programs. We say that LPS has a solution, if there exists a linear program $LP \in LPS$ that has a solution.

We will now define a simplification function that, for a given formula $\varphi \in \Phi_{CTL}$, produces a simplified formula and two sets of integer linear programs. The function is of the form

$$simplify : \Phi_{CTL} \to \Phi_{CTL} \times 2^{\mathcal{E}^X_{lin}} \times 2^{\mathcal{E}^X_{lin}}$$

and we write $simplify(\varphi) = (\varphi', LPS, \overline{LPS})$ when the formula φ is simplified to an equivalent formula φ', and where the following invariant holds:

- if $M \models \varphi$ for some M reachable from M_0 then LPS has a solution, and
- if $M \not\models \varphi$ for some M reachable from M_0 then $\overline{LPS}$ has a solution.

In order to define the simplification function, we use the function $merge : 2^{\mathcal{E}^X_{lin}} \times 2^{\mathcal{E}^X_{lin}} \to 2^{\mathcal{E}^X_{lin}}$ that combines two set of integer linear programs and is defined as $merge\ (LPS_1, LPS_2) = \{LP_1 \cup LP_2 \mid LP_1 \in LPS_1,\ LP_2 \in LPS_2\}$. Finally, let $BASE$ denote the integer linear program with the following equations

$$M_0(p) + \sum_{t \in T} (W(t,p) - W(p,t)) \cdot x_t \geq 0 \quad \text{for all } p \in P$$

that ensures that any solution to $BASE$ must leave a nonnegative number of tokens in every place of N.

Algorithm 1. Simplify $e_1 \bowtie e_2$

Function *simplify*$(e_1 \bowtie e_2)$
  if e_1 *is not linear or* e_2 *is not linear* **then**
    return $(e_1 \bowtie e_2, \{\{0 \leq 1\}\}, \{\{0 \leq 1\}\})$
  $LPS \leftarrow \{\{const(e_1) \bowtie const(e_2)\}\}$
  $\overline{LPS} \leftarrow \{\{const(e_1) \overline{\bowtie} const(e_2)\}\}$
  if $\{LP \cup BASE \mid LP \in LPS\}$ *has no solution* **then**
    return *simplify*$(false)$
  else if $\{LP \cup BASE \mid LP \in \overline{LPS}\}$ *has no solution* **then**
    return *simplify*$(true)$
  else
    return $(e_1 \bowtie e_2, LPS, \overline{LPS})$

Algorithm 2. Simplify $\neg\varphi$

Function *simplify*$(\neg\varphi)$
  $(\varphi', LPS, \overline{LPS}) \leftarrow$ *simplify*(φ)
  if $\varphi' = true$ **then return** *simplify*$(false)$
  if $\varphi' = false$ **then return** *simplify*$(true)$
  return $(\neg\varphi', \overline{LPS}, LPS)$

First, we postulate *simplify*$(true) = (true, \{\{0 \leq 1\}\}, \emptyset)$, *simplify*$(false) = (false, \emptyset, \{\{0 \leq 1\}\})$, and *simplify*$(deadlock) = (deadlock, \{\{0 \leq 1\}\}, \{\{0 \leq 1\}\})$ and these definitions clearly satisfy our invariant.

Algorithm 1 describes how to simplify the atomic predicates, where the function *const* takes as input an arithmetic expression e and returns one side of the linear equation as follows:

$$
\begin{aligned}
const(c) &= c \\
const(p) &= M_0(p) + \sum_{t \in T} (W(t,p) - W(p,t)) \cdot x_t \\
const(e_1 + e_2) &= const(e_1) + const(e_2) \\
const(e_1 - e_2) &= const(e_1) - const(e_2) \\
const(e_1 \cdot e_2) &= const(e_1) \cdot const(e_2).
\end{aligned}
$$

In the algorithm we let $\overline{\bowtie}$ denote the dual operation to $\bowtie$, for example $\overline{>}$ becomes $\leq$ and $\overline{\geq}$ becomes $<$. There is a special case that we must handle here. If in either of the expressions e_1 or e_2 we have a multiplication that includes more than one place (i.e. the expression is not linear) then we would return a nonlinear program that cannot be solved by linear program solvers. To handle this situation, if either side of the comparison in nonlinear, we return the formula unchanged and two singleton sets of linear programs $\{\{0 \leq 1\}\}$ that trivially have a solution (any

Algorithm 3. Simplify $\varphi_1 \Diamond \ldots \Diamond \varphi_n$ for $\Diamond \in \{\wedge, \vee\}$

Function *simplify*$(\varphi_1 \Diamond \ldots \Diamond \varphi_n)$
  Let φ' be an empty formula.
  if $\Diamond = \wedge$ **then** $LPS \leftarrow \{\{0 \leq 1\}\}$; $\overline{LPS} \leftarrow \emptyset$
  if $\Diamond = \vee$ **then** $LPS \leftarrow \emptyset$; $\overline{LPS} \leftarrow \{\{0 \leq 1\}\}$
  for $i := 1$ *to* n **do**
    $(\varphi'_i, LPS_i, \overline{LPS}_i) \leftarrow$ *simplify*(φ_i)
    if $\Diamond = \wedge$ *and* $\varphi'_i =$ *false* **then return** *simplify*(*false*)
    if $\Diamond = \wedge$ *and* $\varphi'_i \neq$ *true* **then**
      $\varphi' \leftarrow \varphi' \wedge \varphi'_i$
      $LPS \leftarrow$ *merge*(LPS, LPS_i)
      $\overline{LPS} \leftarrow \overline{LPS} \cup \overline{LPS}_i$
    if $\Diamond = \vee$ *and* $\varphi'_i =$ *true* **then return** *simplify*(*true*)
    if $\Diamond = \vee$ *and* $\varphi'_i \neq$ *false* **then**
      $\varphi' \leftarrow \varphi' \vee \varphi'_i$
      $LPS \leftarrow LPS \cup LPS_i$
      $\overline{LPS} \leftarrow$ *merge*$(\overline{LPS}, \overline{LPS}_i)$
  if φ' *is empty formula and* $\Diamond = \wedge$ **then return** *simplify*(*true*)
  if φ' *is empty formula and* $\Diamond = \vee$ **then return** *simplify*(*false*)
  if $\Diamond = \wedge$ *and* $\{LP \cup BASE \mid LP \in LPS\}$ *has no solution* **then**
    return *simplify*(*false*)
  if $\Diamond = \vee$ *and* $\{LP \cup BASE \mid LP \in \overline{LPS}\}$ *has no solution* **then**
    return *simplify*(*true*)
  return $(\varphi', LPS, \overline{LPS})$

variable assignment is a solution to the linear program $0 \leq 1$) and hence satisfy our invariant.

The simplification of negation $\neg\varphi$ is given in Algorithm 2. It first recursively computes the simplification φ' of φ and if the answer is conclusive then the negated conclusive answer is returned, otherwise we return $\neg\varphi'$ and swap the two sets of linear programs.

In Algorithm 3 we show how to simplify conjunctions and disjunctions of formulae. We give the simplification function for n-ary operators to mimic the implementation closely. We present both conjunction and disjunction in the same pseudocode in order to clarify the symmetry in handling the Boolean connectives. The algorithm recursively simplifies the subformulae and one by one adds the simplified formulae into the resulting proposition φ', unless a conclusive answer (*true*/*false*) can be given immediately or the subformula can be omitted. Note that for conjunction we merge the current LPS and LPS_i returned for the subformula φ_i as if the conjunction is satisfied in some reachable marking then there must be an $LP \in LPS$ and an $LP_i \in LPS_i$ such that $LP \cup LP_i$ has a solution. Symmetrically, we do the merge also for disjunction and the negated sets of linear programs. Finally, we check whether the created systems of linear

Algorithm 4. Simplify $QX\varphi$, where $Q \in \{A, E\}$

```
1 Function simplify(QXφ)
2     (φ', LPS, LPS̄) ← simplify(φ)
3     if Q = A and φ' = true then return simplify(true)
4     if Q = A and φ' = false then return simplify(deadlock)
5     if Q = E and φ' = true then return simplify(¬deadlock)
6     if Q = E and φ' = false then return simplify(false)
7     return (QXφ', {{0 ≤ 1}}, {{0 ≤ 1}})
```

Algorithm 5. Simplify $QP\varphi$, where $QP \in \{AG, EG, AF, EF\}$

```
1 Function simplify(QPφ)
2     (φ', LPS, LPS̄) ← simplify(φ)
3     if φ' = true then return simplify(true)
4     if φ' = false then return simplify(false)
5     return (QPφ', {{0 ≤ 1}}, {{0 ≤ 1}})
```

programs have solutions and in the negative cases we can sometimes draw a conclusive answer.

Simplification of the next operators is given in Algorithm 4. It is worth noticing that for certain situations, the next operator can be removed and replaced with the deadlock proposition (and hence possibly change the CTL formula into a reachability formula). If none of the simplification cases applies, we return the next operator with the simplified formula together with two sets of linear programs with trivial solutions in order to satisfy our invariant. Similarly, the simplification of the unary CTL temporal operators is given in Algorithm 5.

Finally, in Algorithm 6 we present the simplification of binary CTL temporal operators. Here we first simplify φ_2 and see if we can draw some straightforward conclusions. If this is not the case, we also simplify φ_1 and if it evaluates to *true* or *false*, we can either reduce the binary temporal operator into a unary one or completely remove the unary operator, respectively.

Example 3. Consider again the net from Example 2. We can simplify the formula $EFAX(f_1 \wedge f_2)$ as follows. Let $X = \{x_{s_1}, x_{s_2}, x_{f_1}, x_{f_2}, x_{sync}\}$ be the variables. Using the rewriting rules from Table 4 we have that $EFAX(f_1 \wedge f_2)$ is equivalent to $EFAX(m_1 \geq 1 \wedge w \geq 1 \wedge m_2 \geq 1)$. The linear equations LPS generated by Algorithms 1 and 3 are as follows.

$$\begin{aligned} x_{s_1} - x_{f_1} &\geq 1 \\ 2 + x_{f_1} + x_{f_2} - x_{s_1} - x_{s_2} &\geq 1 \\ x_{s_2} - x_{f_2} &\geq 1 \end{aligned}$$

We do not include $BASE$ here, as the equations above are already unfeasible (have no integer solution). This follows from the observation that the first and

Algorithm 6. Simplify $Q(\varphi_1 U \varphi_2)$, where $Q \in \{A, E\}$

Function *simplify*$(Q(\varphi_1 U \varphi_2))$
  $(\varphi_2', LPS_2, \overline{LPS_2}) \leftarrow$ *simplify*(φ_2)
  if $\varphi_2' =$ *true* **then return** *simplify*$(true)$
  if $\varphi_2' =$ *false* **then return** *simplify*$(false)$
  $(\varphi_1', LPS_1, \overline{LPS_1}) \leftarrow$ *simplify*(φ_1)
  if $\varphi_1' =$ *true* **then return** $(QF\varphi_2', \{\{0 \leq 1\}\}, \{\{0 \leq 1\}\})$
  if $\varphi_1' =$ *false* **then return** $(\varphi_2', LPS_2, \overline{LPS_2})$
  return $(Q(\varphi_1' U \varphi_2'), \{\{0 \leq 1\}\}, \{\{0 \leq 1\}\})$

third equation imply that $x_{s_1} > x_{f_1}$ and $x_{s_2} > x_{f_2}$, respectively, and this contradicts the second equation $2 + x_{f_1} + x_{f_2} > x_{s_1} + x_{s_2}$. Therefore, Algorithm 3 simplifies $EFAX(f_1 \wedge f_2)$ to $EFAXfalse$ and by Algorithm 4, we simplify it further to $EFdeadlock$. No further reduction is possible, however, we simplified a CTL formula into a simple reachability formula for which we can now use specialized algorithms for deadlock detection.

We conclude this section with a theorem stating the correctness of the simplification, meaning that for $simplify(\varphi) = (\varphi', LPS, \overline{LPS})$ we have $M_0 \models \varphi$ if and only if $M_0 \models \varphi'$. In order to do so, we prove a stronger claim that allows us to formally introduce the invariant on the sets of linear programs returned by the function *simplify*.

Theorem 3 (Formula Simplification Correctness). *Let $N = (P, T, W, I)$ be a Petri net, M_0 an initial marking on N, and $\varphi \in \Phi_{CTL}$ a CTL formula. Let $simplify(\varphi) = (\varphi', LPS, \overline{LPS})$. Then for all markings $M \in \mathcal{M}(N)$ such that $M_0 \xrightarrow{w} M$ holds:*

1. *$M \models \varphi$ iff $M \models \varphi'$*
2. *if $M \models \varphi$ then there is $LP \in LPS$ such that $\wp(w)$ is a solution to LP*
3. *if $M \not\models \varphi$ then there is $LP \in \overline{LPS}$ such that $\wp(w)$ is a solution to LP*

where $\wp(w)$ is a solution that assigns to each variable x_t the number of occurrences of the transition t in the transition sequence w.

5 Implementation and Experiments

The formula simplification techniques are implemented in C++ in the `verifypn` engine [14] of the tool TAPAAL [10] and distributed in the latest release at www.tapaal.net. The source code is available at code.launchpad.net/verifypn.

After parsing the PNML model and the formula, TAPAAL applies sequentially the simplification procedures as depicted in Fig. 3, where we first attempt to restructure the formulae to a simpler form using ρ followed by the application of Ω. After this, the main *simplify* procedure is called. The simplification can

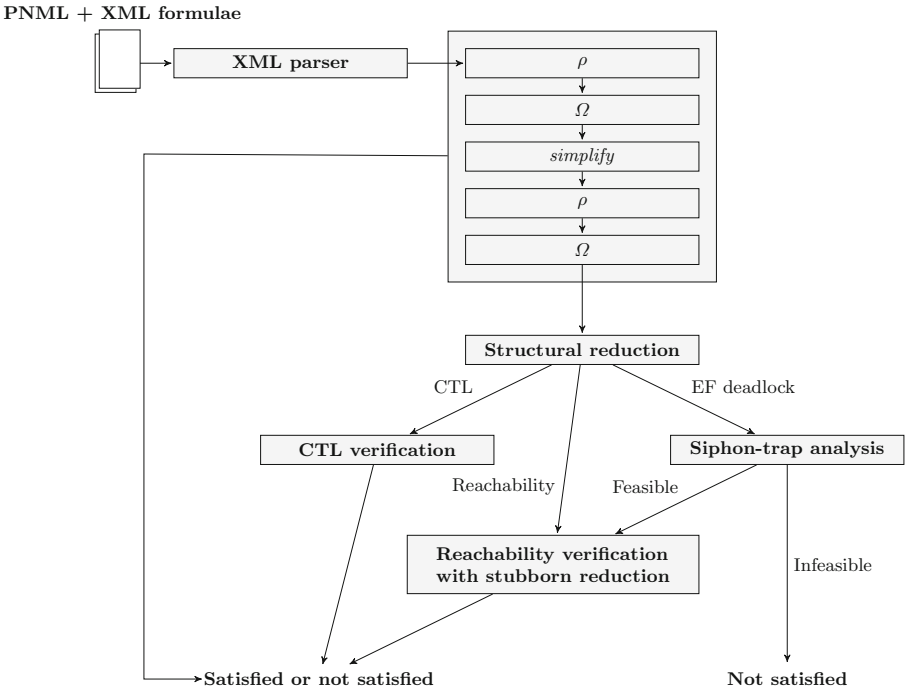


Fig. 3. TAPAAL tool-chain and control flow

create a formula where additional applications of ρ and Ω are possible and can further reduce the formula size. After the simplification is completed, TAPAAL applies structural reductions to the model, removing or merging redundant transitions and places as described in [14]. The engine now proceeds as follows.

1. If the formulae is of the form *EF deadlock* then siphon-trap analysis is attempted, followed by normal explicit-state verification in case of an inconclusive answer.
2. If the formulae falls within pure reachability category ($EF\varphi$ or $\neg EF\varphi$, where φ does not contain further temporal operators), then we call a specialized reachability engine that uses stubborn set reduction.
3. For the general CTL formula, the verification is performed via a translation to a dependency graph and performing on-the-fly computation of its minimum fixed-point assignment as described in [9].

5.1 Implementation Details of the Simplification Procedure

During implementation and subsequent experimentation, we discovered that the construction of linear programs for large models can be both time and memory-consuming. In particular, the *merge*-operation causes a quadratic blowup both in the size and the number of linear programs. To remedy this, we have implemented

Table 5. Formula simplification for CTL cardinality and fireability

CTL cardinality					
Algorithm	Solved	% Solved	Reachability	% Reachability	% Reduction
Ω	117	2.3	1834	36.6	27.2
ρ	7	0.1	1437	28.7	24.1
simplify	1437	28.7	2425	48.4	45.7
all	**1724**	**34.4**	**2993**	**59.8**	**60.3**
CTL fireability					
Ω	194	3.9	1701	34.0	27.1
ρ	0	0.0	1319	26.3	30.0
simplify	255	5.1	1422	28.4	11.0
all	**495**	**9.9**	**2022**	**40.4**	**49.7**

a "lazy" construction of the linear programs—similar to lazy evaluation known from functional programming languages. Instead of computing the full set of linear programs up front, we simply remember the basic linear programs and the tree of operations making up the merged or unioned linear program. Using this construction, we then extract a single linear program on demand, and thus avoid the up-front time and memory overhead of computing the merge and union operations at the call time.

5.2 Experimental Setup

To evaluate the performance of our approach, we conduct two series of experiments on the models and formulae from MCC'17 [15]. First, we investigate the effect of the three different simplification methods proposed in this paper along with their combination as depicted in Fig. 3. In the second experiment we compare the performance of our simplification algorithms to those used by LoLA, the winner of MCC'17. We also conduct a full run of the verification engines after the formula simplification in order to assess the impact of the simplification on the state space search. All experiments were run on AMD Opteron 6376 Processors, restricted to 14 GB of memory on 313 P/T nets from the MCC'17 benchmark. Each category contains 16 different queries which yields a total of 5008 executions for a given category.

5.3 Evaluation of Formula Simplification Techniques

We compare the performance of Ω, ρ and *simplify* functions along with their combined version referred to as *all* (applying sequentially ρ, Ω, *simplify*, ρ and Ω). The execution of each simplification was limited to 20 minutes per formula (excluding the model parsing time) and a timeout for finding a solution to a linear program using `lp_solve` [3] was set to 120 s.

Table 6. Tool comparison on CTL formulae

CTL simplification only						
	TAPAAL		LoLA		LoLA+Sara	
	Solved	% Solved	Solved	% Solved	Solved	% Solved
Cardinality	1724	34	236	5	904	18
Fireability	495	10	173	3	488	10
Total	**2219**	**22**	**409**	**4**	**1392**	**14**
CTL simplification followed by verification						
Cardinality	4232	85	3634	73	3810	76
Fireability	3712	74	3663	73	3690	74
Total	**7944**	**79**	**7297**	**73**	**7500**	**75**

Table 5 reports the numbers (and percentages) of formulae that were solved (simplified to either *true* or *false*), the number of formulae converted from a complex CTL formula into a pure reachability formula and the average formula reduction in percentages (where the formula size before and after the reductions is measured as a number of nodes in its parse tree).

We can observe that the combination of our techniques simplifies about 34% of cardinality queries and 10% of fireability queries into *true* or *false*, while a significant number of queries are simplified from CTL formula into pure reachability problems (60% of cardinality queries and 40% of fireability ones). The average reduction in the query size is 60% for cardinality and 50% for fireability queries. The results are encouraging, though the performance on fireability formulae is considerably worse than for cardinality formulae. The reason is that fireability predicates are translated into Boolean combinations of cardinality predicates and the expanded formulae are less suitable for the simplification procedures due to their increased size. This is also reflected by the time it took to compute the simplification. For CTL cardinality, half of the simplifications terminate in less than 0.05 s, 75% simplifications terminate in less than 0.98 s and 90% of simplifications terminate in less than 9.46 s. The corresponding numbers for CTL fireability are 0.22 s, 13.70 s and 538.34 s.

5.4 Comparison with LoLA

We compare the performance of our tool-chain, presented in Fig. 3, with the tool LoLA [26] and the combination of LoLA and its linear program solver Sara [25] that uses the CEGAR approach. In CTL simplification experiment, we allow 20 min for formula simplification (excluding the net parsing time) and count how many solved (simplified to *true* or *false*) queries each tool computed[1]. For CTL verification, we allow the tools first simplify the query and then proceed

[1] We use the current development snapshots of LoLA (based on version 2.0) and Sara (based on version 1.14), kindly provided by the LoLA and Sara development team.

Table 7. Tool comparison on reachability formulae

Reachability simplification only						
	TAPAAL		LoLA		LoLA+Sara	
	Solved	% Solved	Solved	% Solved	Solved	% Solved
Cardinality	2256	45	277	6	3734	75
Fireability	1073	21	296	6	2880	58
Total	**3329**	**33**	**573**	**6**	**6614**	**66**
Reachability simplification followed by verification						
Cardinality	4638	93	3734	75	4628	92
Fireability	4402	88	2880	58	4372	87
Total	**9040**	**90**	**6614**	**66**	**9000**	**90**

with the verification according to the best setup the tools provide, again with a 20 min timeout excluding the parsing time. We run LoLA and Sara in parallel (in their advantage), each of them having 20 min timeout per execution. The results are presented in Table 6. We can observe that in simplification of CTL cardinality formulae, we are able to provide an answer for 34% of queries while the combination of LoLA and Sara solves only 18% of them. The performance on the CTL fireability simplification is comparable. If we follow the simplification with an actual verification, TAPAAL solves in total 79% of queries and LoLA with Sara 75%. As a result, TAPAAL with the new query simplification algorithms now outperforms the CTL category winner of the last year.

For completeness, in Table 7, we also include the results for the simplification and verification of reachability queries, even though our method is mainly targeted towards CTL formulae. We can notice that thanks to Sara, a fully functional model checker implementing the CEGAR approach, LoLA in combination with Sara solves twice as many queries by simplification as we do. However, once followed by the actual verification (and due to our simplification technique that significantly reduces formula sizes), both tools now show essentially comparable performance with a small margin towards TAPAAL, solving 40 additional formulae.

6 Conclusion

We presented techniques for reducing the size of a CTL formula interpreted over the Petri net model. The motivation is to speed up the state space search and to provide a beneficial interplay with other techniques like partial order and structural reductions. The experiential results—compared with LoLA, the winner of MCC'17 competition—document a convincing performance for simplification of CTL formulae as well as for CTL verification. The techniques were not designed specifically for the simplification of reachability formulae, hence the number of

solved reachability queries by employing only the simplification is much lower than that by the specialized tools like Sara (being in fact a complete model checker). However, when combined with the state space search followed after the formula simplification, the benefits of our techniques become apparent as we now solve 40 additional formulae compared to the combined performance of LoLA and Sara.

The simplification procedure is less efficient for CTL fireability queries than for CTL cardinality queries. This is the case both for our tool as well as LoLA and Sara. The reason is that we do not handle fireability predicates directly and unfold them into Boolean combination of cardinality predicates. This often results in significant explosion in query sizes. The future work will focus on overcoming this limitation and possibly handling the fireability predicates directly in the engine.

Acknowledgements. We would like to thank Karsten Wolf and Torsten Liebke from Rostock University for providing us with the development snapshot of the latest version of LoLA and for their help with setting up the tool and answering our questions. The last author is partially affiliated with FI MU, Brno.

References

1. Avrunin, G.S., Buy, U.A., Corbett, J.C.: Integer programming in the analysis of concurrent systems. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 92–102. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55179-4_10
2. Avrunin, G.S., Buy, U.A., Corbett, J.C., Dillon, L.K., Wileden, J.C.: Automated analysis of concurrent systems with the constrained expression toolset. IEEE Trans. Softw. Eng. **17**(11), 1204–1222 (1991)
3. Berkelaar, M., Eikland, K., Notebaert, P., et al.: lpsolve: open source (mixed-integer) linear programming system. Eindhoven University of Technology (2004)
4. Blondin, M., Finkel, A., Haase, C., Haddad, S.: Approaching the coverability problem continuously. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 480–496. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_28
5. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982). https://doi.org/10.1007/BFb0025774
6. Clarke, E.M., Emerson, E.A., Sifakis, J.: Model checking: algorithmic verification and debugging. Commun. ACM **52**(11), 74–84 (2009)
7. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. (TOPLAS) **8**(2), 244–263 (1986)
8. Corbett, J.C., Avrunin, G.S.: Using integer programming to verify general safety and liveness properties. Formal Methods Syst. Des. **6**(1), 97–123 (1995)

9. Dalsgaard, A.E., Enevoldsen, S., Fogh, P., Jensen, L.S., Jepsen, T.S., Kaufmann, I., Larsen, K.G., Nielsen, S.M., Olesen, M.C., Pastva, S., Srba, J.: Extended dependency graphs and efficient distributed fixed-point computation. In: van der Aalst, W., Best, E. (eds.) PETRI NETS 2017. LNCS, vol. 10258, pp. 139–158. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57861-3_10
10. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K.Y., Møller, M.H., Srba, J.: TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 492–497. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_36
11. Esparza, J., Melzer, S.: Verification of safety properties using integer programming: beyond the state equation. Formal Methods Syst. Des. **16**(2), 159–189 (2000)
12. Geffroy, T., Leroux, J., Sutre, G.: Occam's Razor applied to the Petri net coverability problem. In: Larsen, K.G., Potapov, I., Srba, J. (eds.) RP 2016. LNCS, vol. 9899, pp. 77–89. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45994-3_6
13. Hack, M.H.T.: Analysis of production schemata by Petri nets. Technical report, DTIC Document (1972)
14. Jensen, J.F., Nielsen, T., Oestergaard, L.K., Srba, J.: TAPAAL and reachability analysis of P/T nets. In: Koutny, M., Desel, J., Kleijn, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency XI. LNCS, vol. 9930, pp. 307–318. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53401-4_16
15. Kordon, F., Garavel, H., Hillah, L.M., Hulin-Hubard, F., Berthomieu, B., Ciardo, G., Colange, M., Dal Zilio, S., Amparore, E., Beccuti, M., Liebke, T., Meijer, J., Miner, A., Rohr, C., Srba, J., Thierry-Mieg, Y., van de Pol, J., Wolf, K.: Complete Results for the 2017 Edition of the Model Checking Contest, June 2017. http://mcc.lip6.fr/2017/results.php
16. Kristensen, L.M., Schmidt, K., Valmari, A.: Question-guided stubborn set methods for state properties. Formal Methods Syst. Des. **29**(3), 215–251 (2006)
17. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
18. Murata, T., Koh, J.Y.: Reduction and expansion of live and safe marked graphs. IEEE Trans. Circ. Syst. **27**(1), 68–70 (1980)
19. Nemhauser, G.L., Wolsey, L.A.: Integer Programming and Combinatorial Optimization. Wiley, Chichester (1992). Nemhauser, G.L., Savelsbergh, M.W.P., Sigismondi, G.S.: Constraint classification for mixed integer programming formulations. COAL Bull. **20**, 8–12 (1988)
20. Schmidt, K.: Stubborn sets for standard properties. In: Donatelli, S., Kleijn, J. (eds.) ICATPN 1999. LNCS, vol. 1639, pp. 46–65. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48745-X_4
21. Schmidt, K.: Integrating low level symmetries into reachability analysis. In: Graf, S., Schwartzbach, M. (eds.) TACAS 2000. LNCS, vol. 1785, pp. 315–330. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46419-0_22
22. Schmidt, K.: Narrowing Petri net state spaces using the state equation. Fundamenta Informaticae **47**(3–4), 325–335 (2001)
23. Valmari, A.: Stubborn sets for reduced state space generation. In: Rozenberg, G. (ed.) ICATPN 1989. LNCS, vol. 483, pp. 491–515. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-53863-1_36
24. Valmari, A., Hansen, H.: Stubborn set intuition explained. In: Koutny, M., Kleijn, J., Penczek, W. (eds.) Transactions on Petri Nets and Other Models of Concurrency XII. LNCS, vol. 10470, pp. 140–165. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-55862-1_7

25. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 224–238. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_19
26. Wolf, K.: Running LoLA 2.0 in a model checking competition. In: Koutny, M., Desel, J., Kleijn, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency XI. LNCS, vol. 9930, pp. 274–285. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53401-4_13

Basis Coverability Graph for Partially Observable Petri Nets with Application to Diagnosability Analysis

Engel Lefaucheux[1,2(✉)], Alessandro Giua[3,4], and Carla Seatzu[3]

[1] Univ. Rennes, Inria, Campus Universitaire de Beaulieu, Rennes, France
engel.lefaucheux@inria.fr
[2] LSV, ENS Paris-Saclay, CNRS, Cachan, France
[3] Department of Electrical and Electronic Engineering, University of Cagliari, Cagliari, Italy
[4] Aix Marseille Univ., Université de Toulon, CNRS, ENSAM, LSIS, Marseille, France

Abstract. Petri nets have been proposed as a fundamental model for discrete-event systems in a wide variety of applications and have been an asset to reduce the computational complexity involved in solving a series of problems, such as control, state estimation, fault diagnosis, etc. Many of those problems require an analysis of the reachability graph of the Petri net. The basis reachability graph is a condensed version of the reachability graph that was introduced to efficiently solve problems linked to partial observation. It was in particular used for diagnosis which consists in deciding whether some fault events occurred or not in the system, given partial observations on the run of the system. However this method is, with very specific exceptions, limited to bounded Petri nets. In this paper, we introduce the notion of basis coverability graph to remove this requirement. We then establish the relationship between the coverability graph and the basis coverability graph. Finally, we focus on the diagnosability problem: we show how the basis coverability graph can be used to get an efficient algorithm.

1 Introduction

The *marking reachability problem* is a fundamental problem of Petri nets (PNs) which can be stated as follows: *Given a net system* $\langle N, M_0 \rangle$ *and a marking* M*, determine if* M *belongs to the reachability set* $R(N, M_0)$. It plays an important role since many other properties of interest can be solved by reduction to this problem. The marking reachability problem has been shown to be decidable in [11] and was shown to be EXPSPACE-hard in [15].

In the case of *bounded* PNs, i.e., net systems whose reachability set is finite, a straightforward approach to solve this problem consists in constructing the *reachability graph*, which provides an explicit representation of the net behavior, i.e., its reachability set and the corresponding firing sequences of transitions. However, albeit finite, the reachability graph may have a very large number of nodes

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 164–183, 2018.
https://doi.org/10.1007/978-3-319-91268-4_9

due to the so called *state space explosion* that originates from the combinatorial nature of discrete event systems. For this reason, practically efficient approaches, which do not require to generate the full state space, have been explored. We mention, among others, partial order reduction techniques, such as the general approaches based on stubborn sets [18] and persistent sets [8] or the Petri net approaches based on unfolding [13] and maximal permissive steps [2].

In the case of *unbounded* PNs, whose reachability set is infinite, the authors of [9] have shown that a finite *coverability graph* may be constructed which provides a semi-decision procedure (necessary conditions) for the marking reachability problem. It provides an over-approximation of both the reachability set and the set of firing sequences. As was the case for the reachability graph, this approach is not efficient and improvements to the basic algorithm have later been proposed [14].

Recently some of us have proposed a quite general approach that exploits the notion of *basis marking* to practically reduce the computational complexity of solving the reachability problem for bounded nets. This method has originally been introduced to solve problems of state estimation under partial observation [7] but has later been extended to address fault diagnosis [4], state-based opacity [17] and general reachability problems [10].

The approach in [10] considers a partition of the set of transitions $T = T_e \cup T_i$: T_e is called set of *explicit transitions* and T_i is called set of *implicit transitions*. The main requirement is that the subnet containing only implicit transitions be acyclic. The firing of implicit transitions is abstracted and only the firing of explicit transitions need to be enumerated. The advantage of this technique is that only a subset of the reachability space—i.e., the set of the so-called basis markings—is enumerated. All other markings are reachable from a basis marking by firing only implicit transitions and can be characterized by the integer solutions of a system of linear equations. In a certain sense, this hybrid approach combines a behavioral analysis (limited to the firing of transitions in T_e) with a structural analysis (which describes the firing of transitions in T_e).

The objective of this paper is mainly to show that the approach of [10] can be generalized to unbounded nets. We define a *basis coverability graph* where the firing of implicit transitions is abstracted, thus reducing the number of nodes of the standard coverability graph. In addition, we show how this approach can be applied to study the *diagnosability* of Petri nets in the logic framework of [16]. Diagnosability is achieved in a system where transitions can be observable or not is one can deduce from the observations that a specified faulty transition was fired. In this case, we consider as implicit the set of unobservable transitions. However, since the firing of unobservable faulty transitions need to be recorded, we further extend the approach of [10] by considering that there may exists a subset of implicit transitions (called *relevant transitions*) which, albeit abstracted, need to be handled with special care. In terms of computational cost, relevant transitions are in between observable and implicit transitions.

The paper is structured as follows. In Sect. 2, we recall some usual definitions for Petri Nets and their coverability graph. In Sect. 3, we introduce the notion of

basis coverability graph and establish some of its properties. In Sect. 4 we give the definitions of the diagnosability of a Petri Net. Finally in Sect. 5 we study unbounded Petri nets and show how to use the basis coverability graph for the diagnosability analysis.

2 Background on Petri Nets and Coverability Graph

2.1 Petri Nets

In this section the formalism used in the paper is recalled. For more details on Petri nets the reader is referred to [12].

Definition 1. *A Petri net (PN) is a structure $N = (P, T, Pre, Post)$, where P is a set of m places; T is a set of n transitions; $Pre : P \times T \to \mathbb{N}$ and $Post : P \times T \to \mathbb{N}$ are the pre– and post– incidence functions that specify the arcs. We also define $C = Post - Pre$ as the incidence matrix of the net.*

A *marking* is a vector $M : P \to \mathbb{N}$ that assigns to each place of a PN a nonnegative integer number of tokens. A *net system* (NS) $\langle N, M_0 \rangle$ is a PN N with an initial marking M_0. A transition t is enabled at M iff $M \geq Pre(\cdot, t)$ and may fire yielding the marking $M' = M + C(\cdot, t)$. One writes $M\ [\sigma\rangle$ to denote that the sequence of transitions $\sigma = t_{j_1} \cdots t_{j_k}$ is enabled at M, and $M\ [\sigma\rangle\ M'$ to denote that the firing of σ yields M'. One writes $t \in \sigma$ to denote that a transition t is contained in σ. The length of the sequence σ (denoted$|\sigma|$) is the number of transitions in the sequence, here k.

The set of all sequences that are enabled at the initial marking M_0 is denoted $L(N, M_0)$, i.e., $L(N, M_0) = \{\sigma \in T^* \mid M_0[\sigma\rangle\}$. Given $k \geq 0$, the set of all sequences of length k is written T^k. A marking M is *reachable* in $\langle N, M_0 \rangle$ iff there exists a firing sequence σ such that $M_0\ [\sigma\rangle\ M$. The set of all markings reachable from M_0 defines the *reachability set* of $\langle N, M_0 \rangle$ and is denoted $R(N, M_0)$.

Let $\pi : T^* \to \mathbb{N}^n$ be the function that associates with the sequence $\sigma \in T^*$ a vector $y \in \mathbb{N}^n$, called the *firing vector* of σ. In particular, $y = \pi(\sigma)$ is such that $y(t) = k$ iff the transition t is contained k times in σ.

A PN having no directed circuits is called *acyclic*. Given $k \in \mathbb{N}$, a place p of an NS $\langle N, M_0 \rangle$ is *k-bounded* if for all $M \in R(N, M_0)$, $M(p) \leq k$. It is bounded if there exists $k \in \mathbb{N}$ such that p is k-bounded. An NS is bounded (resp. k-bounded) iff all of its places are bounded (resp. k-bounded).

A sequence is *repetitive* iff it can be repeated indefinitely (i.e. σ is repetitive in the marking M iff $M[\sigma\rangle M'$ with $M' \geq M$). There are two kinds of repetitive sequences: a repetitive sequence is *stationary* if it does not modify the marking (i.e. $M[\sigma\rangle M$), it is *increasing* otherwise. Remark that an NS containing an increasing sequence can not be bounded.

Example 1. Consider the NS of Fig. 1, the sequence t_1, is increasing in the initial marking $M_0 = [2, 0, 0, 0, 0]$. Firing t_1 k times in M_0 leads to the marking $M_1 = [2, k, 0, 0, 0]$. Therefore the place p_2 is not bounded. However, every other place is 2-bounded.

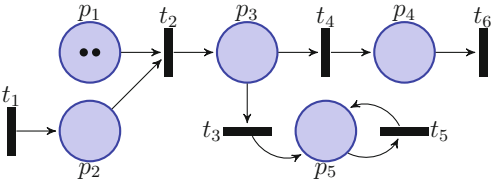


Fig. 1. A net system. Circles are places and rectangles are transitions. In the initial marking, p_1 has two tokens represented by the two black dots.

Definition 2. *Given a net $N = (P, T, Pre, Post)$, and a subset $T' \subseteq T$ of its transitions, let us define the* $T'-$induced subnet *of N as the new net $N' = (P, T', Pre', Post')$ where $Pre', Post'$ are the restrictions of $Pre, Post$ to T'. The net N' can be thought as obtained from N removing all transitions in $T \setminus T'$. Let us also write $N' \prec_{T'} N$.*

2.2 Coverability Graph

For a bounded NS $\langle N, M_0 \rangle$, one can enumerate the elements of the reachability set $R(N, M_0)$ and establish the transition function between the markings. The resulting graph is called *Reachability Graph*. If the NS is not bounded, this construction does not terminate. Instead, an usual method is to build the *Coverability Graph* which is a finite over-approximation of the reachability set and of the net language [9]. We will define in this section the coverability graph of an NS which if the NS is bounded is equal to the reachability graph of this NS.

An ω-marking is a vector from the set of places to $\mathbb{N} \cup \{\omega\}$, where ω should be thought of as "arbitrarily large": for all $k \in \mathbb{N}$, we have $k < \omega$ and $\omega \pm k = \omega$. An ω-marking M is (resp. strictly) covered by an ω-marking M', written $M \leq M'$ (resp. $M \lneq M'$) iff for every place p of the net, $M(p) \leq M'(p)$ (resp. and there exists at least one place p such that $M(p) < M'(p)$).

Definition 3. *Given an NS $\langle N, M_0 \rangle$, the associated coverability graph $CG_{\langle N, M_0 \rangle} = (\mathcal{M}, M_0, \Delta)$ is defined in the following manner.*

We first define inductively a temporary set $\mathcal{M}_t$ of pairs of ω-markings and set of ω-markings and the temporary transition function Δ_t by:

- *$(M_0, \{M_0\}) \in \mathcal{M}_t$ and*
- *$(M', B') \in \mathcal{M}_t$ iff there exists $(M, B) \in \mathcal{M}_t$ and $t \in T$ such that*
  - *either $M[t\rangle M'$, $B' = B \cup \{M'\}$ and for all $M'' \in B, M' \not\geq M''$;*
  - *or, for M^t such that $M[t\rangle M^t$, there exists $M'' \in B$ such that $M^t \gneq M''$. For every such M'', let $p_1, \ldots, p_k$ be the set of places such that for all j, $M^t(p_j) \gneq M''(p_j)$, then $\forall j, M'(p_j) = \omega$. For every place p such that $M'(p) \neq \omega$, $M'(p) = M^t(p)$. Moreover $B' = B \cup \{M'\}$.*

In both cases, $((M, B), t, (M', B')) \in \Delta_t$.

We then define $\mathcal{M} = \{M \mid \exists B, (M, B) \in \mathcal{M}_t\}$ and given M and M' in $\mathcal{M}$, $(M, t, M') \in \Delta$ iff there exists B, B' such that $((M, B), t, (M', B')) \in \Delta_t$.

The temporary graph built here is equivalent to the coverability tree of [5]. They proved in [9] that the coverability tree (and thus our temporary graph) terminates in a finite number of steps.

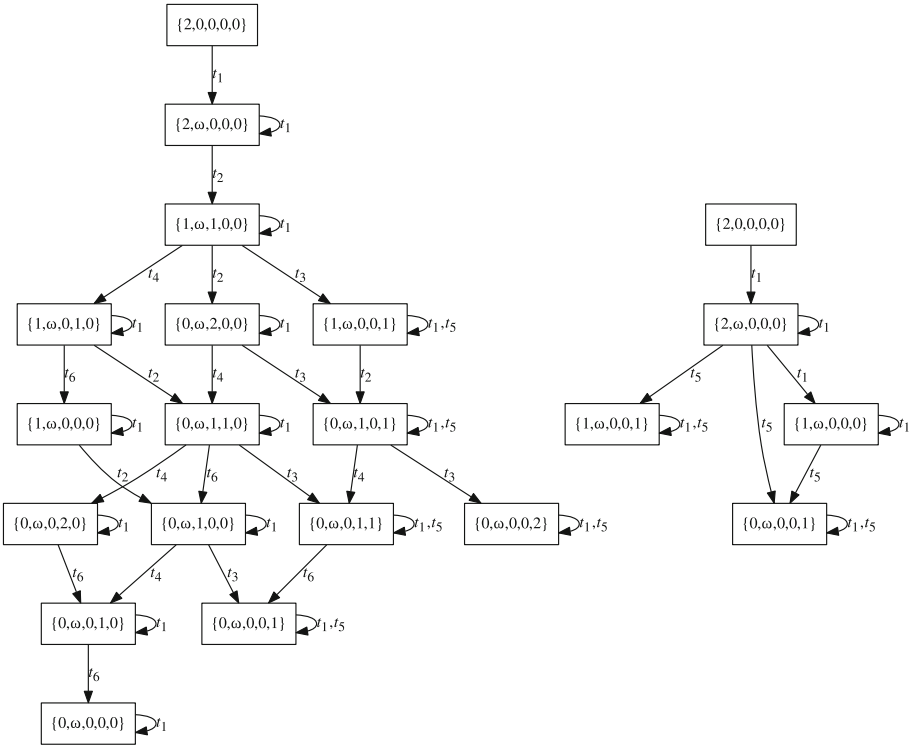


Fig. 2. Left: the coverability graph of the NS in Fig. 1. Right: the BCG of the NS in Fig. 1 where $T_i = \{t_2, t_3, t_4, t_6\}$ and $T_s = \{t_6\}$. The firing vectors are omitted on the edges.

Example 2. The coverability graph of the NS in Fig. 1 is shown in Fig. 2. The firing of t_1 at the initial marking adds a token to the second place, reaching a marking strictly greater than the initial marking in this place and equal everywhere else. Correspondingly in the coverability graph an ω appears in the second component of the marking to show that there is a repetitive sequence enabled by the system which increases the number of tokens in the second place.

A marking M is ω-covered by an ω-marking M_ω, denoted $M \leq_\omega M_\omega$ if for every place p such that $M_\omega(p) \neq \omega$, $M_\omega(p) = M(p)$. Using this definition and the coverability graph, we define the coverability set of an NS which is an over-approximation of the reachability set.

Definition 4. *Given an NS* $\langle N, M_0 \rangle$*, let* $\mathcal{M}$ *be the set of* ω*-markings of its coverability graph, the coverability set of* $\langle N, M_0 \rangle$ *is*

$$CS(N, M_0) = \{M \in \mathbb{N}^m \mid \exists M_\omega \in \mathcal{M}, M \leq_\omega M_\omega\}$$

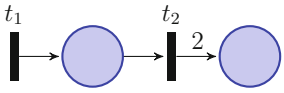


Fig. 3. A Petri net where the coverability set strictly subsumes the reachability set. Transition t_2 is unobservable.

Example 3. The coverability set of the NS in Fig. 1 is equal to its reachability set. This is not the case however for the NS in Fig. 3 where the reachability set is $\{(k, 2r) \mid k, r \in \mathbb{N}\}$ while the coverability set is $\{(k, r) \mid k, r \in \mathbb{N}\}$. We however clearly see that the coverability set subsumes the reachability set. The coverability graph of this NS is represented in Fig. 4.

Fig. 4. Left: the coverability graph of the NS in Fig. 3. Right: the BCG of the NS in Fig. 3 where $T_i = \{t_2\}$ and $T_s = \emptyset$. The firing vectors are omitted on the edges.

We will use the rest of this section to recall a few known applications of the coverability graph and the coverability set. All those results can be found in [5]. First, as claimed earlier, the coverability set subsumes the reachability set.

Proposition 1. *Let* $\langle N, M_0 \rangle$ *be an NS,* $R(N, M_0) \subseteq CS(N, M_0)$*.*

The coverability graph can be used to determine if an NS is bounded.

Proposition 2. *Given an NS* $\langle N, M_0 \rangle$*,*

- *a place* p *is* k*-bounded* $\Leftrightarrow$ *for each marking* M *of* $CG_{\langle N, M_0 \rangle}$*,* $M[p] \leq k$*.*
- *the marked net is bounded* $\Leftrightarrow$ *no node of* $CG_{\langle N, M_0 \rangle}$ *contains the symbol* ω*.*

Repetitiveness can be partially checked on the coverability graph.

Proposition 3. *Given an NS* $\langle N, M_0 \rangle$*, a marking* M *and a non-empty sequence* σ' *of transitions enabled by* M*,*

- σ *is repetitive* $\Rightarrow$ *there exists a directed cycle in the coverability graph whose arcs form* σ *starting in an* ω *marking* M_ω *such that* $M_\omega \geq_\omega M$*.*
- σ *is stationary* $\Leftarrow$ *there exists a directed cycle starting in* M *in the graph that does not pass through markings containing* ω *and whose arcs form* σ*.*

The coverability graph can also be used to test whether a transition can eventually be fired by the NS. A transition t is dead if there is no reachable marking enabling it, it is quasi-live otherwise. It is live if for all reachable markings M, there is a marking M' reachable from M enabling it. A marking is dead if every transition is dead from this marking.

Proposition 4. *Consider a marked net $\langle N, M_0\rangle$, its coverability graph and an observable transition t.*

1. *Transition t is dead $\Leftrightarrow$ no arc labelled t belongs to the graph.*
2. *Transition t is quasi-live $\Leftrightarrow$ an arc labelled t belongs to the graph.*
3. *Transition t is live $\Rightarrow$ an arc labelled t belongs to each ergodic component of the graph.*
4. *A marking M is dead $\Leftarrow$ one ω-marking M_ω of the coverability graph ω-covering M has no output arc.*

3 Basis Coverability Graph

3.1 Building the Basis Coverability Graph

While the reachability/coverability graph has many applications, one of its downside is its size. For bounded NS, the authors of [4,6] introduced the notion of basis reachability graph which keeps most of the information relevant for partially observed systems of the reachability graph while decreasing, in some cases exponentially, the size of the graph. Their goal at the time was to study diagnosis. They then generalised this approach to study reachability (regardless of labeling on transitions) in [10]. The idea of the basis reachability graph is to select a set of transitions called "implicit" in [10] (and unobservable in [4]) that will be abstracted and to only represent the "explicit" transitions that can be fired (possibly after some implicit transition) in a given marking. In this section, we will describe how to apply this idea to unbounded NS and how to build instead a *Basis Coverability Graph* (BCG). When the NS is bounded, the BCG is equal to the basis reachability graph.

Given a set of transitions T of a PN, we denote $T_i \subseteq T$ and $T_e = T \backslash T_i$ the sets of *implicit* and *explicit* transitions respectively. Let C_i (C_e) be the restriction of the incidence matrix to T_i (T_e) and n_i and n_e, respectively, be the cardinality of the above sets of transitions. Given a sequence $\sigma \in T^*$, $P_i(\sigma)$, resp., $P_e(\sigma)$, denotes the projection of σ over T_i, resp., T_e.

We will sometimes need the following assumptions.

A1: The T_i-induced subnet is acyclic.
A2: Every sequence containing only implicit transitions is of finite length.

Remark that for bounded NS, the first assumption, which is an usual requirement for problems such as diagnosis of discrete event systems, implies the second one.

When the partition between implicit and explicit transitions is not given, one can always choose a partition respecting the two assumptions above (for example

$T_e = T$). The authors of [10] discuss how to choose an appropriate partition for the basis reachability graph and how this choice affects the cardinality of the set of markings of the graph.

Definition 5. *Given a marking M and an explicit transition $t \in T_e$, let*

$$\Sigma(M,t) = \{\sigma \in T_i^* \mid M[\sigma\rangle M',\ M' \geq Pre(\cdot,t)\}$$

be the set of explanations *of t at M, and let*

$$Y(M,t) = \pi(\Sigma(M,t))$$

be the e-vectors *(or* explanation vectors*),* i.e.*, firing vectors associated with the explanations.*

Thus $\Sigma(M,t)$ is the set of implicit sequences whose firing at M enables t. Among the above sequences we will select those whose firing vector is minimal and those who are minimal while containing a transition among a chosen set $T_s \subseteq T_i$ which will be called the set of *relevant* transitions. This second category is used to solve problems where it may be necessary to keep track of the occurrence of a subset of implicit transitions. In particular it will be used in the section about diagnosis later in this paper. As they are transitions we want to take into account yet are not fully explicit, relevant transitions are more costly than implicit transitions yet not as much as explicit transitions as we will see later. The firing vector of these sequences are called *(T_s-) minimal e-vectors.*

Definition 6. *Given a marking M, a transition $t \in T_e$ and a set of relevant transitions $T_s \subseteq T_i$, let*

$$\Sigma_{\min}(M,t) = \{\sigma \in \Sigma(M,t) \mid \nexists\ \sigma' \in \Sigma(M,t) : \pi(\sigma') \lneq \pi(\sigma)\}$$

be the set of minimal explanations *of t at M, and*

$$\Sigma_{\min}^{T_s}(M,t) = \{\sigma \in \Sigma(M,t) \mid \sigma \cap T_s \neq \emptyset \wedge \nexists\ \sigma' \in \Sigma(M,t) : \sigma \cap T_s = \sigma' \cap T_s \wedge \pi(\sigma') \lneq \pi(\sigma)\}$$

*the set of T_s-*minimal explanations *of t at M.*

Remark that for two sets of relevant transitions $T_s \subseteq T_i$ and $T_s' \subseteq T_i$, if $T_s \subseteq T_s'$, for every marking M and explicit transition $t \in T_e$, $\Sigma_{\min}^{T_s}(M,t) \subseteq \Sigma_{\min}^{T_s'}(M,t)$. We will now build the BCG with a construction similar to the one of the coverability graph. From a given marking, instead of choosing a transition and creating the marking obtained by firing this transition, we will fire a sequence composed of a minimal explanation of an explicit transition followed by the explicit transition in question. In other words, we skip all the intermediary markings that were reached by the firing of the implicit transitions. Moreover, in order to determine which places are labelled by ω, instead of only remembering the markings encountered, the temporary construction keeps pairs of marking encountered and of the e-vector of the minimal sequence fired from that marking. From these pairs, one can reconstruct every marking that could have been reached.

Definition 7. *Given an NS $\langle N, M_0\rangle$ verifying Assumption (A1) and a set of relevant transition $T_s \subseteq T_i$, the associated basis coverability graph (BCG) with relevant transitions T_s $BCG^{T_s}_{\langle N,M_0\rangle} = (\mathcal{M}, M_0, \Delta)$ is defined in the following manner.*

We first define inductively a temporary set $\mathcal{M}_t$ of pairs of ω-markings and set of pairs of ω-markings and firing vectors and the temporary transition function Δ_t by:

- $(M_0, \emptyset) \in \mathcal{M}_t$ *and*
- $(M', B') \in \mathcal{M}_t$ *iff there exists* $(M, B) \in \mathcal{M}_t$, $t \in T_e$ *and* $\sigma \in \Sigma_{\min}(M, t) \cup \Sigma^{T_s}_{\min}(M, t)$ *with* $M[\sigma t\rangle M_n$, *where* $B' = B \cup \{(M, \pi(\sigma))\}$ *and*
  - *either* $M_n = M'$ *and for all* $(M'', \pi) \in B$, $M[\sigma_1\rangle M_1$ *and* $M''[\sigma_2\rangle M_2$ *with* $\pi(\sigma_1) \leq \pi(\sigma t)$ *and* $\pi(\sigma_2) \leq \pi$, *we have* $M_1 \not\geq M_2$;
  - *or there exists* $(M'', \pi) \in B$, $M[\sigma_1\rangle M_1$ *and* $M''[\sigma_2\rangle M_2$ *with* $\pi(\sigma_1) \leq \pi(\sigma t)$ *and* $\pi(\sigma_2) \leq \pi$ *such that* $M_1 \geq M_2$. *Then let* $p_1, \ldots, p_k$ *be the places such that* $\forall i, M_1(p_i) > M_2(p_i)$. *Let* $\tilde{M}$ *be the marking obtained from* M *by replacing the number of tokens of the places* p_i *by* ω. *We repeat the tests from the marking* $\tilde{M}$ *until no new place can be modified. Let* $p_1, \ldots, p_n$ *be the places of* M *where an* ω *was added in this process. Then for every place* p, *if* $p \in \{p_1, \ldots, p_n\}$, $M'(p) = \omega$, *otherwise* $M'(p) = M_n(p)$.

In both cases $((M, B), (\pi(\sigma), t), (M', B')) \in \Delta_t$.

We then define $\mathcal{M} = \{M \mid \exists B, (M, B) \in \mathcal{M}_t\}$ *and given* M *and* M' *in* $\mathcal{M}$, $(M, (\pi(\sigma), t), M') \in \Delta$ *iff there exists* B, B' *such that* $((M, B), (\pi(\sigma), t), (M', B')) \in \Delta_t$.

The markings of the BCG are called basis markings.

This construction does not require to check every implicit sequence of transitions. Indeed, it only focuses on the firing vectors which can be more efficiently analysed. This is only possible thanks to Assumption (A1), or more precisely thanks to Theorem 3.8 of [4] which requires (A1) (this is the result used every time Assumption (A1) is required in the following). This results implies that due to the acyclicity of the implicit net, the implicit transitions of an explanation that are not part of the minimal explanation can be postponed after the explicit transition. This gives an important leeway on the order in which the implicit transitions have to be in. Removing the Assumption (A1) would allow the construction of a variant of the BCG defined here, but its construction would be more costly as we would need to consider the minimal explanations instead of their e-vector and the variant may construct more states than the one built here.

We denote the projection of a sequence $\sigma = (\sigma_1, t_1) \ldots (\sigma_k, t_k) \ldots$ of the BCG on its second component by $P_t(\sigma) = t_1 \ldots t_k \ldots$.

Example 4. Let us first consider the NS in Fig. 3 with $T_i = \{t_2\}$ and no relevant transition and the associated coverability graph and BCG in Fig. 4. In order to fire t_1, firing t_2 is not required. As a consequence, the firing of the transition t_2 is never used in the construction of the BCG.

As another example, we represent the BCG of the NS in Fig. 1 (for $T_i = \{t_2, t_3, t_4, t_6\}$ and $T_s = \{t_6\}$) in Fig. 2. For readability the firing vectors on the

edges are omitted in the figure. This BCG has 11 less states than the coverability graph.

Choosing $T_s = \{t_6\}$ adds the states $\{1, \omega, 0, 0, 0\}$ and $\{0, \omega, 0, 0, 1\}$ and the edges affecting those states. The edge from $\{2, \omega, 0, 0, 0\}$ to $\{0, \omega, 0, 0, 1\}$ corresponds to the $\{t_6\}$-minimal explanation $t_2t_2t_3t_4t_6$ of t_5 which is not a minimal explanation.

3.2 Properties of the Basis Coverability Graph

We will list here some of the properties of the BCG. We will first give a few results on the size of the BCG compared to the coverability graph and under variations of the sets of explicit, implicit and relevant transitions. Then we will define the notion of basis coverability set and show it is a better approximation of the reachability set than the coverability set. Finally, we will see how the properties of boundedness, repetitiveness and liveness translate from the coverability graph to the BCG.

The BCG was introduced in order to gain in efficiency compared to the coverability graph. The first property to mention is thus that the BCG is always smaller than or equal to the coverability graph. This is formally proved in the following.

Proposition 5. *Given an NS $\langle N, M_0\rangle$ verifying Assumption (A1) with set of implicit transitions T_i, for any set of relevant transitions $T_s \subseteq T_i$ it holds that every basis marking M of $BCG^{T_s}_{\langle N,M_0\rangle}$ is a marking of $CG_{\langle N,M_0\rangle}$.*

Proof. As any marking of the BCG is reachable from M_0, we will show the result by induction on the length of a path reaching this marking. Let M be a marking of $BCG^{T_s}_{\langle N,M_0\rangle}$ and σ a sequence such that $M_0[\sigma\rangle M$.

If $|\sigma| = 0$, $M = M_0$ which is a marking of $CG_{\langle N,M_0\rangle}$.

Given $n \in \mathbb{N}$, suppose that the property is true for every marking reached by a path of length at most n. If $|\sigma| = n + 1$, there exists a sequence σ_1 and a transition (e, t) of $BCG^{T_s}_{\langle N,M_0\rangle}$ such that $\sigma = \sigma_1(e, t)$, let M_1 such that $M_0[\sigma_1\rangle M_1$, then by hypothesis M_1 belongs to $CG_{\langle N,M_0\rangle}$. Moreover, as there is a transition $M_1[(e, t)\rangle M$ in the BCG, there exists a minimal explanation $\sigma' = t_1, \dots t_n \in \Sigma_{\min}(M_1, t) \cup \Sigma^{T_s}_{\min}(M_1, t)$ such that $\pi(\sigma') = e$ and one of the two conditions for a BCG transition between M_1 and M is validated. If it is the first condition, there is a path $M_1[t_1\rangle M_2 \dots [t_n\rangle M_n[t\rangle M$ in $CG_{\langle N,M_0\rangle}$, thus M is a marking of $CG_{\langle N,M_0\rangle}$. If it is the second condition, assume the process ends in a single round. Then there exists a marking $M_<$ either encountered while reading σ in the BCG or reachable by a subset of a minimal explanation that is smaller than a marking $M_>$ reachable from M_1. As all the markings of the BCG encountered along σ_1 be longs to the coverability graph, using Assumption (A1) there is a path in the coverability graph from $M_<$ to an ω-marking $M^\omega_>$ with $M^\omega_> \geq_\omega M_>$ and as the process occurs only once, the only places where $M^\omega_>$ contains an ω and $M_>$ does not are the places where $M_>$ is strictly greater than $M_<$. By firing the transitions corresponding to the firing vector e that are left after reaching $M_>$ in the coverability graph from $M^\omega_>$ we reach M. The same

idea works if the process requires multiple rounds, however, in order to visit every states that are covered and covering, one may need to extend the run. As a consequence, M is a marking of $CG_{\langle N,M_0\rangle}$. □

How much is gained depends on the partition between implicit, explicit and relevant transitions. For example, if every transition is explicit, the BCG is exactly equal to the coverability graph. On the contrary, increasing the number of implicit transitions reduce the size of the BCG.

Proposition 6. *Consider an NS $\langle N, M_0\rangle$ verifying Assumption (A1) and two sets of implicit transitions T_i and T_i' with $T_i' \subseteq T_i$. For any set of relevant transitions T_s such that $T_s \subseteq T_i'$, every basis marking of the BCG of $\langle N, M_0\rangle$ with implicit transitions T_i is a basis marking of the BCG of $\langle N, M_0\rangle$ with implicit transition T_i'.*

Proof. We call C and C' the two BCG with implicit sets of transitions T_i and T_i'. We will show that any basis marking M of C is a basis marking of C' by induction on the length n of the sequence reaching it. If $n = 0$, $M = M_0$ and belongs to C'. Else, there is a sequence $\sigma = \sigma_1(e, t)$ (with e a firing vector of implicit transition and t an explicit transition) and a basis marking M_0 such that $M[\sigma_1\rangle M_0[(e, t)\rangle M$ in C. By induction hypothesis, M_0 is a basis marking of C'. Let σ' be a minimal explanation of t with $\pi(\sigma') = e$. $\sigma' = \sigma_0 t_0 \sigma_1 \dots t_k \sigma_{k+1}$ where for all $i \leq k+1$ the transitions t_i are explicit transitions and the σ_i are sequences of implicit transitions with respect to T_i'. Due to Assumption (A1), we can assume without loss of generality that the σ_i, $i \leq k$, are minimal explanations of t_i. Therefore there exists basis markings in C', $M_1, \dots M_k$, such that $M_0[(\pi(\sigma_0), t_0)\rangle M_1 \dots [(\pi(\sigma_k), t_k)\rangle M_k[(\pi(\sigma_{k+1}), t)\rangle M$. Thus M belongs to C'. □

With a similar proof, we can show that turning implicit transitions into relevant transitions increases the number of markings.

Proposition 7. *Consider an NS $\langle N, M_0\rangle$ verifying Assumption (A1) with set of implicit transitions T_i. For any two sets of relevant transitions $T_s, T_s' \subseteq T_i$ with $T_s \subseteq T_s'$, every basis marking of $BCG^{T_s}_{\langle N,M_0\rangle}$ is a basis marking of $BCG^{T_s'}_{\langle N,M_0\rangle}$.*

Relevant transitions are in between explicit and implicit transitions in terms of cost. This is strict as seen for example on the NS in Fig. 3 when choosing t_2 explicit and t_s either explicit or relevant (the associated BCG are represented

Fig. 5. Left: the BCG of the NS in Fig. 3 with t_1 and t_2 explicit. Right: the BCG of the NS in Fig. 3 with t_1 relevant and t_2 explicit. The firing vectors are omitted on the edges.

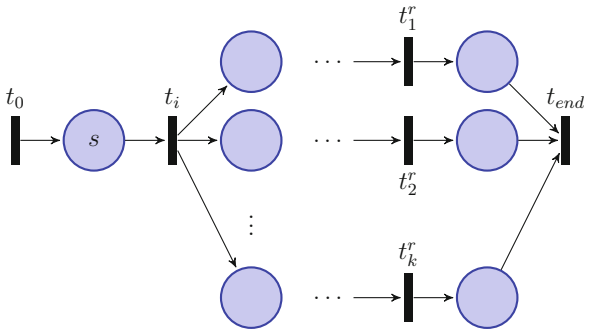


Fig. 6. A Petri net where the BCG has exponentially less states than the coverability graph. s is the number of tokens contained in this place in the initial marking and there is $r + 1$ places in each of the parallel lines.

in Fig. 5). Here, making the transition relevant instead of explicit removes one basis marking. In fact, on this example making t_1 relevant or implicit does not change anything contrary to what was seen in Example 4.

Let us now discuss about the importance of the gain of the BCG construction through an example.

Example 5. Consider the NS in Fig. 6, transitions t_0, t_i and t_{end} being explicit while the others are implicit. The BCG has exactly $\frac{(s+3)(s+2)}{2}$ basis markings, thus is quadratic in s and does not depend on r or k. However, the coverability graph has at least $\sum_{j=0}^{s} \binom{r+j}{j}^k$ markings (this is the number of markings reached while never firing t_0). Thus is among others exponential in k. Moreover, as this is without firing t_0, this only describes a part of the coverability graph that do not contain any ω.

We will now give some results showing that the BCG can effectively be used in many cases instead of the coverability graph. As a first step, we will show that the BCG can be used to define a set of markings that are an over-approximation of the reachability set. We denote by $R_i(N, M)$ the set of markings reachable from M using only implicit transitions in the Petri net N. Given an ω-marking M_ω and a marking M, $M_\omega =_\omega M$ iff for every place p such that $M_\omega(p) \neq \omega$, $M_\omega(p) = M(p)$.

Definition 8. *Given an NS $\langle N, M_0\rangle$ with m places and a set of implicit transitions T_i, let $T_s \subseteq T_i$ be a set of relevant transitions and let V be the set of basis markings of $BCG^{T_s}_{\langle N,M_0\rangle}$. The basis coverability set of $\langle N, M_0\rangle$ with relevant transitions T_s is*

$$BCS^{T_s}(N, M_0) = \{M \in \mathbb{N}^m \mid \exists M_\omega \in V, \exists M^u_\omega \in R_i(N, M_\omega), M^u_\omega \geq_\omega M\}$$

This set can be easily computed for NS verifying (A1). For every possible choice of T_s, the basis coverability set is an over-approximation of the reachability set.

Proposition 8. *Given an NS $\langle N, M_0\rangle$ with set of implicit transitions T_i verifying Assumption (A1) and a set of relevant transitions $T_s \subseteq T_i$, it holds $R(N, M_0) \subseteq BCS^{T_s}(N, M_0)$.*

Proof. Let σ be a sequence such that $M_0[\sigma\rangle M$ in the NS. We will proceed by induction on the length of σ.

If $|\sigma| = 0$, $M = M_0$ which is a marking of the BCG.

Given $n \in \mathbb{N}$, supposing that the property is true for every marking reached by a path of length at most n. For $|\sigma| = n + 1$, $\sigma = \sigma_1 t$. Let $M_0[\sigma_1\rangle M_1$. By the induction hypothesis there exists a basis marking M^b_ω and an ω-marking M^u_ω such that $M^u_\omega \in R_i(N, M^b_\omega)$ and $M^u_\omega \geq_\omega M_1$.

- if t is implicit, as $M^u_\omega \geq_\omega M_1$, t is enabled by M^u_ω and the marking reached by firing t in M^u_ω, let's call it $M^{u,2}_\omega$, verifies $M^{u,2}_\omega \geq_\omega M$ and $M^{u,2}_\omega \in R_i(N, M^b_\omega)$.
- if t is explicit, let σ_t such that $M^b_\omega[\sigma_t\rangle M^u_\omega$. Since σ_t is an explanation of t, there thus exist a minimal explanation σ_{min} such that $\pi(\sigma_{min}) \leq \pi(\sigma_t)$ and a sequence $\sigma_e \in T_i^*$ such that $\pi(\sigma_{min})+\pi(\sigma_e) = \pi(\sigma_t)$. Let $M_s \leq_\omega M^b_\omega$ such that $M_s[\sigma_t\rangle M_1$ and M_f the marking such that $M_s[\sigma_{min}t\rangle M_f$. Using Assumption (A1), $M_f[\sigma_e\rangle M$. By construction of the BCG, there exists a basis marking M^b_2 reachable with transition $(\pi(\sigma_{min}), t)$ from M^b_ω such that $M^b_2 \geq_\omega M_f$. Moreover, as $M_f[\sigma_e\rangle M$, triggering σ_e in M^b_2 leads to a marking M^2_ω such that $M^2_\omega \geq_\omega M$. □

The following result characterizes a monotonicity property of the basis coverability set with respect to the corresponding set of relevant transitions. It is a direct corollary of Proposition 7.

Corollary 1. *Given an NS $\langle N, M_0\rangle$ verifying Assumption (A1) with set of implicit transitions T_i. For any two sets of relevant transitions T_s and T'_s such that $T_s \subseteq T'_s \subseteq T_i$, $BCS^{T_s}(N, M_0) \subseteq BCS^{T'_s}(N, M_0)$.*

The inclusion can be strict. Indeed, let us observe the NS of Fig. 3 with t_2 implicit. The BCG with $T_s = \emptyset$ has two basis markings $[0, 0]$ and $[\omega, 0]$. The associated basis coverability set is $\{[n, 2m] \mid n, m \in \mathbb{N}\}$, which is equal to the reachability set. However, the BCG with $T_s = \{t_2\}$ has the two previous basis markings plus $[\omega, \omega]$. Therefore its basis coverability set is $\{[n, m] \mid n, m \in \mathbb{N}\}$, which is equal to the coverability set. In fact the basis coverability set is always a better approximation than the coverability set.

Proposition 9. *Given an NS $\langle N, M_0\rangle$ verifying Assumption (A1) with set of implicit transitions T_i and a set of relevant transitions $T_s \subseteq T_i$, it holds $BCS^{T_s}(N, M_0) \subseteq CS(N, M_0)$.*

Proof. Let $M \in BCS(N, M_0)$. By definition, there exists $M' \geq_\omega M$ and M_b state of the BCG with $M' \in R_i(N, M_b)$. By Proposition 5, M_b is a state of $CG_{\langle N,M_0\rangle}$. Let σ be an implicit sequence such that $M_b[\sigma\rangle M'$. By definition of R_i and of the coverability graph there exists a state M_c of $CG_{\langle N,M_0\rangle}$ such that $M_b[\sigma\rangle M_c$ in the CG and $M_c \geq M'$. Thus $M_c \geq M$. Therefore $M \in CS(N, M_0)$. □

We now show how the results relative to the coverability graph recalled in the previous section (namely Propositions 2 and 3) can be transposed on the BCG. As those results hold for every choice of set of relevant transitions $T_s \subseteq T_i$, this set is omitted for the rest of the section.

Proposition 10. *Given an NS $\langle N, M_0 \rangle$ with set of implicit transitions T_i verifying Assumptions (A1) and (A2),*

- *a place p is k-bounded $\Rightarrow$ for each basis marking M of the BCG $M[p] \leq k$.*
- *p is not k-bounded $\Rightarrow$ there exists a basis marking M_ω and an ω-marking $M_u \in R_i(N, M_\omega)$ with $M_u(p) > k$;*
- *the NS is bounded $\Leftrightarrow$ no basis marking of the BCG contains the symbol ω.*

Proof.
- The first item holds as this implication is true for every marking of the coverability graph according to Propositions 2 and 5 which claims that the markings of the BCG are markings of the coverability graph.
- Suppose that p is not k-bounded. There thus exists a marking $M \in R(N, M_0)$ with $M(p) > k$. As $R(N, M_0) \subseteq BCS(N, M_0)$ according to Proposition 8, there exists a basis marking M_ω and an ω-marking $M_u \in R_i(N, M_\omega)$ such that $M_u \geq_\omega M$. Thus $M_u(p) > k$.
- For the third item, the left to right implication is once again due to Proposition 5 and the fact that the equivalence holds when considering the coverability graph as stated in Proposition 2.
  For the right to left implication, suppose that no ω appears in the BCG. Then $BCS(N, M_0)$ is finite as in every basis marking, which are also markings reachable in the NS, there is finitely many sequences of implicit transitions enabled thanks to assumption (A2). Therefore, according to Proposition 8 the reachability set $R(N, M_0)$ is finite. This implies that the NS is bounded. □

The reverse implication of the first item is false. Indeed, observe the NS of Fig. 3, the two basis markings of the BCG are $[0, 0]$ and $[\omega, 0]$, however the second place is not bounded by 0, in fact it is not bounded at all. In this respect, the BCG may not explicitly show all the informations that appears in the coverability graph. The second item shows the stronger requirement, using the implicit reach, that is needed to get the reverse implication.

Proposition 11. *Given an NS $\langle N, M_0 \rangle$ with set of implicit transitions T_i verifying Assumption (A1), a non-empty sequence σ' of explicit transitions and a marking M*

- *there exists a repetitive sequence σ with $P_e(\sigma) = \sigma'$ enabled by $M \Rightarrow$ there exists $k \in \mathbb{N}$, two basis markings M^1_ω, M^2_ω and two ω-markings $M^i_u \in R_i(N, M^i_\omega)$, $i \in \{1, 2\}$, such that:*
  - *$M \leq_\omega M^i_u$, $i \in \{1, 2\}$;*
  - *there is a path starting in M^1_ω and ending in M^2_ω in the BCG whose arcs, projected on the second component, form σ';*
  - *there is a directed cycle starting in M_ω in the BCG whose arcs, projected on the second component, form $(\sigma')^k$.*

– *there exists a directed cycle starting in M_ω in the BCG that does not pass through markings containing ω and whose arcs, projected on the second component, form σ' where M_ω is a basis marking such that $M \in R_i(N, M_\omega) \Rightarrow$ there exists a stationary sequence σ with $P_e(\sigma) = \sigma'$ enabled by M.*

Proof. – Suppose that σ is repetitive from M. Due to Proposition 8, there exists a basis marking M^0_ω and an ω-marking M^0_u with $M^0_u \geq_\omega M$ and $M^0_u \in R_i(N, M^0_\omega)$. Let $\sigma = \sigma_1 t_1 \dots \sigma_n t_n \sigma_{n+1}$ where the σ_i's are sequences of implicit transitions and the t_i's are explicit transitions. As the NS verifies (A1) and by construction of the BCG, there exists a sequence $\sigma^1 = \sigma^1_1 t_1 \dots \sigma^1_n t_n$ enabled by M^0_ω where the σ^1_i's are minimal explanations of the t_i's and ending in a basis marking M^1_ω such that there exists an ω-marking M^1_u with $M^1_u \geq_\omega M$ and $M^1_u \in R_i(N, M^1_\omega)$. This translates in the BCG into a sequence $(\pi(\sigma^1_1), t_1) \dots (\pi(\sigma^1_n), t_n)$ from M^0_ω to M^1_ω. This can be repeated, giving a family of sequences $(\sigma^j)_{j\in\mathbb{N}}$, of basis markings $(M^j_\omega)_{j\in\mathbb{N}}$ and of ω-marking $(M^j_u)_{j\in\mathbb{N}}$ such that $M^{j-1}_\omega[\sigma^j\rangle M^j_\omega$, $M^j_u \geq_\omega M$ and $M^j_u \in R_i(N, M^j_\omega)$. Due to the finite number of basis markings, there exists $k, k', k < k'$, such that $M^k_\omega = M^{k'}_\omega$. There thus exists a directed cycle starting in M^k_ω whose arcs, projected on the second component, form $P_e(\sigma)^{k'-k}$.

– Suppose that there exists a directed cycle starting in the basis marking M_ω in the BCG that does not pass through markings containing ω and whose arcs, projected on the second component, form σ'. Using the Proposition 5, M_ω is a marking of $CG_{\langle N, M_0\rangle}$. Moreover due to the construction of the BCG there exists σ such that $P_e(\sigma) = \sigma'$ and a directed cycle starting in M_ω in $CG_{\langle N, M_0\rangle}$ that does not pass through markings containing ω and whose arcs form σ. Due to Proposition 3, this implies that σ is stationary. □

A marking M is *finitely dead* if there exists a bound k such that every sequence of transitions enabled by M does not contain an explicit transition and has a length smaller than k.

Proposition 12. *Consider an NS $\langle N, M_0\rangle$ with set of implicit transitions T_i verifying Assumptions (A1) and (A2), its BCG and an explicit transition t.*

1. *Transition t is dead $\Leftrightarrow$ there is no arc labelled by t' in the BCG with $P_t(t') = t$.*
2. *Transition t is quasi-live $\Leftrightarrow$ there is an arc labelled by t' in the BCG with $P_t(t') = t$.*
3. *Transition t is live $\Rightarrow$ there is an arc labelled by t' in each ergodic component of the BCG with $P_t(t') = t$.*
4. *A basis marking M_ω in the BCG has no output arc $\Rightarrow$ any marking M with $M_\omega \geq_\omega M$ is finitely dead.*

Proof. 1. $\Rightarrow$ If an arc labelled (p, t) belongs to the BCG, then there is a basis marking M_ω by which it is enabled. Thanks to Proposition 5, this basis marking is an ω-marking of the coverability graph. Moreover due to the construction of the BCG, this implies that there is a sequence σt, with σ implicit, enabled by M_ω in $CG_{\langle N, M_0\rangle}$. Thanks to Proposition 4 this implies that t is not dead.

$\Leftarrow$ If t is not dead, then there exists a reachable marking M in the NS such that t is enabled by M. Due to Proposition 8, there thus exists an ω-marking M_u and a basis marking M_ω such that $M_u \geq_\omega M$ and $M_u \in R_i(N, M_\omega)$. Let σ such that $M_\omega[\sigma\rangle M_u$. σ is an explanation of t, there thus exists a minimal explanation σ' of t. Therefore $(\pi(\sigma'), t)$ is enabled by M_ω.

2. This item is equivalent to the previous one.
3. Suppose that t is live. According to Proposition 4, there thus exists an arc labelled t in each ergodic component of $CG_{\langle N,M_0\rangle}$. Let M_ω be a basis marking, due to Proposition 5, it is a marking of $CG_{\langle N,M_0\rangle}$ too. There thus exists a path $\sigma = \sigma_1 t_1, \ldots, \sigma_n t_n$ in $CG_{\langle N,M_0\rangle}$ enabled by M_ω with $t_n = t$. Without loss of generalities thanks to the (A1) assumption, one can suppose the sequences σ_i to be minimal explanations of t_i. By construction of the BCG, there thus exists a path $\sigma' = (\pi(\sigma_1, t_1) \ldots (\pi(\sigma_n), t_n)$ enable in M_ω in the BCG. As it is true for every marking M_ω, there is an arc whose second component is t in every ergodic component.
4. Let M_ω be a basis marking with no output arc. Let M such that $M_\omega \geq_\omega M$, suppose there exists an explicit transition t and an implicit sequence σ such that σt is enabled by M. As $M_\omega \geq_\omega M$, σt is enabled by M_ω in $CG_{\langle N,M_0\rangle}$, which would imply by construction of the BCG that there exists an outgoing arc labelled by (p, t) in M_ω for some firing vector p, which is a contradiction. Therefore any sequence enabled by M is only composed of implicit transitions. As those sequences are finite due to (A2), this means that the number of implicit transitions that can be fired is bounded. Thus M is finitely dead. □

4 Diagnosability of Unbounded Net Systems

4.1 Definition of Diagnosability

In the following, we want to use the BCG to deal with the problem of fault diagnosis where the goal is to detect the occurrence of a fault under partial observation. To this aim, we associate a well precise physical meaning to implicit, explicit, and relevant transitions. In more detail:

- Implicit transitions correspond to transitions that cannot be observed. They are called *silent* or *unobservable* and could either model a regular (nominal) behaviour or a faulty behaviour of the system.
- Conversely, explicit transitions model transitions that can be observed. Those *observable* transitions are assumed to be a regular behaviour of the system.
- The set of faulty transitions is chosen as the set of relevant transitions.

We denote the above three sets as T_u, T_o, and T_f, respectively and choose $T_e = T_o$ and $T_i = T_u$.

In simple words, we may assume that observable transitions model events whose occurrence is detected by the presence of a sensor. On the contrary, unobservable transitions correspond to events to whom no sensor is associated. Note that, in the general case, the same output signal may correspond to different

events (different transition firings). This can be easily modelled using the notion of *labeling function.* $\mathcal{L} : T \to L \cup \{\varepsilon\}$ that assigns to each transition $t \in T$ either a symbol from a given alphabet of events L (if $T \in T_o$) or the empty string ε (if $T \in T_u$). We extend naturally $\mathcal{L}$ to sequences of transitions with $\mathcal{L}(\sigma t) = \mathcal{L}(\sigma)\mathcal{L}(t)$. The observed word w of events associated with the sequence σ is $w = \mathcal{L}(\sigma)$. Note that the length of a sequence σ is always greater than or equal to the length of the corresponding word w (denoted $|w|$). In fact, if σ contains k' transitions in T_u then $|\sigma| = k' + |w|$. Given a word $w \in L^*$, we write $\mathbb{P}(w) = \sum_{\sigma \in P_e^{-1}(w)} \mathbb{P}(\sigma)$. Assuming (A2), this sum is finite.

The goal of diagnosis is to detect whether a faulty event occurred in the system. We denote by $T_f \subseteq T_u$ the set of faulty transitions. A sequence σ is faulty if there exists $t \in T_f$ such that $t \in \sigma$, otherwise it is correct. An observed word w is surely faulty (resp. correct) iff every sequence σ with $\mathcal{L}(\sigma) = w$ is faulty (resp. correct) sequences, otherwise it is ambiguous. An NS system is diagnosable iff all faults can be detected after a finite delay.

Definition 9. *An NS $\langle N, M_0 \rangle$ is diagnosable if for every faulty sequence σ enabled by M_0, there exists $n \in \mathbb{N}$ such that for all sequences $\sigma' \in T^n$ with $\sigma\sigma'$ enabled by M_0, $\mathcal{L}(\sigma\sigma')$ is surely faulty.*

Example 6. Consider again the NS in Fig. 1, where the labelling function $\mathcal{L}$ is such that $\mathcal{L}(t_1) = b, \mathcal{L}(t_2) = a, \mathcal{L}(t_3) = \mathcal{L}(t_4) = \varepsilon$ and $\mathcal{L}(t_5) = \mathcal{L}(t_6) = c$. Thus, t_3 and t_4 are unobservable. Transition t_5 being observable, the T_u-induced subnet is acyclic.

Choosing $T_f = \{t_3\}$, the infinite sequence $\sigma_f = t_1 t_2 t_3 (t_5)^3$ is faulty and its observed word bac^3 is surely faulty, so the fault can be detected here. However, the sequences $\sigma_f = t_1 t_2 t_3 (t_1)^*$ are faulty but their observed word bab^* is ambiguous as it can also correspond to the correct sequences $t_1 t_2 (t_1)^*$ too. Thus this NS is not diagnosable.

4.2 Diagnosability Analysis

Diagnosability was proven decidable [1,3]. To do so, the authors of [3] gave a characterisation of diagnosability using a tool called Verifier Net. The verifier net is obtained by a composition (related to a parallel composition of the studied NS and its $T \setminus T_f$-induced subnet with synchronisation on the observable transitions.

Definition 10. *Given an NS $\langle N, M_0 \rangle$, let $\langle N', M_0' \rangle$ be the $T \setminus T_f$-induced subnet of $\langle N, M_0 \rangle$ (prime are used to differentiate states and transitions of N' from those of N). We build the verifier net (VN) $\langle \tilde{N}, \tilde{M}_0$ of $\langle N, M_0 \rangle$ with $\tilde{N} = (\tilde{P}, \tilde{T}, \tilde{Pre}, \tilde{Post})$ where:*

- $\tilde{P} = P \cup P'$,
- $\tilde{T} = (T_o' \times T_o) \cup (T \setminus T_f \times \{\lambda\}) \cup (\{\lambda\} \times T)$,
- *for $t \in T, t' \in T' \setminus T_f, p \in P$, and $p' \in P'$, we have*
  - $\tilde{Pre}(p, (\lambda, t)) = Pre(p, t)$ *and* $\tilde{Post}(p, (\lambda, t)) = Post(p, t)$,

- $\tilde{Pre}(p', (t', \lambda)) = Pre(p', t')$ *and* $\tilde{Post}(p', (t', \lambda)) = Post(p', t')$,
- *if* $\mathcal{L}(t) = \mathcal{L}(t') \neq \varepsilon$, $\tilde{Pre}(p', (t', t)) = Pre(p', t')$ *and* $\tilde{Post}(p', (t', t)) = Post(p', t')$, $\tilde{Pre}(p, (t', t)) = Pre(p, t)$ *and* $\tilde{Post}(p, (t', t)) = Post(p, t)$.

All unspecified values are equal to 0.

Theorem 1 ([3]). *An NS* $\langle N, M_0 \rangle$ *verifying Assumption (A1) is diagnosable iff there does not exist any cycle in the coverability graph of the VN which (1) starts from an* ω*-marking reachable by a faulty sequence and (2) is associated with a repetitive sequence in the associated VN.*

We will now use this characterisation to formulate a similar one using the BCG instead of the coverability graph. A sequence of the BCG is called faulty if one of the minimal e-vector used activated a transition of T_f (i.e. the corresponding sequence belongs to $\Sigma_{min}^{T_f}$).

Theorem 2. *An NS* $\langle P, M_0 \rangle$ *verifying Assumptions (A1) and (A2) is diagnosable iff there does not exist any cycle in the BCG with relevant set of transitions* T_f *of the VN which (1) starts from a basis marking reachable by a faulty sequence and (2) is associated with a repetitive sequence in the associated VN.*

Proof. We will show that the existence of such a cycle in the BCG is equivalent to the existence of this cycle in the coverability graph.

Supposing there exists a cycle associated with a firable repetitive sequence $\sigma \in T^*$ in the associated VN that starts from a basis marking M_ω reached by a faulty sequence in the BCG with relevant set of transition T_f of the VN, then by Proposition 5, M_ω is an ω-marking of the coverability graph and by construction of the BCG, there exists a directed cycle starting in M_ω in the coverability graph whose arcs form σ.

Now suppose that there is a firable repetitive sequence $\sigma = \sigma_1 t_1 \dots \sigma_n t_n$ in the VN that is associated to a cycle starting from an ω-marking reached by a faulty sequence in the coverability graph of the VN. There thus exists a marking M of the VN such that σ is repetitive starting in M. Because of the assumption (A2), σ contains at least one observable transition. According to Proposition 11, there thus exists a basis marking M_ω and an ω-marking M_u such that $M_u \in R_i(N, M_\omega)$, $M_u \geq_\omega M$ and there is a $k \in \mathbb{N}$ and a directed cycle starting in M_ω whose arcs, projected on the second component, form $P_o(\sigma)^k$. Moreover, as M is reached by a faulty sequence $\sigma' = \sigma'_1 t'_1 \dots \sigma'_n t'_n \sigma'_{n+1}$, one can choose M_ω to be reached by a sequence that used a minimal explanation from $\Sigma_{min}^{T_f}$: if σ'_i is faulty, one can choose the minimal explanation of t_i to belong in $\Sigma_{min}^{T_f}$.

Consequently the characterisation of Theorems 1 and 2 are equivalent and can both be used to solve diagnosability. □

5 Conclusion

In this paper, we introduced the notion of basis coverability graph which provides an abstracted representation of the coverability graph. We established multiple properties of the basis coverability graph, especially how it can be used to

approximate the reachability set efficiently. We then gave an application of the basis coverability graph with the diagnosability analysis problem. We showed how the basis reachability graph can be employed to efficiently replace a previously known characterisation of the diagnosability of an unbounded NS. The logical next step would be to implement the algorithms obtained and compare their efficiency with other algorithms ([1] for example) on case studies.

References

1. Bérard, B., Haar, S., Schmitz, S., Schwoon, S.: The complexity of diagnosability and opacity verification for Petri nets. In: van der Aalst, W., Best, E. (eds.) PETRI NETS 2017. LNCS, vol. 10258, pp. 200–220. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57861-3_13
2. Boucheneb, H., Barkaoui, K.: Reducing interleaving semantics redundancy in reachability analysis of time Petri nets. ACM Trans. Embed. Comput. Syst. **12**(1), 7:1–7:24 (2013)
3. Cabasino, M.P., Giua, A., Lafortune, S., Seatzu, C.: A new approach for diagnosability analysis of Petri nets using verifier nets. IEEE Trans. Autom. Control **57**(12), 3104–3117 (2012)
4. Cabasino, M.P., Giua, A., Seatzu, C.: Fault detection for discrete event systems using Petri nets with unobservable transitions. Automatica **46**(9), 1531–1539 (2010)
5. Cabasino, M.P., Giua, A., Seatzu, C.: Introduction to Petri nets. In: Seatzu, C., Silva, M., van Schuppen, J. (eds.) Control of Discrete-Event Systems. LNCIS, vol. 433, pp. 191–211. Springer, London (2013). https://doi.org/10.1007/978-1-4471-4276-8_10
6. Cabasino, M.P., Giua, A., Pocci, M., Seatzu, C.: Discrete event diagnosis using labeled Petri nets. An application to manufacturing systems. Control Eng. Pract. **19**(9), 989–1001 (2011)
7. Giua, A., Seatzu, C., Corona, D.: Marking estimation of Petri nets with silent transitions. IEEE Trans. Autom. Control **52**(9), 1695–1699 (2007)
8. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem, vol. 1032. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-60761-7
9. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)
10. Ma, Z.Y., Tong, Y., Li, Z.W., Giua, A.: Basis marking representation of Petri net reachability spaces and its application to the reachability problem. IEEE Trans. Autom. Control **62**(3), 1078–1093 (2017)
11. Mayr, E.W.: An algorithm for the general Petri net reachability problem. In: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, STOC 1981, pp. 238–246. ACM (1981)
12. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
13. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains, part I. Theor. Comput. Sci. **13**(1), 85–108 (1981). Special Issue Semantics of Concurrent Computation
14. Reynier, P.-A., Servais, F.: Minimal coverability set for Petri nets: Karp and Miller algorithm with pruning. Fundam. Informaticae **122**(1–2), 1–30 (2013)

15. Lipton, R.: The reachability problem requires exponential space. Technical report, Yale University (1976)
16. Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D.: Diagnosability of discrete-event systems. IEEE Trans. Autom. Control **40**(9), 1555–1575 (1995)
17. Tong, Y., Li, Z., Seatzu, C., Giua, A.: Verification of state-based opacity using Petri nets. IEEE Trans. Autom. Control **62**(6), 2823–2837 (2017)
18. Valmari, A.: The state explosion problem. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 429–528. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_21

Co-finiteness and Co-emptiness of Reachability Sets in Vector Addition Systems with States

Petr Jančar[1(✉)], Jérôme Leroux[2], and Grégoire Sutre[2]

[1] Department of Computer Science, Faculty of Science, Palacký University, Olomouc, Czech Republic
petr.jancar@upol.cz

[2] University of Bordeaux, CNRS, LaBRI, UMR 5800, Talence, France
{jerome.leroux,gregoire.sutre}@labri.fr

Abstract. The coverability and boundedness problems are well-known exponential-space complete problems for vector addition systems with states (or Petri nets). The boundedness problem asks if the reachability set (for a given initial configuration) is finite. Here we consider a dual problem, the co-finiteness problem that asks if the complement of the reachability set is finite; by restricting the question we get the co-emptiness (or universality) problem that asks if all configurations are reachable.

We show that both the co-finiteness problem and the co-emptiness problem are complete for exponential space. While the lower bounds are obtained by a straightforward reduction from coverability, getting the upper bounds is more involved; in particular we use the bounds derived for reversible reachability by Leroux in 2013.

The studied problems have been motivated by a recent result for structural liveness of Petri nets; this problem has been shown decidable by Jančar in 2017 but its complexity has not been clarified. The problem is tightly related to a generalization of the co-emptiness problem for non-singleton sets of initial markings, in particular for downward closed sets. We formulate the problems generally for semilinear sets of initial markings, and in this case we show that the co-emptiness problem is decidable (without giving an upper complexity bound) and we formulate a conjecture under which the co-finiteness problem is also decidable.

1 Introduction

Context. Analysis of behavioural properties of (models of) systems is a natural and wide area of study; the decidability and complexity questions for respective properties are an important part of such research. As the most relevant for us we recall the reachability and liveness problems for Petri nets.

This work was supported by the grant GAČR:18-11193S of the Czech Grant Agency (P. Jančar) and by the grant ANR-17-CE40-0028 of the French National Research Agency ANR, project BraVAS (J. Leroux and G. Sutre).

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 184–203, 2018.
https://doi.org/10.1007/978-3-319-91268-4_10

A concrete source of motivation for us has been the recent paper [7] that answered the decidability question for structural liveness in Petri nets positively; the open status of this question was previously recalled, e.g., in [2]. It is natural to continue with studying the computational complexity of this problem. Here we contribute indirectly to this topic by studying some related naturally arising problems concerning reachability sets.

The algorithm in [7] reduces the structural liveness problem to the question if a Petri net with a downward closed set of initial markings is "universal", in the sense that every marking is reachable from the initial ones. This question has been solved by using the involved result proved in [8], namely that there is an algorithm that halts with a Presburger description of the reachability set when this set is semilinear. Since this approach is not constructive, it does not provide any complexity upper bound. This led us to consider the universality problem, which we call the *co-emptiness problem*, on its own. There is also a naturally related *co-finiteness problem* asking if a set of initial markings allows to reach all but finitely many markings; this problem can be thus seen as dual to the well-known boundedness problem that asks if the reachability set is finite.

Contributions. We formulate the co-emptiness and co-finiteness problem generally for semilinear sets of initial markings. We show that the co-emptiness problem is decidable using a reduction to [8] that is similar to the above-mentioned approach used in [7] to decide the structural liveness problem. As before, no complexity upper bound can be derived from that approach. In the case of the co-finiteness problem we are even not sure with decidability, but we formulate a conjecture under which the problem is decidable.

We then consider restrictions to the case with finite sets of initial markings and then in particular to the case with singleton sets of initial markings.

In the case of finite initial sets we show that the co-emptiness problem reduces in logarithmic space to the reachability problem. The converse reduction (reachability to co-emptiness) is left open.

In the case of singleton initial sets we show EXPSPACE-completeness for both co-emptiness and co-finiteness. This is the main technical result of the paper. While the lower bound is obtained by an easy reduction from the coverability problem (a well known EXPSPACE-complete problem, similarly as boundedness), getting the upper bound is more involved. Using the bound obtained for reversible reachability by Leroux in [9], we reduce the co-emptiness problem (with a single initial marking) to a large number of coverability questions in a large Petri net. The latter is bounded in such a way that the questions can be still answered in exponential space, using Rackoff's technique [13].

Though our results do not improve our knowledge about the complexity of structural liveness directly, we show that a related problem, namely the structural deadlock-freedom problem is tightly related (interreducible in polynomial time) with the co-emptiness problem in the case of downward closed sets of initial markings.

We have found more convenient to present our results on the model of vector addition systems with states, or shortly VASSs. This model is equivalent to Petri nets and all our results, while proved for VASSs, also hold for Petri nets.

Outline. In Sect. 2 we recall some preliminary notions, such as vector addition systems with states and semilinear sets. Section 3 defines the co-emptiness problem and the co-finiteness problem, and presents our partial decidability results for the general case and for the restriction to finite sets of initial configurations. The main result is contained in Sect. 4 where we show the EXPSPACE-completeness of co-emptiness and co-finiteness in the case with singleton sets of initial configurations. Section 5 presents two applications of the co-emptiness problem: we recall the structural liveness, and show the tight relation of structural deadlock-freedom to the co-emptiness problem with downward closed sets of initial configurations. We conclude the paper by Sect. 6.

2 Preliminaries

By $\mathbb{Z}$ we denote the set of integers, and by $\mathbb{N}$ the set $\{0, 1, 2, \dots\}$ of nonnegative integers. By $[i, j]$, where $i, j \in \mathbb{Z}$, we denote the set $\{i, i+1, \dots, j\}$ (which is empty when $i > j$).

For a vector $v \in \mathbb{Z}^d$ ($d \in \mathbb{N}$), by $v(i)$ we denote the i-th component of v. On $\mathbb{Z}^d$ we define the operations $+$, $-$ and the relations $\geq$, $\leq$ componentwise. For $v_1, v_2 \in \mathbb{Z}^d$ we thus have $v_1 + v_2 = w$ where $w(i) = v_1(i) + v_2(i)$ for all $i \in [1, d]$; we have $v_1 \leq v_2$ iff $v_1(i) \leq v_2(i)$ for all $i \in [1, d]$. For $k \in \mathbb{N}$ and $v \in \mathbb{Z}^d$ we put $k \cdot v = (k \cdot v(1), k \cdot v(2), \cdots, k \cdot v(d))$; we also write kv instead of $k \cdot v$.

Slightly abusing notation, by $(v_1, v_2, \dots, v_m)$ where $v_i \in \mathbb{Z}^{d_i}$ for $i \in [1, m]$ we do not denote an m-tuple of vectors but the corresponding vector of dimension $d = \sum_{i \in [1,m]} d_i$.

The *norm* $\|v\|$ *of a vector* $v \in \mathbb{Z}^d$ is $\max\{|v(i)|; i \in [1, d]\}$, and the *norm* $\|V\|$ *of a finite set* $V \subseteq \mathbb{Z}^d$ is $\max\{\|v\|; v \in V\}$; here we stipulate $\max \emptyset = 0$.

When the dimension d is clear from context, by $\mathbf{0}$ we denote the zero vector ($\mathbf{0}(i) = 0$ for all $i \in [1, d]$), and by $\mathbf{e}_i$ ($i \in [1, d]$) the vector satisfying $\mathbf{e}_i(i) = 1$ and $\mathbf{e}_i(j) = 0$ for all $j \in [1, d] \setminus \{i\}$.

For a set A, by A^* we denote the set of finite sequences of elements of A, and by ε we denote the empty sequence. For $w \in A^*$, $|w|$ denotes its length.

Vector Addition Systems with States (VASSs). A *vector addition system with states* (a *VASS*) is a tuple $\mathcal{V} = (d, Q, \mathcal{A}, T)$ where $d \in \mathbb{N}$ is the *dimension*, Q is the finite set of *(control) states*, $\mathcal{A} \subseteq \mathbb{Z}^d$ is the finite set of *actions*, and $T \subseteq Q \times \mathcal{A} \times Q$ is the finite set of *transitions*. We often present $t \in T$ where $t = (q, \mathbf{a}, q')$ as $q \xrightarrow{\mathbf{a}} q'$ or $t : q \xrightarrow{\mathbf{a}} q'$.

The set of *configurations* of $\mathcal{V} = (d, Q, \mathcal{A}, T)$ is the set $Q \times \mathbb{N}^d$; we rather present a configuration (q, v) as $q(v)$ (where $q \in Q$, $v \in \mathbb{N}^d$). For actions $\mathbf{a} \in \mathcal{A}$ we define relations $\xrightarrow{\mathbf{a}}_{\mathcal{V}}$ on the set $Q \times \mathbb{N}^d$ of configurations by putting

$$q(v) \xrightarrow{\mathbf{a}}_{\mathcal{V}} q'(v') \text{ if } q \xrightarrow{\mathbf{a}} q' \text{ is a transition in } T \text{ and } v' = v + \mathbf{a}.$$

Hence for a transition $q \xrightarrow{\mathbf{a}} q'$ and $v \in \mathbb{N}^d$ we have $q(v) \xrightarrow{\mathbf{a}}_{\mathcal{V}} q'(v+\mathbf{a})$ iff $v+\mathbf{a} \geq \mathbf{0}$.

Relations $\xrightarrow{\mathbf{a}}_{\mathcal{V}}$ are naturally extended to relations $\xrightarrow{\alpha}_{\mathcal{V}}$ for $\alpha \in \mathcal{A}^*$; we write just $\xrightarrow{\alpha}$ instead of $\xrightarrow{\alpha}_{\mathcal{V}}$ when $\mathcal{V}$ is clear from context. The extension is defined inductively: we put $q(v) \xrightarrow{\varepsilon} q(v)$; if $q(v) \xrightarrow{\mathbf{a}} q'(v')$ and $q'(v') \xrightarrow{\alpha} q''(v'')$, then $q(v) \xrightarrow{\mathbf{a}\alpha} q''(v'')$. We note that $q(v) \xrightarrow{\alpha} q'(v')$ where $\alpha = \mathbf{a}_1\mathbf{a}_2 \cdots \mathbf{a}_m$ implies that $v' = v + \sum_{i\in[1,m]} \mathbf{a}_i$. We also note the *monotonicity*:

$$\text{if } q(v) \xrightarrow{\alpha} q'(v'), \text{ then for any } \bar{v} \geq v \text{ we have } q(\bar{v}) \xrightarrow{\alpha} q'(v' + \bar{v} - v).$$

Reachability Sets. Given a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$, by $q(v) \xrightarrow{*}_{\mathcal{V}} q'(v')$, or by $q(v) \xrightarrow{*} q'(v')$ when $\mathcal{V}$ is clear from context, we denote that $q'(v')$ is *reachable from* $q(v)$, i.e., that $q(v) \xrightarrow{\alpha} q'(v')$ for some $\alpha \in \mathcal{A}^*$. The *reachability set for an* (initial) *configuration* $q(v)$ is the set

$$\big[q(v)\big\rangle_{\mathcal{V}} = \{q'(v') \mid q(v) \xrightarrow{*}_{\mathcal{V}} q'(v')\}.$$

For a *set* $C \subseteq Q \times \mathbb{N}^d$ *of (initial) configurations* we put

$$\big[C\big\rangle_{\mathcal{V}} = \bigcup_{q(v)\in C} \big[q(v)\big\rangle_{\mathcal{V}}.$$

We also write just $\big[q(v)\big\rangle$ and $\big[C\big\rangle$ when $\mathcal{V}$ is clear from context.

We write $q(v) \xrightarrow{*} C$ if there is $q'(v') \in C$ such that $q(v) \xrightarrow{*} q'(v')$; similarly $C \xrightarrow{*} q(v)$ if there is $q'(v') \in C$ such that $q'(v') \xrightarrow{*} q(v)$.

Semilinear Sets of Configurations. A *set* $C \subseteq Q \times \mathbb{N}^d$ is *linear* if

$$C = \{q(b + n_1p_1 + \cdots + n_kp_k) \mid n_1, \ldots, n_k \in \mathbb{N}\}$$

for some $q \in Q$, $k \in \mathbb{N}$, and $b, p_1, \ldots, p_k \in \mathbb{N}^d$. A *set* $C \subseteq Q \times \mathbb{N}^d$ is *semilinear* if $C = L_1 \cup L_2 \cup \cdots \cup L_m$ for some $m \in \mathbb{N}$ and linear sets L_j, $j \in [1, m]$.

We recall that semilinear sets correspond to the sets definable in Presburger arithmetic [4].

Vector Addition Systems (VASs). A *vector addition system* (*VAS*) is a VASS $(d, Q, \mathcal{A}, T)$ where Q is a singleton. In this case the single control state plays no role, in fact; it is thus natural to view a VAS as a pair $\mathcal{U} = (d, \mathcal{A})$ for a finite set $\mathcal{A} \subseteq \mathbb{Z}^d$. The configurations are here simply $v \in \mathbb{N}^d$, and for $\mathbf{a} \in \mathcal{A}$ we have

$$v \xrightarrow{\mathbf{a}} v' \text{ iff } v' = v + \mathbf{a} \text{ (for any } v, v' \in \mathbb{N}^d).$$

We write $\big[v\big\rangle_{\mathcal{U}}$, or just $\big[v\big\rangle$, for the reachability set of v. For a VAS the terms "action" and "transition" are identified.

Binary and Unary Presentations. Instances of the problems that we will consider comprise VASSs and (presentations of semilinear sets of) configurations. We implicitly assume that the numbers in the respective vectors are presented in binary. When giving a complexity lower bound, we will explicitly refer to a unary presentation to stress the substance of the lower bound.

3 Co-finiteness and Co-emptiness of Reachability Sets

Now we introduce the two main problems considered in this paper.

Co-finiteness. The *co-finiteness problem*:

Instance: a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and a (presentation of a) semilinear set $C \subseteq Q \times \mathbb{N}^d$.
Question: is $\big[C\big\rangle$ co-finite, i.e., is the set $(Q \times \mathbb{N}^d) \setminus \big[C\big\rangle$ finite?

Co-emptiness (or Universality). By narrowing the co-finiteness question we get the *co-emptiness problem*:

Instance: a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and a (presentation of a) semilinear set $C \subseteq Q \times \mathbb{N}^d$.
Question: is $\big[C\big\rangle$ co-empty, i.e., is $\big[C\big\rangle = Q \times \mathbb{N}^d$?

We note that co-emptiness can be also naturally called *universality*.

3.1 Decidability of the General Problems

We recall the classical *reachability problem*, defined as follows:

Instance: a VASS $\mathcal{V}$ and two configurations $q(v), q'(v')$.
Question: is $q(v) \xrightarrow{*}_{\mathcal{V}} q'(v')$?

The problem is decidable [12] but its complexity remains elusive; the problem is known to be EXPSPACE-hard [11], and the best known upper bound is non-primitive recursive [10].

By adding the acceleration techniques in [8], and decidability of Presburger arithmetic, it is straightforward to derive decidability of the co-emptiness problem. We first recall a crucial fact.

Theorem 1 (reformulation of Lemma XI.1 of [8]). *Given a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and a semilinear set C of configurations, for every semilinear set $D \subseteq \big[C\big\rangle_{\mathcal{V}}$ there is a sequence $\alpha_1, \ldots, \alpha_k$ of words in $\mathcal{A}^*$ such that for every $q(v) \in D$ we have*

$$C \xrightarrow{\alpha_1^{n_1} \ldots \alpha_k^{n_k}}_{\mathcal{V}} q(v)$$

for some $n_1, \ldots, n_k \in \mathbb{N}$.

We thus deduce that the co-emptiness problem can be decided by the following two procedures that are executed concurrently:

- One procedure systematically searches for some configuration $q(v)$ such that $q(v) \notin \lceil C \rangle$ which is verified by using an algorithm deciding (non)reachability; this search succeeds iff $\lceil C \rangle$ is not co-empty.
- The other procedure systematically searches for some words $\alpha_1, \ldots, \alpha_k$ such that for every configuration $q(v)$ there are $n_1, \ldots, n_k$ in $\mathbb{N}$ such that $C \xrightarrow{\alpha_1^{n_1} \ldots \alpha_k^{n_k}} q(v)$. This property (of $\alpha_1, \ldots, \alpha_k$) can be formulated in Presburger arithmetic and is thus decidable. The search succeeds iff $\lceil C \rangle$ is co-empty.

However, the complexity of the co-emptiness problem is still open. In fact, we even have no reduction from or to the reachability problem.

The decidability status of the co-finiteness problem is not clear. We show how to solve the problem under a conjecture. Let us first introduce the notion of inductive set. A set of configurations D of a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ is *inductive* if for every configuration $q(v)$ in D and every transition $t : q \xrightarrow{\mathbf{a}} q'$ in T such that $v + \mathbf{a} \geq \mathbf{0}$, we have $q'(v + \mathbf{a}) \in D$. We observe that if an inductive set D contains a set of initial configurations C then $\lceil C \rangle \subseteq D$. Moreover, we can effectively decide if a semilinear set D is inductive. We introduce the following conjecture.

Conjecture 2. Given a VASS $\mathcal{V}$ and a semilinear set C of configurations, if $\lceil C \rangle_{\mathcal{V}}$ is co-infinite (i.e., not co-finite), then there is an inductive semilinear set D such that $C \subseteq D$ (hence also $\lceil C \rangle_{\mathcal{V}} \subseteq D$) and D is co-infinite.

Under that conjecture, the co-finiteness problem can be also decided by two algorithmic procedures executed concurrently:

- One procedure systematically searches for some inductive co-infinite semilinear set D that contains C; this search succeeds iff $\lceil C \rangle$ is co-infinite (under the conjecture).
- The other procedure systematically searches for some words $\alpha_1, \ldots, \alpha_k$ and a natural number n, such that for every configuration $q(v)$ with $\|v\| \geq n$ there are $n_1, \ldots, n_k$ in $\mathbb{N}$ satisfying $C \xrightarrow{\alpha_1^{n_1} \ldots \alpha_k^{n_k}} q(v)$. This property (of $\alpha_1, \ldots, \alpha_k$ and n) can be formulated in Presburger arithmetic, and is thus decidable. The search succeeds iff $\lceil C \rangle$ is co-finite thanks to Theorem 1. Indeed, when the reachability set $\lceil C \rangle$ is co-finite then it is semilinear, and we can apply Theorem 1 with $D = \lceil C \rangle$.

Hence we have derived:

Theorem 3. *The co-emptiness problem is decidable.*
The co-finiteness problem is decidable when assuming validity of Conjecture 2.

3.2 Finitely Many Initial Configurations

As already mentioned, we have no complexity upper bound for the (decidable) co-emptiness problem in our general form. In the rest of this section we focus on the *FMIC co-emptiness problem*, and the *FMIC co-finiteness problem*, where "FMIC" refers to "Finitely Many Initial Configurations". We give the result captured by Theorem 5.

Lemma 4. *Given a VASS* $\mathcal{V} = (d, Q, \mathcal{A}, T)$ *and any set* $C \subseteq Q \times \mathbb{N}^d$*, we have*

$$\big[C\big\rangle = Q \times \mathbb{N}^d \text{ iff } \big[C\big\rangle \supseteq D_1 \cup D_2$$

where $D_1 = \{q(v + \mathbf{e}_i) \mid q(v) \in C, i \in [1, d]\}$ *and* $D_2 = \{q(\boldsymbol{0}) \mid q \in Q\}$.

Proof. If $\big[C\big\rangle = Q \times \mathbb{N}^d$, then we trivially have $\big[C\big\rangle \supseteq D_1 \cup D_2$.

Let us now assume $\big[C\big\rangle \supseteq D_1$. We show that

$$q(v) \in \big[C\big\rangle \text{ implies } q(v + \mathbf{e}_i) \in \big[C\big\rangle \tag{1}$$

(for all $q \in Q$, $v \in \mathbb{N}^d$, $i \in [1, d]$). Indeed, if $q_0(v_0) \xrightarrow{*} q(v)$ for some $q_0(v_0) \in C$, then $\big[C\big\rangle \ni q_0(v_0 + \mathbf{e}_i)$ since $\big[C\big\rangle \supseteq D_1$ and $q_0(v_0 + \mathbf{e}_i) \xrightarrow{*} q(v + \mathbf{e}_i)$ by monotonicity; hence $q(v + \mathbf{e}_i) \in \big[C\big\rangle$.

If, moreover, $\big[C\big\rangle \supseteq D_2$, then by (1) we get $q(v) \in \big[C\big\rangle$ for all $q \in Q$, $v \in \mathbb{N}^d$. □

Theorem 5. *The FMIC co-emptiness problem is logspace reducible to the reachability problem.*

Proof. Given a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and a finite set $C \subseteq Q \times \mathbb{N}^d$, deciding if $\big[C\big\rangle = Q \times \mathbb{N}^d$ boils down to verifying if each configuration in the finite set

$$D_1 \cup D_2 = \{q_1(v_1), \ldots, q_k(v_k)\}$$

defined in Lemma 4 is reachable from (a configuration in) C.

Let $\mathcal{V}'$ arise from $\mathcal{V}$ by adding a fresh control state q_0 and transitions $q_0 \xrightarrow{v} q$ for all $q(v) \in C$. Hence

$$\big[C\big\rangle_{\mathcal{V}} \supseteq D_1 \cup D_2 \text{ iff } q_0(\mathbf{0}) \xrightarrow{*}_{\mathcal{V}'} q_j(v_j) \text{ for all } j \in [1, k].$$

Let us now consider a VASS $\mathcal{V}''$ of dimension kd comprising k disjoint copies of $\mathcal{V}'$ (each copy works on its own counters); let (q, j) denote the control state q of $\mathcal{V}'$ in the j-th copy ($j \in [1, k]$).

Finally, we let $\mathcal{V}'''$ arise from $\mathcal{V}''$ by adding transitions $(q_j, j) \xrightarrow{\mathbf{0}} (q_0, j + 1)$, for $j \in [1, k - 1]$. We observe that

$$q_0(\mathbf{0}) \xrightarrow{*}_{\mathcal{V}'} q_j(v_j) \text{ for all } j \in [1, k] \text{ iff } (q_0, 1)(\mathbf{0}) \xrightarrow{*}_{\mathcal{V}'''} (q_k, k)(v_1, \ldots, v_k).$$

We have thus shown the claimed logspace reduction. □

We leave open the question if the FMIC co-finiteness problem can be similarly reduced to the reachability problem. Another open question is if reachability can be reduced to FMIC co-finiteness or FMIC co-emptiness. In the next section, we characterize the complexity of both problems for the case of single initial configurations.

4 Single Initial Configurations

In this section we restrict our attention to the *SIC co-emptiness problem* and the *SIC co-finiteness problem* where SIC refers to "Single Initial Configuration"; the problem instances are thus restricted so that the given sets C are singletons ($C = \{q_0(v_0)\}$). In the rest of this section we prove the following theorem.

Theorem 6. *Both the SIC co-finiteness problem and the SIC co-emptiness problem are EXPSPACE-complete.*

We recall that the integers in the problem instances are presented in binary. Nevertheless the lower bound will be shown already for *unary VASs* (hence with no control states and with a unary presentation of integers).

We first recall two well-known EXPSPACE-complete problems for VASSs where the lower bound also holds for unary VASs.

Coverability. The *coverability problem*:

Instance: a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$, $q_0, q_1 \in Q$, $v_0, v_1 \in \mathbb{N}^d$;
Question: is $q_0(v_0) \xrightarrow{*} q_1(\bar{v}_1)$ for some $\bar{v}_1 \geq v_1$?

Boundedness. The *boundedness problem*:

Instance: a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$, $q_0 \in Q$, $v_0 \in \mathbb{N}^d$;
Question: is $\big[q_0(v_0)\big\rangle$ finite ?

The EXPSPACE-hardness results follow from [11] (see also, e.g. [3]), the upper bounds follow from [13]. A generalization of [13], extending a class of problems known to be in EXPSPACE, was given in [14], which was later corrected in [1].

4.1 EXPSPACE-Hardness

Showing the hardness part of Theorem 6 is relatively straightforward; we reduce coverability in unary VASs (which we recalled as an EXPSPACE-complete problem) to both SIC co-finiteness and SIC co-emptiness by the following lemma.

Lemma 7. *Given a unary VAS $\mathcal{U} = (d, \mathcal{A})$ and $v_0, v_1 \in \mathbb{N}^d$, there is a logspace construction yielding a unary VAS $\mathcal{U}' = (d+1, \mathcal{A}')$ and $v_0' \in \mathbb{N}^{d+1}$ such that:*

(a) if $v_0 \xrightarrow{*}_{\mathcal{U}} \bar{v}_1$ for some $\bar{v}_1 \geq v_1$, then $\big[v_0'\big\rangle_{\mathcal{U}'} = \mathbb{N}^{d+1}$;
(b) otherwise (when $v_0 \xrightarrow{*}_{\mathcal{U}} w$ implies $w \not\geq v_1$) the set $\mathbb{N}^{d+1} \setminus \big[v_0'\big\rangle_{\mathcal{U}'}$ is infinite.

Proof. Let us assume a unary VAS $\mathcal{U} = (d, \mathcal{A})$ and vectors $v_0, v_1 \in \mathbb{N}^d$. We consider $\mathcal{U}' = (d+1, \mathcal{A}')$ and $v_0' = (v_0, 0)$ where

$$\mathcal{A}' = \{(\mathbf{a}, 0) \mid \mathbf{a} \in \mathcal{A}\} \cup \{\mathbf{b}_1, \mathbf{b}_2\} \cup \{\mathbf{c}_j \mid j \in [1, d]\} \cup \{-\mathbf{e}_j \mid j \in [1, d]\}$$

$$\text{for } \mathbf{b}_1 = (-v_1, 2),\ \mathbf{b}_2 = (v_0, -1),\ \mathbf{c}_j = \mathbf{e}_j - \mathbf{e}_{d+1}.$$

It suffices to verify that the points (a) and (b) are satisfied (for $\mathcal{U}'$ and v_0'):

(a) Suppose $v_0 \xrightarrow{\alpha}_{\mathcal{U}} \bar{v}_1$ for some $\bar{v}_1 \geq v_1$ and $\alpha = \mathbf{a}_1\mathbf{a}_2\cdots\mathbf{a}_m$. For $\alpha' = (\mathbf{a}_1, 0)(\mathbf{a}_2, 0)\cdots(\mathbf{a}_m, 0)$, in $\mathcal{U}'$ we then have

$$(v_0, 0) \xrightarrow{\alpha'} (\bar{v}_1, 0) \xrightarrow{\mathbf{b}_1} (\bar{v}_1 - v_1, 2) \xrightarrow{\mathbf{b}_2} (v_0 + \bar{v}_1 - v_1, 1).$$

By monotonicity, for any $k \in \mathbb{N}$ we have

$$v_0' = (v_0, 0) \xrightarrow{(\alpha'\mathbf{b}_1\mathbf{b}_2)^k} w_k = (v_0 + k(\bar{v}_1 - v_1), k);$$

hence $w_k(d+1) = k$. For any $w \in \mathbb{N}^{d+1}$ and the sum $k = \sum_{j\in[1,d+1]} w(j)$ we have $w_k \xrightarrow{*} w$; indeed, in w_k we can first empty (i.e., set to zero) all components $j \in [1, d]$ by using actions $-\mathbf{e}_j$ ($j \in [1, d]$), and then distribute the k tokens from component $d+1$ by the actions $\mathbf{c}_j$ so that w is reached. Hence $\big[v_0'\big\rangle_{\mathcal{U}'} = \mathbb{N}^{d+1}$.

(b) Suppose there is no $\bar{v}_1 \geq v_1$ such that $v_0 \xrightarrow{*}_{\mathcal{U}} \bar{v}_1$. Then for any $w \in \big[v_0'\big\rangle_{\mathcal{U}'}$ we have $w \not\geq (v_1, 0)$ and $w(d+1) = 0$, since the actions $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{c}_j$ are dead (they cannot get enabled from v_0'); indeed, by monotonicity the actions $-\mathbf{e}_j$ cannot help to cover $(v_1, 0)$ from v_0'. Hence the set $\mathbb{N}^{d+1} \setminus \big[v_0'\big\rangle_{\mathcal{U}'}$ is infinite. □

4.2 EXPSPACE-Membership

We now prove the EXPSPACE-membership claimed by Theorem 6. This is more involved; besides a closer look at the results in [13], we will also use the following result from [9], from which we derive Lemma 9.

Theorem 8 ([9]). *Given a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and two configurations $q_0(v_0)$ and $q_1(v_1)$ reachable one from the other (i.e., $q_0(v_0) \xrightarrow{*} q_1(v_1) \xrightarrow{*} q_0(v_0)$), there is a word $\alpha \in \mathcal{A}^*$ such that*

(a) $q_0(v_0) \xrightarrow{\alpha} q_1(v_1)$, and
(b) $|\alpha| \leq 6 \cdot (d+3)^2 \cdot x^{45(d+3)^{d+5}}$ where $x = 1 + 2|Q| + 2\|\mathcal{A}\| + 2\|v_0\| + \|v_1\|$.

Proof. Theorem 10.1 of [9] states that for every pair (v_0', v_1') of configurations of a VAS $(p, \mathcal{A}')$ that are reachable one from the other there is a word $\alpha' \in (\mathcal{A}')^*$ such that:

$$v_0' \xrightarrow{\alpha'} v_1' \quad \text{and} \quad |\alpha'| \leq 17p^2y^{15p^{p+2}}$$

where $y = (1 + 2\|\mathcal{A}'\|)(1 + \|v_0'\| + \|v_1' - v_0'\|)$. We extend this result to a VASS $(d, Q, \mathcal{A}, T)$ by encoding it as a VAS $(p, \mathcal{A}')$ using [6, Lemma 2.1]. With this encoding, $p = d+3$, $\|\mathcal{A}'\| \leq \max\{\|\mathcal{A}\|, |Q|\cdot(|Q|-1)\}$ and the encodings of $q_0(v_0)$ and $q_1(v_1)$ provide vectors v_0', v_1' satisfying $\|v_0'\| \leq \|v_0\| + |Q|$ and $\|v_1' - v_0'\| = \|v_1 - v_0\| \leq \|v_1\| + \|v_0\|$. It follows that $(1 + 2\|\mathcal{A}'\|) \leq x^2$ and $(1 + \|v_0'\| + \|v_1' - v_0'\|) \leq x$. Thus y is bounded by x^3. Finally, since the effect of an action of the VASS is simulated by three actions of the simulating VAS, we deduce that there exists a word $\alpha \in \mathcal{A}^*$ such that $q_0(v_0) \xrightarrow{\alpha} q_1(v_1)$ and such that $|\alpha| \leq \frac{1}{3}|\alpha'|$. We derive the bound on $|\alpha|$ by observing that $\frac{17}{3} \leq 6$. □

Pumpability of Components. Given a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$, we say that *component* $i \in [1, d]$ is *pumpable in* $q(v)$ if $q(v) \xrightarrow{*} q(v + k\mathbf{e}_i)$ for some $k \geq 1$.

Lemma 9. *For any VASS* $\mathcal{V} = (d, Q, \mathcal{A}, T)$ *and any* $q \in Q$, $v \in \mathbb{N}^d$, $i \in [1, d]$ *where component* i *is pumpable in* $q(v)$ *there is* $\alpha \in \mathcal{A}^*$ *such that*

(a) $q(v) \xrightarrow{\alpha} q(v + k\mathbf{e}_i)$ *for some* $k \geq 1$, *and*
(b) $|\alpha| \leq 6 \cdot (d+3)^2 \cdot x^{45(d+3)^{d+5}}$ *where* $x = 2 + 2|Q| + 2\|\mathcal{A}\| + 3\|v\|$.

The trivial fact $k \leq |\alpha| \cdot \|\mathcal{A}\|$ *thus also yields a double-exponential bound on* k.

Proof. We consider a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and assume $q(v) \xrightarrow{*}_{\mathcal{V}} q(v + k\mathbf{e}_i)$ where $k \geq 1$. For the VASS $\mathcal{V}'$ arising from $\mathcal{V}$ by adding (action $-\mathbf{e}_i$ and) the transition $q \xrightarrow{-\mathbf{e}_i} q$ we get

$$q(v) \xrightarrow{*} q(v + k\mathbf{e}_i) \xrightarrow{-\mathbf{e}_i} \cdots \xrightarrow{-\mathbf{e}_i} q(v + \mathbf{e}_i) \xrightarrow{-\mathbf{e}_i} q(v);$$

hence $q(v)$ and $q(v + \mathbf{e}_i)$ are reachable one from the other (they are in the reversible-reachability relation) in $\mathcal{V}'$. Using Theorem 8, we derive that

$$q(v) \xrightarrow{\alpha}_{\mathcal{V}'} q(v + \mathbf{e}_i) \tag{2}$$

for some $\alpha \in (\mathcal{A} \cup \{-\mathbf{e}_i\})^*$ that is bounded as in the point (b) of the claim.

If α in (2) is $\mathbf{a}_1\mathbf{a}_2 \cdots \mathbf{a}_m$, then there are states $q_1, q_2, \ldots, q_{m-1}$ such that

$$q(v) \xrightarrow{\mathbf{a}_1} q_1(v_1) \xrightarrow{\mathbf{a}_2} q_2(v_2) \xrightarrow{\mathbf{a}_3} \cdots q_{m-1}(v_{m-1}) \xrightarrow{\mathbf{a}_m} q(v + \mathbf{e}_i) \tag{3}$$

for the corresponding v_j ($j \in [1, m-1]$). We can view (3) as a sequence of transitions; let $\ell \geq 0$ be the number of occurrences of the transition $q \xrightarrow{-\mathbf{e}_i} q$ in (3). Due to monotonicity, we can omit these occurrences and keep performability: we get

$$q(v) \xrightarrow{\mathbf{a}_{i_1}\mathbf{a}_{i_2}\cdots\mathbf{a}_{i_{m-\ell}}}_{\mathcal{V}} q(v + (\ell+1)\mathbf{e}_i) \tag{4}$$

for the sequence $\mathbf{a}_{i_1}\mathbf{a}_{i_2} \cdots \mathbf{a}_{i_{m-\ell}}$ arising from $\mathbf{a}_1\mathbf{a}_2 \cdots \mathbf{a}_m$ by omitting the respective ℓ occurrences of $-\mathbf{e}_i$. The proof is thus finished. □

We derive the following important corollary:

Corollary 10. *There is an exponential-space algorithm that, given a VASS* $\mathcal{V} = (d, Q, \mathcal{A}, T)$ *and* $q(v)$, *decides if all components* $i \in [1, d]$ *are pumpable in* $q(v)$, *and in the positive case provides an (at most double-exponential) number* $n \geq 1$ *such that* $q(v) \xrightarrow{*} q(v + n\mathbf{e}_i)$ *for each* $i \in [1, d]$.

Proof. It suffices to consider a nondeterministic algorithm trying to find, for each $i \in [1, d]$ separately, α_i with length bounded as in Lemma 9 such that $q(v) \xrightarrow{\alpha_i} q(v + k_i\mathbf{e}_i)$ for some $k_i \geq 1$. The algorithm just traverses along (a guessed bounded) α_i, keeping only the current configuration in memory; hence exponential space is sufficient.

By monotonicity, $q(v) \xrightarrow{*} q(v + k_i\mathbf{e}_i)$ implies that $q(v) \xrightarrow{*} q(v + xk_i\mathbf{e}_i)$ for all $x \geq 1$. Hence if $k_i \geq 1$ for all $i \in [1, d]$ are found, then the least common multiple (or even simply the product) of all k_i, $i \in [1, d]$, can be taken as the claimed number n. □

Before giving the algorithm deciding SIC co-emptiness we introduce some useful natural notions, namely a notion of "reversing a VASS" (letting its computations run backwards), and a notion of "transforming a VASS modulo n" (where the component-values are divided by n while the remainders are kept in the control states).

Reversed VASS. To a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ we associate its *reversed VASS*

$$\mathcal{V}^{\leftarrow} = (d, Q, -\mathcal{A}, T^{\leftarrow})$$

where $-\mathcal{A} = \{-\mathbf{a} \mid \mathbf{a} \in \mathcal{A}\}$ and $T^{\leftarrow} = \{q' \xrightarrow{-\mathbf{a}} q \mid q \xrightarrow{\mathbf{a}} q' \text{ is in } T\}$.
The next proposition can be easily verified by induction on m.

Proposition 11. *For any VASS $\mathcal{V}$ and $m \geq 1$, we have*

$$q_0(v_0) \xrightarrow{\mathbf{a}_1} q_1(v_1) \xrightarrow{\mathbf{a}_2} q_2(v_2) \xrightarrow{\mathbf{a}_3} \cdots q_{m-1}(v_{m-1}) \xrightarrow{\mathbf{a}_m} q_m(v_m) \text{ in } \mathcal{V}$$
$$\text{iff}$$
$$q_m(v_m) \xrightarrow{-\mathbf{a}_m} q_{m-1}(v_{m-1}) \xrightarrow{-\mathbf{a}_{m-1}} \cdots q_1(v_1) \xrightarrow{-\mathbf{a}_1} q_0(v_0) \text{ in } \mathcal{V}^{\leftarrow}.$$

Modulo-n VASS. Given a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and $n \geq 1$, we put

$$\mathcal{V}_{(n)} = (d, Q \times \{0, 1, \ldots, n-1\}^d, \mathcal{A}', T_{(n)})$$

where $T_{(n)}$ arises as follows:
each transition $q \xrightarrow{\mathbf{a}} q'$ in T and each $u \in \{0, 1, \ldots, n-1\}^d$ determines

$$\text{the transition } (q, u) \xrightarrow{\mathbf{a}'} (q', u') \text{ in } T_{(n)}$$

where u' and $\mathbf{a}'$ are the unique vectors such that $u + \mathbf{a} = u' + n\mathbf{a}'$ and $u' \in \{0, 1, \ldots, n-1\}^d$. The set $\mathcal{A}'$ is simply $\{\mathbf{a}' \mid ((q, u) \xrightarrow{\mathbf{a}'} (q', u')) \in T_{(n)}\}$.

The next proposition is again easily verifiable by induction on m.

Proposition 12. *For any VASS $\mathcal{V}$, $n \geq 1$, and $m \geq 1$, we have*

$$q_0(v_0) \xrightarrow{\mathbf{a}_1} q_1(v_1) \xrightarrow{\mathbf{a}_2} q_2(v_2) \xrightarrow{\mathbf{a}_3} \cdots q_{m-1}(v_{m-1}) \xrightarrow{\mathbf{a}_m} q_m(v_m) \text{ in } \mathcal{V}$$
$$\text{iff}$$
$$(q_0, u_0)(v'_0) \xrightarrow{\mathbf{a}'_1} (q_1, u_1)(v'_1) \xrightarrow{\mathbf{a}'_2} \cdots \xrightarrow{\mathbf{a}'_m} (q_m, u_m)(v'_m) \text{ in } \mathcal{V}_{(n)}$$

where $u_j + nv'_j = v_j$ for every $j \in [0, m]$ (and $u_{j-1} + \mathbf{a}_j = u_j + n\mathbf{a}'_j$ for every $j \in [1, m]$).

Algorithm Deciding SIC Co-emptiness. We define the following algorithm.

Algorithm Alg-Co-Empt

Input: a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and a configuration $q_0(v_0)$.

Output: YES if $\big[q_0(v_0)\big\rangle = Q \times \mathbb{N}^d$, and NO otherwise.

1. Check if each component $i \in [1,d]$ is pumpable in $q_0(v_0)$, and in the positive case compute an (at most double-exponential) number n as described in Corollary 10 (hence $q_0(v_0) \xrightarrow{*} q_0(v_0 + n\mathbf{e}_i)$ for each $i \in [1,d]$).
   In the negative case (when some component is not pumpable) **return** NO.
2. Let $\mathcal{V}'$ be the VASS $\mathcal{V}' = (\mathcal{V}^{\leftarrow})_{(n)} = (d, Q \times \{0,1,\ldots,n-1\}^d, \mathcal{A}', T')$ {i.e., the reversed VASS modulo n, where n is computed in the point 1}.
   Create the configuration $(q_0,u_0)(v_0')$ of $\mathcal{V}'$ corresponding to the configuration $q_0(v_0)$ of $\mathcal{V}$ (hence $v_0 = u_0 + nv_0'$).
3. For each control state (q,u) of $\mathcal{V}'$ check if $(q,u)(\mathbf{0})$ covers $(q_0,u_0)(v_0')$ (in $\mathcal{V}'$), i.e., if $(q,u)(\mathbf{0}) \xrightarrow{*}_{\mathcal{V}'} (q_0,u_0)(\bar{v})$ for some $\bar{v} \geq v_0'$.
   If the answer is negative for some (q,u), then **return** NO, otherwise (when all $(q,u)(\mathbf{0})$ cover $(q_0,u_0)(v_0')$) **return** YES.

Correctness and Exponential-Space Complexity of Alg-Co-Empt

Lemma 13. *Algorithm* Alg-Co-Empt *satisfies its specification (i.e., returns YES if* $\big[q_0(v_0)\big\rangle = Q \times \mathbb{N}^d$*, and NO otherwise).*

Proof. If Alg-Co-Empt, when given $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and $q_0(v_0)$, returns NO in the point 1, then for some $i \in [1,d]$ we have $q_0(v_0) \not\xrightarrow{*} q_0(v_0 + x\mathbf{e}_i)$ for all $x \geq 1$; therefore the set $(Q \times \mathbb{N}^d) \setminus \big[q_0(v_0)\big\rangle_{\mathcal{V}}$ is nonempty and even infinite.

Suppose now that the test in the point 1 has been positive, and a respective number n has been computed.

Assume first that $\big[q_0(v_0)\big\rangle_{\mathcal{V}} = Q \times \mathbb{N}^d$ and let us show that the algorithm returns YES. Let (q,u) be a control state of $\mathcal{V}' = (\mathcal{V}^{\leftarrow})_{(n)}$. Since $\big[q_0(v_0)\big\rangle_{\mathcal{V}} = Q \times \mathbb{N}^d$, we have $q_0(v_0) \xrightarrow{*}_{\mathcal{V}} q(u)$. It follows that $(q,u)(\mathbf{0}) \xrightarrow{*}_{\mathcal{V}'} (q_0,u_0)(v_0')$, which also entails that $(q,u)(\mathbf{0})$ covers $(q_0,u_0)(v_0')$ in $\mathcal{V}'$. We have proved that the algorithm returns YES.

Conversely, we assume that the algorithm returns YES and we prove that $\big[q_0(v_0)\big\rangle_{\mathcal{V}} = Q \times \mathbb{N}^d$. Let $q(v)$ be a configuration of $\mathcal{V}$ and let $(q,u)(v')$ be the corresponding configuration in $\mathcal{V}'$, i.e., $v = u + nv'$. Since $(q,u)(\mathbf{0})$ covers $(q_0,u_0)(v_0')$, there exists $\bar{v}_0' \geq v_0'$ such that $(q,u)(\mathbf{0}) \xrightarrow{*}_{\mathcal{V}'} (q_0,u_0)(\bar{v}_0')$. It follows that $q_0(u_0 + n\bar{v}_0') \xrightarrow{*}_{\mathcal{V}} q(u)$. By monotonicity, we derive that

$$q_0(u_0 + n\bar{v}_0' + nv') \xrightarrow{*}_{\mathcal{V}} q(u + nv') = q(v).$$

By the definition of n, we get

$$q_0(v_0) \xrightarrow{*}_{\mathcal{V}} q_0(v_0 + n(\bar{v}_0' - v_0') + nv') = q_0(u_0 + n\bar{v}_0' + nv').$$

We have proved that $q_0(v_0) \xrightarrow{*}_{\mathcal{V}} q(v)$, and thus $\big[q_0(v_0)\big\rangle_{\mathcal{V}} = Q \times \mathbb{N}^d$. □

We still need to show that Alg-Co-Empt works in exponential space (Lemma 15). We first give a straightforward extension to VASSs of a result formulated in [13] for VASs.

Proposition 14. *For any VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and any configurations $q_0(v_0)$ and $q_1(v_1)$, if $q_0(v_0) \xrightarrow{*} q_1(v_1)$ then $q_0(v_0) \xrightarrow{\alpha} q_1(\bar{v}_1)$ for some $\bar{v}_1 \geq v_1$ and $\alpha \in \mathcal{A}^*$ such that $|\alpha| < x^{(d+1)!}$, where $x = |Q| \cdot (1 + \|\mathcal{A}\| + \|v_1\|)$.*

Proof. The bounds given in [13] for VASs are easily extended to VASSs. Instead of giving a full proof, we only explain how to adapt the proof of [13] to deal with control states.

The notions of *paths*, of *i-bounded* sequences and of *i-covering* sequences from [13, pp. 224–225] are extended with control states in the obvious way. For each $q \in Q$ and $v \in \mathbb{Z}^d$, define $m(i, q, v)$ to be the length of the shortest i-bounded, i-covering path in $\mathcal{V}$ starting from $q(v)$, with the convention that $m(i, q, v) = 0$ if there is none.

Now define $f(i) = \max\{m(i, q, v) \mid q \in Q, v \in \mathbb{Z}^d\}$. With the same reasoning as in [13, Lemma 3.4], we get that

$$f(0) \leq |Q| \text{ and } f(i+1) \leq |Q| \cdot (\max\{\|\mathcal{A}\|, \|v_1\|\} \cdot f(i))^{i+1} + f(i).$$

It follows that $f(i+1) \leq (x f(i))^{i+1}$. An immediate induction on i yields that $f(i) \leq x^{(i+1)!}$. In particular, we get that $m(d, q_0, v_0) \leq f(d) \leq x^{(d+1)!}$. Now, if $q_0(v_0) \xrightarrow{*} q_1(v_1)$ then $0 < m(d, q_0, v_0)$. This entails that $q_0(v_0) \xrightarrow{\alpha} q_1(\bar{v}_1)$ for some $\bar{v}_1 \geq v_1$ and $\alpha \in \mathcal{A}^*$ such that $|\alpha| = m(d, q_0, v_0) - 1 < x^{(d+1)!}$. □

Lemma 15. *Algorithm* Alg-Co-Empt*works (i.e., can be implemented to work) in exponential space.*

Proof. The point 1 of Alg-Co-Empt, including the binary presentation of the computed number n, can be performed in exponential space, w.r.t. the size of the binary presentation of the input $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and $q_0(v_0)$; this follows by Corollary 10.

The VASS $\mathcal{V}' = (\mathcal{V}^{\leftarrow})_{(n)}$ in the point 2 is not needed to be constructed explicitly. The algorithm creates the configuration $(q_0, u_0)(v_0')$ and then stepwise generates the control states (q, u) ($q \in Q$, $u \in \{0, 1, \ldots, n-1\}^d$) of $\mathcal{V}'$ and checks if $(q, u)(\mathbf{0})$ covers $(q_0, u_0)(v_0')$ in $\mathcal{V}'$.

It thus suffices to show that checking if $(q, u)(\mathbf{0})$ covers $(q_0, u_0)(v_0')$ (i.e., if $(q, u)(\mathbf{0}) \xrightarrow{*}_{\mathcal{V}'} (q_0, u_0)(\bar{v})$ for some $\bar{v} \geq v_0'$) can be done in exponential space (w.r.t. the binary presentation of $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and $q_0(v_0)$). By Proposition 14, it is enough to search for witnesses of coverability $(q, u)(\mathbf{0}) \xrightarrow{\alpha} (q_0, u_0)(\bar{v})$ of length $|\alpha| < x^{(d+1)!}$, where $x = |Q| n^d \cdot (1 + \|\mathcal{A}'\| + \|v_0'\|)$. Since n is at most double-exponential, $x^{(d+1)!}$ is also at most double-exponential. As in the proof of Corollary 10, the algorithm just traverses along (a guessed bounded) α, keeping only the current configuration in memory; so exponential space is sufficient. □

Algorithm Deciding SIC Co-finiteness. We will adjust the algorithm Alg-Co-Emptso that, given $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and $q_0(v_0)$, it answers YES iff the set $(Q \times \mathbb{N}^d) \setminus \big[q_0(v_0)\big\rangle$ is finite; this can happen even if some $(q, u)(\mathbf{0})$ does not cover $(q_0, u_0)(v_0')$ in $\mathcal{V}'$. Informally speaking, it suffices to check if $(q, u)(\mathbf{0})$ covers

$(q_0, u_0)(v_0')$ whenever we "ignore" one-component of $\mathbf{0}$, making it "arbitrarily large".

By ω we denote an "infinite amount", satisfying $z < \omega$ and $z + \omega = \omega + z = \omega$ for all $z \in \mathbb{Z}$. Given $\mathcal{V} = (d, Q, \mathcal{A}, T)$, by the set of *extended configurations* we mean the set $Q \times (\mathbb{N} \cup \{\omega\})^d$; the relations $q(v) \xrightarrow{\mathbf{a}} q'(v')$, $q(v) \xrightarrow{\alpha} q'(v')$ ($\alpha \in \mathcal{A}^*$), and $q(v) \xrightarrow{*} q'(v')$ are then naturally extended to the relations on $Q \times (\mathbb{N} \cup \{\omega\})^d$. (Hence, e.g., if $q(v) \xrightarrow{\alpha} q'(v')$ then $v(i) = \omega$ iff $v'(i) = \omega$, for any $i \in [1, d]$.)

Let us now consider the following algorithm.

Algorithm Alg-Co-Finit

Input: a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and a configuration $q_0(v_0)$.

Output: YES if $(Q \times \mathbb{N}^d) \setminus \big[q_0(v_0)\big\rangle$ is finite, and NO otherwise.

1. As in Alg-Co-Empt.
2. As in Alg-Co-Empt.
3. For each control state (q, u) of $\mathcal{V}'$ and each $i \in [1, d]$ check if $(q, u)(\omega \mathbf{e}_i)$ covers $(q_0, u_0)(v_0')$ (in $\mathcal{V}'$), i.e., if
$$(q, u)(\omega \mathbf{e}_i) \xrightarrow{*}_{\mathcal{V}'} (q_0, u_0)(\bar{v}) \text{ for some } \bar{v} \geq v_0';$$
by $\omega \mathbf{e}_i$ we denote the d-dimensional vector where the i-th component is ω and the other components are zero.
If the answer is negative for some (q, u) and $i \in [1, d]$, then **return** NO, otherwise (when all $(q, u)(\omega \mathbf{e}_i)$ cover $(q_0, u_0)(v_0')$) **return** YES.

Correctness and Exponential-Space Complexity of Alg-Co-Finit

Lemma 16. *Algorithm* Alg-Co-Finit *satisfies its specification (i.e., returns YES if $(Q \times \mathbb{N}^d) \setminus \big[q_0(v_0)\big\rangle$ is finite, and NO otherwise).*

Proof. We reason analogously as in the proof of Lemma 13. We have already noted that if NO is returned in the point 1, then $(Q \times \mathbb{N}^d) \setminus \big[q_0(v_0)\big\rangle$ is infinite.

Assume first that $\big[q_0(v_0)\big\rangle_{\mathcal{V}}$ is co-finite and let us show that the algorithm returns YES. Let (q, u) be a control state of $\mathcal{V}'$ and let $i \in [1, d]$. Since $\big[q_0(v_0)\big\rangle_{\mathcal{V}}$ is co-finite, there is a number $x \geq 1$ such that $q_0(v_0) \xrightarrow{*}_{\mathcal{V}} q(u + nx\mathbf{e}_i)$. It follows that $(q, u)(x\mathbf{e}_i) \xrightarrow{*}_{\mathcal{V}'} (q_0, u_0)(v_0')$, which also entails that $(q, u)(\omega \mathbf{e}_i)$ covers $(q_0, u_0)(v_0')$ in $\mathcal{V}'$. We have proved that the algorithm returns YES.

Assume now that the algorithm returns YES and let us prove that $\big[q_0(v_0)\big\rangle_{\mathcal{V}}$ is co-finite. Since $(q, u)(\omega \mathbf{e}_i)$ covers $(q_0, u_0)(v_0')$ in $\mathcal{V}'$ for every control state (q, u) of $\mathcal{V}'$ and for every $i \in [1, d]$, there is a (large enough) number x such that $(q, u)(x\mathbf{e}_i)$ covers $(q_0, u_0)(v_0')$ for every control state (q, u) and every $i \in [1, d]$. Below we prove that every configuration $q(v)$ of $\mathcal{V}$ such that $\|v\| \geq nx$ is reachable from $q_0(v_0)$; this will entail that $\big[q_0(v_0)\big\rangle_{\mathcal{V}}$ is co-finite (i.e., $(Q \times \mathbb{N}^d) \setminus \big[q_0(v_0)\big\rangle_{\mathcal{V}}$ is finite).

We thus fix an arbitrary $q(v)$ and $i \in [1, d]$ such that $v(i) \geq nx$. Let $(q, u)(v')$ be the configuration of $\mathcal{V}'$ corresponding to $q(v)$; hence $v = u + nv'$. Since $(q, u)(x\mathbf{e}_i)$ covers $(q_0, u_0)(v_0')$ in $\mathcal{V}'$, there is $\bar{v}_0' \geq v_0'$ such that

$(q,u)(x\mathbf{e}_i) \xrightarrow{*}_{\mathcal{V}'} (q_0,u_0)(\bar{v}_0')$; this entails $q_0(u_0+n\bar{v}_0') \xrightarrow{*}_{\mathcal{V}} q(u+nx\mathbf{e}_i)$.

Since $v(i) \geq nx$, we have $v'-x\mathbf{e}_i \geq \mathbf{0}$. By monotonicity we derive

$$q_0(u_0+n\bar{v}_0'+n(v'-x\mathbf{e}_i)) \xrightarrow{*}_{\mathcal{V}} q(u+nx\mathbf{e}_i+n(v'-x\mathbf{e}_i)) = q(v).$$

By the definition of n, we get

$$q_0(v_0) \xrightarrow{*}_{\mathcal{V}} q_0(v_0+n(\bar{v}_0'-v_0')+n(v'-x\mathbf{e}_i)) = q_0(u_0+n\bar{v}_0'+n(v'-x\mathbf{e}_i)).$$

Hence we indeed have $q_0(v_0) \xrightarrow{*}_{\mathcal{V}} q(v)$. □

Lemma 17. *Algorithm* Alg-Co-Finit*works (i.e., can be implemented to work) in exponential space.*

Proof. This is analogous to the proof of Lemma 15. We just note that deciding if $(q,u)(\omega\mathbf{e}_i)$ covers $(q_0,u_0)(v_0')$ is even easier than deciding if $(q,u)(\mathbf{0})$ covers $(q_0,u_0)(v_0')$, since the i-th component can be simply ignored. □

5 Applications of the Co-emptiness Problem

A motivation for the study in this paper has been the decidability proof for structural liveness in [7], which is based on a particular version of the co-emptiness problem. We now give more details (in the framework of VASSs, which is equivalent to the framework of Petri nets used in [7]), and some partial complexity results. The main aim is to attract a further research effort on this topic, since the complexity of various related problems has not been answered. In particular, we have no nontrivial complexity bounds for the structural liveness problem (besides its decidability).

Assuming a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$, we are now particularly interested in the co-emptiness of $\big[D\big\rangle_{\mathcal{V}}$ for downward closed sets $D \subseteq Q \times \mathbb{N}^d$, which constitute a subclass of semilinear sets. We use the notation

$$\downarrow C = \{q(v) \mid v \leq v' \text{ for some } q(v') \in C\}$$

for the *downward closure* of a set $C \subseteq Q \times \mathbb{N}^d$ (of configurations of $\mathcal{V}$). We say that $C \subseteq Q \times \mathbb{N}^d$ is *downward closed* if $\downarrow C = C$.

We write just $\downarrow q(v)$ instead of $\downarrow \{q(v)\}$.

Downward closed sets are semilinear since each such set can be presented as

$$\downarrow q_1(\bar{v}_1) \cup \downarrow q_2(\bar{v}_2) \cup \cdots \cup \downarrow q_m(\bar{v}_m)$$

for some $m \in \mathbb{N}$ and $\bar{v}_i \in (\mathbb{N} \cup \{\omega\})^d$ $(i \in [1,m])$, where we put

$$\downarrow q(\bar{v}) = \{q(v) \mid v \leq \bar{v}, v \in \mathbb{N}^d\}.$$

(Recall that $k < \omega$ for each $k \in \mathbb{N}$.)

Later we use another natural presentation of downward closed sets: for each $q \in Q$ we provide a constraint in the form of a (finite) conjunction of disjunctions

of atomic constraints of the form $v(i) \leq c$ where $i \in [1,d]$ and $c \in \mathbb{N}$ (then $q(v) \in Q \times \mathbb{N}^d$ is in the set iff v satisfies the constraint associated with q).

The *DCIS co-emptiness problem* where "DCIS" stands for "Downward Closed Initial Sets of configurations" (i.e., given $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and a downward closed set $D \subseteq Q \times \mathbb{N}^d$, is $\big[D\big\rangle_{\mathcal{V}} = Q \times \mathbb{N}^d$?) is decidable by Theorem 3 (and the fact that D is semilinear). The complexity is open, even the reductions to/from the reachability problem are unclear. Now we explain the previously mentioned motivation for such studies.

Liveness of Transitions and Configurations. We recall some standard definitions and facts. Given a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$,

- a transition $t \in T$ is *enabled* in a configuration $q(v)$ if t is of the form $t : q \xrightarrow{\mathbf{a}} q'$ and $v + \mathbf{a} \geq \mathbf{0}$;
- a transition t is *live* in $q(v)$ if for every $\bar{q}(\bar{v}) \in \big[q(v)\big\rangle$ there is $q'(v') \in \big[\bar{q}(\bar{v})\big\rangle$ such that t is enabled in $q'(v')$;
- a transition t is *dead* in $q(v)$ if there is no $q'(v') \in \big[q(v)\big\rangle$ such that t is enabled in $q'(v')$.

We note that t is not live in $q(v)$ iff t is dead in some $q'(v') \in \big[q(v)\big\rangle$.

The next proposition (which also defines $\mathcal{D}_{t,\mathcal{V}}$ and $\mathcal{D}_{\mathcal{V}}$) is obvious, due to monotonicity.

Proposition 18. *Given a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$, for each $t \in T$ the set*

$$\mathcal{D}_{t,\mathcal{V}} = \{q(v) \mid t \text{ is dead in } q(v)\}$$

is downward closed. Hence also the set

$$\mathcal{D}_{\mathcal{V}} = \{q(v) \mid \text{some } t \in T \text{ is dead in } q(v)\} = \textstyle\bigcup_{t \in T} \mathcal{D}_{t,\mathcal{V}}$$

is downward closed.

Given a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$, a *configuration* $q(v)$ is *live* if each $t \in T$ is live in $q(v)$, i.e., if $q(v) \not\xrightarrow{*}_{\mathcal{V}} \mathcal{D}_{\mathcal{V}}$. A VASS $\mathcal{V}$ is *structurally live* if it has a live configuration, hence if the set

$$\mathcal{L}_{\mathcal{V}} = \{q(v) \mid q(v) \text{ is a live configuration of } \mathcal{V}\}$$

is nonempty. While the membership problem for ($\mathcal{D}_{t,\mathcal{V}}$ or) $\mathcal{D}_{\mathcal{V}}$ is essentially a version of the (non)coverability problem, which also allows to construct a natural presentation of the (downward closed) sets $\mathcal{D}_{t,\mathcal{V}}$ and $\mathcal{D}_{\mathcal{V}}$, the membership problem for $\mathcal{L}_{\mathcal{V}}$ is close to the *reachability problem* as was already noted by Hack [5] long time ago.

The set $\mathcal{L}_{\mathcal{V}}$ is indeed more involved than $\mathcal{D}_{\mathcal{V}}$; it is obviously not downward closed but it is not upward closed either (in general), and it can be even non-semilinear; we can refer to [7] for a concrete example, as well as for the following idea of decidability.

The structural liveness can be decided as follows. We recall the reversed VASS $\mathcal{V}^{\leftarrow}$, and note that $\mathcal{V}$ is not structurally live iff $\big[\mathcal{D}_{\mathcal{V}}\big\rangle_{\mathcal{V}^{\leftarrow}}$ is co-empty:

Proposition 19. *For any VASS* $\mathcal{V} = (d, Q, \mathcal{A}, T)$ *we have*

$$\big[\mathcal{D}_{\mathcal{V}}\big\rangle_{\mathcal{V}^{\leftarrow}} = (Q \times \mathbb{N}^d) \setminus \mathcal{L}_{\mathcal{V}}.$$

Hence $\mathcal{V}$ *is not structurally live iff* $\big[\mathcal{D}_{\mathcal{V}}\big\rangle_{\mathcal{V}^{\leftarrow}} = Q \times \mathbb{N}^d$.

Proof. We recall that $q(v)$ is not live iff $\big[q(v)\big\rangle_{\mathcal{V}} \cap \mathcal{D}_{\mathcal{V}} \neq \emptyset$ (i.e., iff $q(v) \xrightarrow{*}_{\mathcal{V}} q'(v')$ where some $t \in T$ is dead in $q'(v')$). Hence $q(v)$ is not live iff $q'(v') \xrightarrow{*}_{\mathcal{V}^{\leftarrow}} q(v)$ for some $q'(v') \in \mathcal{D}_{\mathcal{V}}$ (using Proposition 11).

Therefore $\big[\mathcal{D}_{\mathcal{V}}\big\rangle_{\mathcal{V}^{\leftarrow}} = (Q \times \mathbb{N}^d) \setminus \mathcal{L}_{\mathcal{V}}$. □

Proposition 19 allows us to decide structural liveness of a given VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ by a reduction to the co-emptiness problem, using the above-mentioned constructability of $\mathcal{D}_{\mathcal{V}}$.

Structural Deadlock-Freedom and DCIS Co-emptiness. We have shown that the complementary problem of the structural liveness problem (hence "non structural liveness") can be reduced to the DCIS co-emptiness problem (with downward closed sets of initial configurations). However, we have no reduction from the latter problem to the former.

We now show that a special form of structural liveness, namely structural deadlock-freedom, is closely related to the DCIS co-emptiness problem. We use the previously mentioned presentation of downward closed sets by conjunctions of disjunctions of atomic constraints of the form $v(i) \leq c$ (for each $q \in Q$).

Given a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$, a configuration $q(v)$ is *deadlock-free* if every configuration in $\big[q(v)\big\rangle$ enables some transition. A VASS $\mathcal{V}$ is *structurally deadlock-free* if it has a deadlock-free configuration. The *structural deadlock-freedom problem* asks, given a VASS $\mathcal{V}$, if $\mathcal{V}$ is structurally deadlock-free.

In the rest of this section we prove the following theorem.

Theorem 20. *The complementary problem of the structural deadlock-freedom problem is polynomially interreducible with the DCIS co-emptiness problem. This entails that the structural deadlock-freedom problem is decidable.*

We have already noted that the DCIS co-emptiness problem is decidable. The interreducibility claimed in Theorem 20 is proven in the rest of this section. We first define the set

$$\mathcal{S}_{\mathcal{V}} = \{q(v) \mid \text{no } t \in T \text{ is enabled in } q(v)\}$$

of "sink configurations" or "deadlocks" (hence $\mathcal{S}_{\mathcal{V}} = \bigcap_{t \in T} \mathcal{D}_{t,\mathcal{V}}$). It is obvious that $\mathcal{S}_{\mathcal{V}}$ is the downward closed set described so that to each $q \in Q$ we attach the constraint

$$\bigwedge_{(q \xrightarrow{\mathbf{a}} q') \in T} \quad \bigvee_{\substack{i \in [1,d] \\ \mathbf{a}(i) < 0}} v(i) \leq -\mathbf{a}(i) - 1.$$

This presentation of $\mathcal{S}_\mathcal{V}$ can be clearly constructed in polynomial time, when given a VASS $\mathcal{V}$. Hence Proposition 21 entails the "left-to-right" reduction in Theorem 20 (recall that $\mathcal{V}^{\leftarrow}$ denotes the reversed VASS of $\mathcal{V}$). The other reduction is shown by Proposition 22.

Proposition 21. *A VASS $\mathcal{V}$ is not structurally deadlock-free iff $\big[\mathcal{S}_\mathcal{V}\big\rangle_{\mathcal{V}^{\leftarrow}}$ is co-empty.*

Proof. We consider a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$, and observe that $q(v)$ is not deadlock-free iff $\big[q(v)\big\rangle_\mathcal{V} \cap \mathcal{S}_\mathcal{V} \neq \emptyset$. Hence $q(v)$ is not deadlock-free iff $q(v) \in \big[\mathcal{S}_\mathcal{V}\big\rangle_{\mathcal{V}^{\leftarrow}}$ (using Proposition 11). It follows that $\mathcal{V}$ is not structurally deadlock-free iff $\big[\mathcal{S}_\mathcal{V}\big\rangle_{\mathcal{V}^{\leftarrow}} = Q \times \mathbb{N}^d$. □

Proposition 22. *Given a VASS $\mathcal{V}$ and a downward-closed set D of configurations, we can construct, in polynomial time, a VASS $\mathcal{V}'$ such that $\big[D\big\rangle_\mathcal{V}$ is co-empty iff $\mathcal{V}'$ is not structurally deadlock-free.*

Proof. Let us assume a VASS $\mathcal{V} = (d, Q, \mathcal{A}, T)$ and a downward-closed set D of configurations given, for each $q \in Q$, by conjunctions of disjunctions of atomic constraints of the form $v(i) \leq c$. By negating these formulas, we derive, in polynomial time, a collection $(B_q)_{q \in Q}$ of finite subsets of $\mathbb{N}^d$ such that

$$(Q \times \mathbb{N}^d) \setminus D = \{q(v) \mid v \geq b \text{ for some } b \in B_q\}.$$

(Hence $(B_q)_{q \in Q}$ represents the upward closed complement of D.)

We now define the VASS $\hat{\mathcal{V}} = (d, \hat{Q}, \hat{\mathcal{A}}, \hat{T})$ as follows:

(a) $\hat{Q} = Q \cup \{(q, b) \mid q \in Q, b \in B_q\}$.
(b) $\hat{T}$ consists of the following transitions:
  i. $q \xrightarrow{-b} (q, b)$ and $(q, b) \xrightarrow{b} q$ for all $q \in Q$, $b \in B_q$, and
  ii. $(q, b) \xrightarrow{\mathbf{a}+b} q'$ for all $(q \xrightarrow{\mathbf{a}} q') \in T$ and $b \in B_q$.
(c) $\hat{\mathcal{A}} = \{\hat{\mathbf{a}} \mid q \xrightarrow{\hat{\mathbf{a}}} q' \in \hat{T} \text{ for some } q, q' \in \hat{Q}\}$.

It is obvious that for all configurations $q(v)$ and $q'(v')$ of $\mathcal{V}$ we have that

$$q(v) \xrightarrow{*}_{\hat{\mathcal{V}}} q'(v') \text{ implies } q(v) \xrightarrow{*}_{\mathcal{V}} q'(v')$$

but the converse does not hold in general. We will show that

$$\big[D\big\rangle_{\mathcal{V}^{\leftarrow}} = Q \times \mathbb{N}^d \text{ iff } \hat{\mathcal{V}} \text{ is not structurally deadlock-free.}$$

The proof will be finished, by taking $\mathcal{V}' = \widehat{\mathcal{V}^{\leftarrow}}$ (and noting that $(\mathcal{V}^{\leftarrow})^{\leftarrow} = \mathcal{V}$).

($\Rightarrow$) Assume $\big[D\big\rangle_{\mathcal{V}^{\leftarrow}} = Q \times \mathbb{N}^d$. Observe that $(q, b)(v) \xrightarrow{b} q(v+b)$ in $\hat{\mathcal{V}}$ for every $(q, b) \in \hat{Q}$ and $v \in \mathbb{N}^d$. We now show that no configuration $q(v)$ with $q \in Q$ is deadlock-free in $\hat{\mathcal{V}}$, which clearly entails that $\hat{\mathcal{V}}$ is not structurally deadlock-free.

Let us fix some $q(v) \in Q \times \mathbb{N}^d$. Since $q(v) \in \big[D\big\rangle_{\mathcal{V}^{\leftarrow}}$, there are $\mathbf{a}_1, \ldots, \mathbf{a}_m \in \mathcal{A}$ and $q_0(v_0), \ldots, q_m(v_m) \in (Q \times \mathbb{N}^d)$ such that

$$q(v) = q_0(v_0) \xrightarrow{\mathbf{a}_1} q_1(v_1) \xrightarrow{\mathbf{a}_2} \cdots q_{m-1}(v_{m-1}) \xrightarrow{\mathbf{a}_m} q_m(v_m) \in D \text{ in } \mathcal{V}$$

(recall Proposition 11). Moreover, we may assume w.l.o.g. that $q_i(v_i) \notin D$ for all $i \in [0, m-1]$. So for each $i \in [0, m-1]$ there is $b_i \in B_{q_i}$ such that $v_i \geq b_i$. We derive that

$$q_i(v_i) \xrightarrow{-b_i} (q_i, b_i)(v_i - b_i) \xrightarrow{\mathbf{a}_{i+1}+b_i} q_{i+1}(v_{i+1}) \text{ in } \hat{\mathcal{V}}$$

for all $i \in [0, m-1]$. It follows that $q(v) \xrightarrow{*} q_m(v_m)$ in $\hat{\mathcal{V}}$. Since $q_m(v_m) \in D$ then $v_m \not\geq b$ for all $b \in B_{q_m}$; hence no transition of $\hat{\mathcal{V}}$ is enabled in $q_m(v_m)$, and $q(v)$ is thus not deadlock-free in $\hat{\mathcal{V}}$.

($\Leftarrow$) Assume that $\hat{\mathcal{V}}$ is not structurally deadlock-free. We fix a configuration $q(v)$ of $\mathcal{V}$ and prove that $q(v) \in \big[D\big\rangle_{\mathcal{V}^{\leftarrow}}$. Since $q(v)$ is also a configuration of $\hat{\mathcal{V}}$, it is not deadlock-free in $\hat{\mathcal{V}}$. So there is a configuration $q'(v')$ of $\hat{\mathcal{V}}$ such that $q(v) \xrightarrow{*}_{\hat{\mathcal{V}}} q'(v')$ and no transition $t \in \hat{T}$ is enabled in $q'(v')$. Since $\hat{\mathcal{V}}$ contains the transition $(q, b) \xrightarrow{b} q$ for every $q \in Q$ and $b \in B_q$, we get that $q' \in Q$. No transition $q' \xrightarrow{-b} (q', b)$ of $\hat{T}$ is enabled in $q'(v')$, so $v' \not\geq b$ for every $b \in B_{q'}$. It follows that $q'(v') \in D$. Since $q(v) \xrightarrow{*}_{\hat{\mathcal{V}}} q'(v')$ implies $q(v) \xrightarrow{*}_{\mathcal{V}} q'(v')$, we get $q(v) \xrightarrow{*}_{\mathcal{V}} D$, i.e., $q(v) \in \big[D\big\rangle_{\mathcal{V}^{\leftarrow}}$. □

6 Conclusion

Motivated by the structural liveness problem for VASS, whose computational complexity is still open, we have introduced and studied in this paper the co-emptiness problem and the co-finiteness problem for VASS. The complexity of the co-emptiness and co-finiteness problems in the case of single initial configurations has been clarified, but the complexity of general versions has been left open, even w.r.t. reductions to/from the reachability problem. This requires further work, in particular with an eye to the applications aiming to clarify structural liveness properties of VASSs, or equivalently of Petri nets.

References

1. Atig, M.F., Habermehl, P.: On Yen's path logic for Petri nets. Int. J. Found. Comput. Sci. **22**(4), 783–799 (2011). https://doi.org/10.1142/S0129054111008428
2. Best, E., Esparza, J.: Existence of home states in Petri nets is decidable. Inf. Process. Lett. **116**(6), 423–427 (2016)
3. Esparza, J.: Decidability and complexity of Petri net problems—an introduction. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1998. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_20
4. Ginsburg, S., Spanier, E.H.: Semigroups, Presburger formulas, and languages. Pac. J. Math. **16**(2), 285–296 (1966)
5. Hack, M.: The recursive equivalence of the reachability problem and the liveness problem for Petri nets and vector addition systems. In: 15th Annual Symposium on Switching and Automata Theory, New Orleans, Louisiana, USA, 14–16 October 1974, pp. 156–164. IEEE Computer Society (1974). https://doi.org/10.1109/SWAT.1974.28

6. Hopcroft, J.E., Pansiot, J.: On the reachability problem for 5-dimensional vector addition systems. Theor. Comput. Sci. **8**, 135–159 (1979)
7. Jančar, P.: Deciding structural liveness of Petri nets. In: Steffen, B., Baier, C., van den Brand, M., Eder, J., Hinchey, M., Margaria, T. (eds.) SOFSEM 2017. LNCS, vol. 10139, pp. 91–102. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51963-0_8
8. Leroux, J.: Presburger vector addition systems. In: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, 25–28 June 2013, pp. 23–32. IEEE Computer Society (2013). https://doi.org/10.1109/LICS.2013.7
9. Leroux, J.: Vector addition system reversible reachability problem. Log. Methods Comput. Sci. **9**(1), 1–16 (2013). https://doi.org/10.2168/LMCS-9(1:5)2013
10. Leroux, J., Schmitz, S.: Demystifying reachability in vector addition systems. In: 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, 6–10 July 2015, pp. 56–67. IEEE Computer Society (2015). https://doi.org/10.1109/LICS.2015.16
11. Lipton, R.J.: The reachability problem requires exponential space. Technical report 63, Department of Computer Science, Yale University, January 1976
12. Mayr, E.W.: An algorithm for the general Petri net reachability problem. SIAM J. Comput. **13**(3), 441–460 (1984). https://doi.org/10.1137/0213029
13. Rackoff, C.: The covering and boundedness problems for vector addition systems. Theor. Comput. Sci. **6**, 223–231 (1978). https://doi.org/10.1016/0304-3975(78)90036-1
14. Yen, H.: A unified approach for deciding the existence of certain Petri net paths. Inf. Comput. **96**(1), 119–137 (1992). https://doi.org/10.1016/0890-5401(92)90059-O

Languages

An Efficient Characterization of Petri Net Solvable Binary Words

David de Frutos Escrig[1], Maciej Koutny[2], and Łukasz Mikulski[3(✉)]

[1] Dpto. Sistemas Informáticos y Computación, Facultad de Ciencias Matemáticas, Universidad Complutense de Madrid, Madrid, Spain
defrutos@sip.ucm.es

[2] School of Computing, Newcastle University, Newcastle upon Tyne NE4 5TG, UK
maciej.koutny@ncl.ac.uk

[3] Faculty of Mathematics and Computer Science, Nicolaus Copernicus University in Toruń, Chopina 12/18, Toruń, Poland
lukasz.mikulski@mat.umk.pl

Abstract. We present a simple characterization of the set of Petri net solvable binary words. It states that they are exactly the extensions of the prefixes of Petri net cyclic solvable words, by some prefix x^k, where x is any letter of the binary alphabet being considered, and k is any natural number. We derive several consequences of this characterization which, in a way, shows that the set of solvable words is 'smaller than expected'. Therefore, the existing conjecture that all of them can be generated by quite simple net is not only confirmed, but indeed reinforced. As a byproduct of the characterization, we also present a linear time algorithm for deciding whether a binary word is solvable. The key idea is that the connection with the cyclic solvable words induces certain structural regularity. Therefore, one just needs to look for possible irregularities, which can be done in a structural way, resulting in a rather surprising linearity of the decision algorithm. Finally, we employ the obtained results to provide a characterization of reversible binary transition systems.

Keywords: Petri net · Binary word · Word solvability · Reversibility Binary transition system

1 Introduction

In the past few years several authors investigated finite labelled transition systems (flts) which can be *solved* by Petri nets (i.e., flts's which are isomorphic to the reachability graphs of Petri nets). The solvability problem turned out to be more complicated than expected and a very particular case of the general problem, where the flts's are *linear* and defined by *binary* words, was studied first. Based mainly on the theory of regions [1] many useful properties of the set of solvable words have been obtained. In particular, [5] presented two decision algorithms (with quadratic time complexity) for the solvability of finite and

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 207–226, 2018.
https://doi.org/10.1007/978-3-319-91268-4_11

cyclic words. Both algorithms are based on a *quantitative* analysis that relates the number of a's and b's in any 'convenient' *piece* of the tested word w, checking whether in all cases the obtained quantities are *close enough*. In this way, the sets of solvable and cyclic solvable words are totally characterized. [5] was a continuation of [2], that introduced quantitative techniques used also here.

Studying the two (collections of) conditions that characterize both solvable and cyclic solvable words, we observed that they are related, and in fact look quite *similar*. However, following a closer investigation we *discovered* some important differences that make it difficult to bring to light the close connection between these two sets of solvable words. As discussed in Sect. 4, the conditions verifying the solvability of (plain) words compare *any* two consecutive parts β and γ, which are connected producing a full sequence $y\beta x\gamma y$, where $x \neq y$ are letters of the alphabet $\{a, b\}$ being considered. However, in the cyclic case, the two compared parts cover the whole verified word, i.e., $w' = y\beta x\gamma$ for a circular permutation w' of the verified word w. The first property can be seen as *local*, since we need to compare two (consecutive) pieces that can be arbitrarily long/short, whereas the second is *global*.

This paper focuses on the *structural* results. Some are recalled from [2,5], and prove that solvable words can be decomposed into *blocks* ab^{x_i}. We reformulate the original conditions giving more precise description of block decomposition which is easier to verify. An informal statement of the resulting criteria is as follows: 'Any two adequate pieces, consecutive or not, of a solvable word must contain *quite similar* proportion of a's and b's'. Moreover, an *adequate* piece can be understood as a sequence $ab^{x_i}ab^{x_{i+1}} \dots ab^{x_{i+m-1}}$ of full blocks.

The only difference between the two sets of conditions is the way in which they treat the beginning of a word. In the plain case, a prefix x^k (followed by $y \neq x$) will not be considered in most cases. In the cyclic case, any part of the word is subjected to the same procedure due to the *circularity* of cyclic words.

As long as we test (nearly) any two pieces of a word in order to verify its solvability, the corresponding algorithm remains quite costly. In fact, a direct check of the restated criteria would be even worse in the general case, since we need to check also nonconsecutive pieces. Having said that, the algorithm could be used to *catch* unsolvable words in a fast way, by exhibiting two *unbalanced* (possibly nonconsecutive) sequences of blocks.

The structural properties stated in [6] imply that no cyclic solvable word w contains both aa and bb, which could be seen as the simplest version of the procedure confirming unsolvability, and is also a basic test when we check solvability. And, if w passes it, we apply the block decomposition results. More precisely, we take advantage of the result that there are no two full blocks ab^{x_i} and ab^{x_j} with $|x_i - x_j| \geq 2$, which is another example of our linear time local test.

If both simple tests are passed, we need to check more sophisticated conditions. In particular, if w is solvable and contains two consecutive ab^{e-1} blocks, then it cannot contain two consecutive ab^e blocks. This corresponds to the *structural periodicity* of solvable words: if there are no two unbalanced pieces at the level of single letters, then we proceed at the level of blocks. We then continue

the analysis in a similar hierarchical way, checking larger and larger parts of the word, until either we find a witness of unsolvability, or we terminate arriving at a trivial solvable word which proves the solvability of the original word. Fortunately, each step of this recursive procedure at least halves the length of the checked word. As a result, even if the number of iterations is logarithmic, the full cost of the algorithm is linear, as each of the iterations is linear in the current size of the checked word.

We adopted from [6] the idea to generate recursive reductions, called here *derivatives*. The authors of [6] introduced a *compression* operator that applies our general derivation operator in a particular case of the generation of a minimal unsolvable word from another, longer word. In fact, along with the compression mechanism, [6] presented an *extension* procedure that played a dual role. In a similar vein, we will introduce a general *integration* operator, and show that derivation/integration preserves and reflects solvability (and so also unsolvability!). Its application will lead, as annouces above, to a pair of optimal (linear time) decision algorithms, which verify both plain and cyclic solvability.

Our initial motivation was to extend the existing results on solvability, continuing the study of the reversibility of linear transition systems initiated in the conference version of [4], where questions about decidability of several related notions are studied. In this paper, we show that a linear transition system can be reversed if and only if both the word w generated by the system, and its reversed version w^{rev} are solvable. Then, we look for a more explicit definition of the set of reversible words. In [3] it was already demonstrated that not every solvable word is reversible. However, we show that all the cyclic solvable words are reversible. Hence, towards the end of in this paper, we will look for the connections and differences between these two classes. We show that there are *a few* 'simple' reversible words that are not cyclic solvable, but all the other 'more complex' reversible words are.

The paper is organized as follows. After recalling the basic notions in Sect. 2, Sect. 3 contains new characterizations of cyclic solvable words, including the efficient algorithm for detect them. Sections 4 and 5 deal with plain solvable words and reversible words, respectively. A brief section containing conclusions ends the paper.

2 Basic Notions

The sets of all integers and non-negative integers are denoted by $\mathbb{Z}$ and $\mathbb{N}$, respectively.

Words. A *word over an alphabet* T is a finite sequence $w \in T^*$, and it is *binary* if $|T| = 2$. The empty word is denoted by ε. The *reverse* of a word $w = t_1 \dots t_n$ is $w^{rev} = t_n \dots t_1$. For a word $w \in T^*$ and a letter $t \in T$, $|w|$ denotes the length of w, and $|w|_t$ denotes the number of occurrences of t in w. Moreover, $|w|_y^x = |w|_x/|w|_y$ and if $|w|_y = 0$, then $|w|_y^x = \infty$. A word $w' \in T^*$ is called a *subword* of $w \in T^*$ if $w = uw'v$, for some $u, v \in T^*$. In particular, w' is a *prefix*

of w if $u = \varepsilon$, a *suffix* of w if $v = \varepsilon$, and an *infix* of w if $u \neq \varepsilon \neq v$. The concatenation of k copies of a word $w \in T^*$ is denoted by w^k. Moreover, w^ω denotes an infinite word obtained by concatenating infinitely many copies of w. Note that in the denotation w^ω we *explicitly indicate the period* w which is used to construct the infinite word $ww\ldots$. Similarly, uw^ω denotes an infinite word $uww\ldots$ constructed using the prefix u and period w.

For two alphabets T and V, a mapping $\phi : T^* \to V^*$ is a *morphism* if $\phi(\varepsilon) = \varepsilon$ and $\phi(uv) = \phi(u)\phi(v)$, for all $u, v \in T^*$. A morphism ϕ is uniquely determined by its application to the members of T.

Transition Systems. A *finite labelled transition system* (or *flts*) is a tuple $TS = (S, T, \to, s_0)$ with a finite set of *states* S, a finite set of *labels* T, a set of *arcs* $\to \subseteq (S \times T \times S)$, and an *initial state* $s_0 \in S$. A label t is *enabled* at $s \in S$, denoted by $s[t\rangle$, if $(s, t, s') \in \to$, for some $s' \in S$. A state s' is *reachable* from s through the execution of $\sigma \in T^*$, denoted by $s[\sigma\rangle s'$, if there is a directed path from s to s' whose arcs are labelled consecutively by σ. The set of states reachable from s is denoted by $[s\rangle$. A sequence $\sigma \in T^*$ is *enabled* at a state s, denoted by $s[\sigma\rangle$, if there is some state s' such that $s[\sigma\rangle s'$.

$t^\bullet_{TS} = \{s \in S \mid \exists s' \in S : (s', t, s) \in \to\}$ and ${}^\bullet t_{TS} = \{s \in S \mid \exists s' \in S : (s, t, s') \in \to\}$ are respectively the sets of all states having an incoming arc labelled with t, and an outgoing arc labelled with t. The set of all arcs labelled by t is denoted by $\overrightarrow{t}$. We assume that each $\overrightarrow{t}$ is nonempty, and each state is reachable from s_0.

Two flts's, $(S, T, \to, s_0)$ and $(S', T, \to', s'_0)$, are *isomorphic* if there is a bijection $\zeta\colon S \to S'$ with $\zeta(s_0) = s'_0$, and $(s, t, s') \in \to \Leftrightarrow (\zeta(s), t, \zeta(s')) \in \to'$, for all $s, s' \in S$ and $t \in T$.

Petri Nets. A *(place/transition) net* is a tuple $N = (P, T, F, M_0)$, where P is a finite set of *places*, T is a disjoint finite set of *transitions* (or actions), F is the *flow function* $F\colon ((P \times T) \cup (T \times P)) \to \mathbb{N}$ specifying the arc weights, and M_0 is the *initial marking* (where a marking is a mapping $M\colon P \to \mathbb{N}$).

A transition $t \in T$ is *enabled* at a marking M, denoted by $M[t\rangle$, if $M(p) \geq F(p, t)$, for every $p \in P$. The *firing* of an enabled t at marking M leads to M', denoted by $M[t\rangle M'$, where $M'(p) = M(p) + F(t, p) - F(p, t)$, for every $p \in P$. The notions of enabledness and firing, $M[\sigma\rangle$ and $M[\sigma\rangle M'$, are extended in the usual way to sequences $\sigma \in T^*$, and $[M\rangle$ denotes the set of all markings reachable from M. For an infinite sequence of transitions, $\sigma \in T^\omega$, we write $M[\sigma\rangle$ if $M[\sigma'\rangle$, for infinitely many finite prefixes σ' of σ. We assume that each transition is enabled in at least one reachable marking.

Transition enabledness is *monotonic*, i.e., if a transition t is enabled at a marking M and $M(p) \leq M'(p)$, for every $p \in P$, then t is also enabled at M'.

The net N can also be specified as (N', M_0), where $N' = (P, T, F)$.

Synthesis of Words. A Petri net $N = (P, T, F, M_0)$ net is *bounded* if the set of reachable markings $[M_0\rangle$ is finite, and its *reachability graph* is then defined as the flts $RG(N) = ([M_0\rangle, T, \{(M, t, M') \mid M, M' \in [M_0\rangle \wedge M[t\rangle M'\}, M_0)$. If an

flts TS is isomorphic to the reachability graph of N, then N *solves* TS, and TS is *synthesizable* (to N).

Let $w = t_1 \dots t_m$ and $u = t_{m+1} \dots t_{m+n}$ be nonempty words, $T = \{t_1, \dots, t_m\}$, and $T' = \{t_1, \dots, t_{m+n}\}$. Then $w/w^\omega/wu^\omega$ is *solvable* if respectively

$$\begin{aligned} TS_w &= (\{0, \dots, m\}, T, \{(i-1, t_i, i) \mid 1 \le i \le m\}, 0) \\ TS^c_w &= (\{0, \dots, m-1\}, T, \{(i-1, t_i, i) \mid 1 \le i < m\} \cup \{(m-1, t_m, 0)\}, 0) \\ TS^c_{w,u} &= (\{0, \dots, m+n-1\}, T', \\ &\qquad \{(i-1, t_i, i) \mid 1 \le i < m+n\} \cup \{(m+n-1, t_m, m)\}, 0) \end{aligned}$$

is a synthesizable flts. Moreover, a Petri net N *solves* $w/w^\omega/wu^\omega$ if respectively the flts $TS_w/TS^c_w/TS^c_{w,u}$ is synthesizable to N (see Fig. 1).

In the rest of this paper, T is a binary alphabet and all the finite words considered belong to T^*. Moreover, x, y, t_i typically stand for the letters in T, and unless stated otherwise, $T = \{a, b\}$.

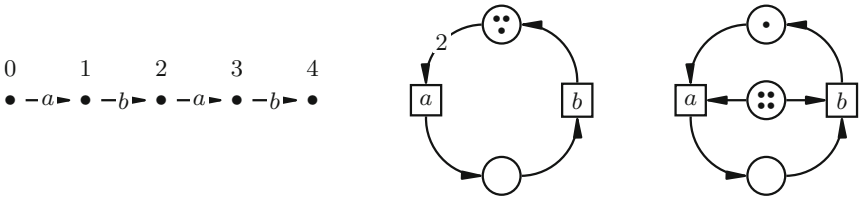


Fig. 1. The flts TS_{abab}, and two Petri nets solving *abab*. The second net without the middle place solves $(ab)^\omega$.

3 A Finer Characterisation of Cyclic Solvable Words

Two words, w and w', are *circular equivalent* if $w = uv$ and $w' = vu$, for some u and v. We denote this by $w \triangleq w'$. One can show that $\triangleq$ is an equivalence relation. The equivalence classes of $\triangleq$ are *circular words*, and the circular word containing $w = t_1 \dots t_n$ is $[w]_\triangleq = \{t_i \dots t_n t_1 \dots t_{i-1} \mid 1 \le i \le n\}$. Hence $|[w]_\triangleq| \le n$, and if $|[w]_\triangleq| = n$, then w is *prime*. One can show that, for each word w, there is a unique prime word u and $k \ge 1$ such that $w = u^k$. The *subwords* of $[w]_\triangleq$ are all the subwords of the members of $[w]_\triangleq$.

A word w is *cyclic solvable* if u^ω is solvable, where u is the prime satisfying $w = u^k$, for some $k \ge 1$. In such a case, every member of $[w]_\triangleq$ is cyclic solvable, and $[w]_\triangleq$ itself is *solvable*. A set of nets $\mathcal{N}$ solves $[w]_\triangleq$ if they cyclic solve all the members of $[w]_\triangleq$.

The above definitions are more subtle than it might appear at a first glance. First, notice that u^ω and $(u^k)^\omega$ 'construct' the same infinite word, but do so using different periods and the solvability is tested using different flts's, viz. TS^c_u and $TS^c_{u^k}$. Moreover, whenever u is not prime, no Petri net can solve u^ω,

and so generate the infinite cyclic word w denoted by it using u as period. In particular, if we take $(u^k)^\omega$ with $k > 1$, then $TS^c_{u^k}$ would include at least two different states generating w, and all such states should be represented by the same marking in the reachability graph of a Petri net to which $TS^c_{u^k}$ might be synthesiszable. This, however, is impossible.

Situations describe above are not a problem for the definition of cyclic solvability since we only consider prime periods u. For example, $w = aa$ is cyclic solvable because $w = a^2$, a is prime, and the flts $TS^c_a = (\{s\}, \{a\}, \{(s, a, s)\}, s)$ can be synthesized to the net $N_a = (\{p\}, \{a\}, F, M_0)$ with $M_0(p) = 1$ and the non-zero entries of F being $F(a, p) = F(p, a) = 1$. On the other hand, the flts $TS^c_{aa} = (\{s_0, s_1\}, \{a\}, \{(s_0, a, s_1), (s_1, a, s_0)\}, s_0)$ is not synthesizable.

Fact 1 (follows from Proposition 5 in [5]). *A solvable circular word does not have both aa and bb as subwords.*

Hence, we will usually assume (wlog) that aa is not a subword of cyclic solvable words. And, in order to avoid cumbersome trivial cases in proofs, we will disregard the solvable circular words $[a]_{\triangleq}$, $[b]_{\triangleq}$, and $[ab]_{\triangleq}$. As a result, we will concentrate on the remaining solvable circular words $[w]_{\triangleq}$, where $w = ab^{x_1}ab^{x_2}\ldots ab^{x_n}$ and $x_1, \ldots, x_n \geq 1$. Each such $[w]_{\triangleq}$ is a *circular block-word*, each w is a *block-word*, and each ab^{x_i} is a *block*. Also, a *block-subword* of $[w]_{\triangleq}$ is ε or $ab^{x_i}ab^{x_i+1}\ldots ab^{x_j}$ (for $i \leq j$) or $ab^{x_i}\ldots ab^{x_n}ab^{x_1}\ldots ab^{x_j}$ (for $i > j$). Note that not all such $[w]_{\triangleq}$ are solvable, but only those that are also 'periodic', in the sense to be made precise later.

Fact 2 (Theorem 3 in [5]). *A circular word $[w]_{\triangleq}$ is solvable iff $|x\alpha|^x_y > |w|^x_y$, for its every subword $x\alpha y$ with $x \neq y$.*

Following the above result, a circular word $[w]_{\triangleq}$ is *balanced* if, for its every subword $x\alpha y$ with $x \neq y$:

$$|x\alpha|^x_y > |w|^x_y. \tag{1}$$

Moreover, if (1) does not hold for $x\alpha y$, then $[w]_{\triangleq}$ is *$x\alpha$-unbalanced*. It turns out that for circular block-words it suffices to concentrate on block-subwords.

Proposition 1. *A circular block-word $[w]_{\triangleq}$ is not balanced iff there is a block-subword w' of $[w]_{\triangleq}$ such that $[w]_{\triangleq}$ is w'-unbalanced or bw'-unbalanced.*

Proof. Suppose that $[w]_{\triangleq}$ is $a\alpha$-unbalanced. Then we can extend $a\alpha$ by some b^j ($j \geq 0$) obtaining a block-subword $w' = a\alpha b^k$ such that $[w]_{\triangleq}$ is w'-unbalanced.

Suppose that $[w]_{\triangleq}$ is $b\alpha$-unbalanced. If $|\alpha|_a = 0$, then $[w]_{\triangleq}$ cannot be $b\alpha$-unbalanced. Hence $b\alpha = b^j w'$, where $j \geq 1$ and w' is a block-subword, and $[w]_{\triangleq}$ is bw'-unbalanced. □

Let w be a block-word and $\alpha\gamma\beta\delta \triangleq w$, where $\alpha, \gamma, \beta, \delta$ are block-subwords of $[w]_{\triangleq}$. Then α and β are *complementary* if $\gamma = \delta = \varepsilon$, and *mutually unbalanced*, denoted by $\alpha \frown \beta$, if $\frac{|\alpha|_b+1}{|\alpha|_a} \leq \frac{|\beta|_b-1}{|\beta|_a}$ or $\frac{|\beta|_b+1}{|\beta|_a} \leq \frac{|\alpha|_b-1}{|\alpha|_a}$.

An alternative presentation of the *balancing property* of circular words is obtained by considering complementary block-subwords. It will later be used to show that for any unbalanced block-word one can construct a pair of mutually unbalanced complementary block-subwords.

Proposition 2. *Let α and β be complementary block-subwords of $[w]_{\triangleq}$.*

- *If $[w]_{\triangleq}$ is $b\alpha$-unbalanced or α'-unbalanced, where $\alpha' b = \alpha$, then $\alpha \frown \beta$.*
- *If $\alpha \frown \beta$, then w is not balanced.*

Proof. Let m and k be the numbers of blocks in α and β, respectively. Then $|b\alpha|_a^b = \frac{|\alpha|_b+1}{m} \leq |w|_a^b = \frac{|\beta|_b-1+|\alpha|_b+1}{m+k}$ is equivalent to

$$m|\alpha|_b + m + k|\alpha|_b + k \leq m|\alpha|_b + m + m(|\beta|_b - 1)$$

which is equivalent to $k(|\alpha|_b + 1) \leq m(|\beta|_b - 1)$. And the latter is equivalent to $\frac{|\alpha|_b+1}{m} \leq \frac{|\beta|_b-1}{k}$.

Moreover, we have that $|a\alpha'|_b^a = \frac{m}{|a\alpha' b|_b-1} \leq |w|_b^a = \frac{m+k}{|a\alpha' b|_b-1+1+|\beta|_b}$ is equivalent to $m(|\beta|_b + 1) \leq k(|a\alpha' b|_b - 1)$ which is equivalent to $\frac{|a\alpha' b|_b-1}{m} \geq \frac{|\beta|_b+1}{k}$. □

Next we show that as soon as we have two mutually unbalanced block-subwords, one can extend one of them by the block-subword positioned 'in between' the original sub-words.

Proposition 3. *Let $w = \alpha\gamma\beta\delta$ be a block-word such that $\alpha, \beta, \gamma, \delta$ are block-subwords of $[w]_{\triangleq}$ and $\alpha \frown \beta$. Then: (i) $\alpha\gamma \frown \beta$ or $\alpha \frown \gamma\beta$; and (ii) $\alpha \frown \beta\delta$ or $\delta\alpha \frown \beta$.*

Proof. (i) It is a consequence of the following general result. Whenever we have $\frac{m_1-1}{n_1} \geq \frac{m_2+1}{n_2}$, then for all p, q with $q \neq 0$, we have either $\frac{(m_1+p)-1}{n_1+q} \geq \frac{m_2+1}{n_2}$ or $\frac{m_1-1}{n_1} \geq \frac{(m_2+p)+1}{n_2+q}$. Indeed, the former is clearly obtained when $\frac{p}{q} > \frac{m_1}{n_1}$, and the latter when $\frac{p}{q} < \frac{m_2}{n_2}$. In general, if $\frac{p}{q} \geq \frac{m_2+1}{n_2}$ then, by $\frac{m_1-1}{n_1} \geq \frac{m_2+1}{n_2}$, we have $\frac{m_1-1}{n_1} \geq \frac{(m_2+p)+1}{n_2+q}$. And, by symmetry, if $\frac{p}{q} \leq \frac{m_1-1}{n_1}$ then $\frac{m_1-1}{n_1} \geq \frac{(m_2+p)+1}{n_2+q}$.

Note: The two cases considered in the second part of the proof cover all possibilities, but the previous easy cases are included to improve readability.

(ii) The proof is similar as for (i) due to the fact that δ lies 'in between' β and α in the circular sense. □

A block-word $w = ab^{x_1}ab^{x_2}\dots ab^{x_n}$ is *block-balanced* if, for all $1 \leq j \leq k \leq l \leq m \leq n$:

$$\frac{\sum_{i=j}^{k} x_i}{k-j+1} > \frac{\sum_{i=l}^{m} x_i}{m-l+1} \implies \frac{\sum_{i=j}^{k} x_i - 1}{k-j+1} < \frac{\sum_{i=l}^{m} x_i + 1}{m-l+1} \tag{2}$$

and

$$\frac{\sum_{i=j}^{k} x_i}{k-j+1} < \frac{\sum_{i=l}^{m} x_i}{m-l+1} \implies \frac{\sum_{i=j}^{k} x_i + 1}{k-j+1} > \frac{\sum_{i=l}^{m} x_i - 1}{m-l+1}. \tag{3}$$

Moreover, if all block-words in $[w]_{\triangleq}$ are block-balanced, then w is *circular block-balanced*. (Note that in the case of circular block-balancing, if all block-words in $[w]_{\triangleq}$ satisfy (2), then they satisfy (3) as well.) The block-word w is *contiguously block-balanced* if the conditions (2) and (3) are satisfied for $l = k + 1$. And w is *contiguously circular block-balanced* if all the members of $[w]_{\triangleq}$ are contiguously block-balanced.

Theorem 1. *A block-word is cyclic solvable iff it is circular block-balanced.*

Proof. Applying Proposition 3 we only need to observe that once we have $w = \alpha\gamma\beta\delta$, which is circular block unbalanced because $\alpha \frown \beta$, then $\alpha\gamma \frown \beta$ or $\alpha \frown \gamma\beta$, will be also mutually unbalanced. Suppose (wlog) that the former holds. Then another application of Proposition 3 implies that $\alpha\gamma \frown \beta\delta$ or $\alpha \frown \gamma\beta\delta$ holds, and an application of Proposition 2 concludes the proof. □

Corollary 1. *A block-word is cyclic solvable iff it is contiguously circular block-balanced.*

Proof. Follows immediately from Theorem 1 and Proposition 3.

Propositions 8 and 9 in [5] imply that solvable circular words can be solved by 'circular' nets with two places and two transitions. For all $A, B \geq 1$, let $\mathcal{N}_{AB}$ be the set of all nets $N_{AB} = (\{p_a, p_b\}, \{a, b\}, F, M_0)$ such that $M_0(p_a) + M_0(p_b) = A + B - 1$ and the non-zero values for F are $F(p_a, a) = F(a, p_b) = A$ and $F(p_b, b) = F(b, p_a) = B$.

Fact 3 (follows from Proposition 9 in [5]). *N_{AB} solves the unique solvable circular word $[w_{AB}]_{\triangleq}$ satisfying $|w_{AB}|_a = B$ and $|w_{AB}|_b = A$. Also, if $gcd(A, B) = 1$ then w_{AB} is prime, and if $gcd(A, B) = C > 1$ then $w = u^C$, where u is prime.*

A general result about the extendability of solvable words by prefixes is

Fact 4 (Proposition 4 in [2]). *If aw is solvable, then aaw is also solvable.*

In other words, a prefix a can be extended to a^k without losing solvability. This is not always possible using b instead of a, since otherwise any word would be solvable. However, any cyclic solvable word can be prefixed by a^k or b^k, for any $k \geq 1$, yielding a solvable word (although, in general, not cyclic solvable, and not even a subword of a cyclic solvable word). It seems that this fact was not yet used in the literature, and it has indeed been one of the key observations leading to our new characterization of (plain) solvable words.

Proposition 4. *Let $A, B, k \geq 1$ and $N_{AB} = (\{p_a, p_b\}, \{a, b\}, F, M_0) \in \mathcal{N}_{AB}$ be a net solving w^ω. Moreover, let $N' = (\{p_a, p_b\}, \{a, b\}, F', M'_0)$ be a net such that:*

- *$F'(p_a, a) = F'(a, p_b) = A$, $F'(p_b, b) = B + k$, $F'(b, p_y) = kA$, and $F'(b, p_a) = B$ are the non-zero values of F', and*
- *$M'_0(p_a) = kA + M_0(p_a)$ and $M'_0(p_b) = kA + M_0(p_b)$.*

Then N' solves $a^k w^\omega$.
Note: A symmetric construction solves $b^k w^\omega$.

Proof. The kA additional tokens in p_a allow the firing of a^k. At the same time, b is not enabled since $M_0(p_b) + (k-1)A < B + kA$ as $M_0(p_b) > A + B$. After the firing of a^k, the kA produced tokens will remain *frozen* at p_b, and the remaining tokens will 'constitute' the initial marking M_0 for N_{AB}, so that the subsequent behaviour of N' will generate w^ω. □

A direct consequence of the last result is that all the infinite words represented by $x^k w^\omega$, where w is a cyclic solvable word, are solvable.

Given an integer $e \geq 2$, a block-word $ab^{x_1} \dots ab^{x_m}$ is an *e-block-word* if $x_1, \dots, x_m \in \{e-1, e\}$. From the results in [5], it immediately follows that each cyclic solvable block-word is an e-block-word, for some $e \geq 2$. We next apply our characterization of cyclic solvable words to demonstrate a *duality* between the ab^{e-1} blocks and ab^e blocks, which can be swapped without affecting solvability.

Proposition 5. *An e-block-word $ab^{x_1} \dots ab^{x_m}$ is cyclic solvable iff the e-block-word $ab^{2e-1-x_1} ab^{2e-1-x_2} \dots ab^{2e-1-x_m}$ is cyclic solvable.*

Proof. We first observe that $\frac{\sum_{s=i}^{i+m-1} 2e-1-x_s}{m} = 2e-1-\frac{\sum_{s=i}^{i+m-1} x_s}{m}$. Hence we have that $\frac{\sum_{s=i}^{i+m-1} x_s}{m} > \frac{\sum_{t=j}^{j+k-1} x_t}{k} \implies \frac{\sum_{s=i}^{i+m-1} x_s - 1}{m} < \frac{\sum_{t=j}^{j+k-1} x_t + 1}{k}$ is equivalent to:

$$\frac{\sum_{s=i}^{i+m-1} (2e-1-x_s)}{m} < \frac{\sum_{t=j}^{j+k-1} (2e-1-x_t)}{k} \implies \frac{\sum_{s=i}^{i+m-1} (2e-1-x_s)+1}{m} > \frac{\sum_{t=j}^{j+k-1} (2e-1-x_t)-1}{k}. \square$$

3.1 An Efficient Algorithm to Detect Cyclic Solvable Words

Having observed (wlog) that all the solvable cyclic block-words are e-block-words (i.e., they are built from *two* kinds of blocks), one can intuitively foresee the *internal periodicity* of cyclic solvable block-words. This means that the full word w not only appears as period of the corresponding infinite (cyclic) word, but also the internal structure of w is *periodic*, and the blocks are distributed in a periodic way.

Definition 1. *The* derivative words *of an e-block-word w are defined as follows:*

- *$\partial_1(w)$ is obtained by replacing in w each block ab^{e-1} by 1, and each block ab^e by 2.*
- *$\partial_2(w)$ is obtained by replacing in w each block ab^{e-1} by 2, and each block ab^e by 1.*

We use as alphabet $\{1, 2\}$ in order to reflect the numerical information contained in the blocks. Note that both derivative words ignore the actual value of e.

As shown later in Corollary 2, cyclic solvable e-block-words cannot contain both $ab^{e-1}ab^{e-1}$ and $ab^e ab^e$. Hence, we complete the definition of derivative words in the following way.

Definition 2. *An e-block-word w is* derivable *if it does not contain $ab^{e-1}ab^{e-1}$ or ab^eab^e. Moreover, its* derivative *is defined as $\partial(w) = \partial_2(w)$ if the former holds, and as $\partial(w) = \partial_1(w)$ otherwise.*

A word $b^{x_0}ab^{x_1}\dots ab^{x_m}ab^{x_{m+1}}$ is semi-derivable *if $ab^{x_1}\dots ab^{x_m}$ is derivable and $1 \leq x_0, x_{m+1} \leq \min\{x_1,\dots,x_m\}+1$.*

Hence we have $\partial(w) = 1^{y_0}21^{y_1}21^{y_2}\dots21^{y_n}$, for some $n \geq 1$ and $y_1,\dots,y_n \geq 1$. Moreover, inverse transformations can *integrate* words like $1^{y_0}21^{y_1}21^{y_2}\dots21^{y_n}$. Since we can choose any $e \geq 2$, and either ab^{e-1} or ab^e to appear more often this can be done in infinitely many ways.

Definition 3. *For each $e \geq 2$ and $v = 1^{y_0}21^{y_1}21^{y_2}\dots21^{y_n}$, we define:*

$$\int_{e,1} v = \phi_{1\mapsto ab^{e-1},2\mapsto ab^e}(v) \quad \text{and} \quad \int_{e,2} v = \phi_{1\mapsto ab^e,2\mapsto ab^{e-1}}(v)$$

where $\phi_{1\mapsto z,2\mapsto z'}$ is a morphism replacing 1 by z and 2 by z'.

Hence we always have $\partial_i(\int_{e,i} v) = v$, for all $i \in \{1,2\}$ and $e \geq 2$.

We will now aim at a *structural* characterization of cyclic solvable words, that will lead to an (optimal!) algorithm detecting such words. In order to see how the relationship between (un)balanced words and their derivatives works, we discuss a simple example.

Consider $v = 212111$ which contains two mutually unbalanced block-subwords 21 and 2111 (note that $\frac{3-1}{1} \not< \frac{1+1}{1}$). Let us have a look at its integrals $w_1 = \int_{2,1} v = (ab^2ab)(ab^2abababab)$ and $w_2 = \int_{2,2} v = (ababab^2)(ababab^2ab^2ab^2)$, where the parenthesis separate contributions of the two blocks in v. Both w_1 and w_2 are unbalanced. In the first case, we can take the block-subwords $w_{11} = ab^2abab^2$ and $w_{12} = abababab$ satisfying:

$$\frac{|w_{11}|_b-1}{|w_{11}|_a} = \frac{2\cdot2+1\cdot1-1}{2+1} \not< \frac{0\cdot2+3\cdot1+1}{0+3} = \frac{|w_{12}|_b+1}{|w_{12}|_a}.$$

In the second case, we can take $w_{21} = ab^2ab^2ab^2$ and $w_{22} = ababab^2ab$ satisfying:

$$\frac{|w_{21}|_b-1}{|w_{21}|_a} = \frac{0\cdot1+3\cdot2-1}{0+3} \not< \frac{2\cdot1+1\cdot2+1}{2+1} = \frac{|w_{22}|_b+1}{|w_{22}|_a}.$$

There are two things to be considered. The first is that ∂_2 works in a *covariant* way, so that 'wherever' v has 'too many' 1's, w_2 has in turn 'too many' b's. On the other hand, ∂_1 works exactly the other way around, i.e., in a *contravariant* way. Besides, in both cases the block-subwords that now are mutually unbalanced do not coincide exactly with the integrals of the mutually unbalanced block-subwords in v. Indeed, w_{11} and w_{22} (observe again the duality!) need to expand the corresponding block-subword, borrowing the following block, while w_{12} and w_{12} lose their first blocks. In the general case, we have the following:

Theorem 2. *An e-block-word w is cyclic solvable iff w is derivable and $\partial(w)$ is cyclic solvable.*

Proof. First, if w contains both $ab^{e-1}ab^{e-1}$ and ab^eab^e, then it is not cyclic solvable. Let us suppose that it contains ab^eab^e, which corresponds to the covariant case, as explained in the introduction to the proof. Now, if $\partial(w)$ is not cyclic solvable, then its alternative presentation $v' = 21^{y_1}21^{y_2}\dots21^{y_n+y_0}$ is not either. To simplify the notation, we consider first the case $y_0 = 0$. Applying Theorem 1, we should have two block-subwords $21^{y_i}21^{y_{i+1}}\dots21^{y_{i+p-1}}$ and $21^{y_j}21^{y_{j+1}}\dots21^{y_{j+k-1}}$, with $\frac{\sum_{s=i}^{i+p-1} y_s - 1}{p} \geq \frac{\sum_{t=j}^{j+k-1} y_t + 1}{k}$. If we consider the corresponding block-subwords in $\int_{e,2} v$:

- $w_1 = ab^{e-1}(ab^e)^{y_i}ab^{e-1}(ab^e)^{y_{i+1}}\dots ab^{e-1}(ab^e)^{y_{i+p-1}}$ and $w_1 = ab^{e-1}w_1'$
- $w_2 = ab^{e-1}(ab^e)^{y_j}ab^{e-1}(ab^e)^{y_{j+1}}\dots ab^{e-1}(ab^e)^{y_{j+k-1}}$ and $w_2' = w_2ab^{e-1}$

we have $\frac{|w_1'|_b - 1}{|w_1'|_a} = \frac{(p-1)(e-1)+e\sum_{s=i}^{i+p-1} y_s - 1}{(p-1)+\sum_{s=i}^{i+p-1} y_s} \geq \frac{(k+1)(e-1)+e\sum_{t=j}^{j+k-1} y_t + 1}{(k+1)+\sum_{t=j}^{j+k-1} y_t} = \frac{|w_2'|_b + 1}{|w_2'|_a}$.

And the contravariant case can be treated in a similar way, but reversing the inequality.

For the converse, let us consider an e-block-word $w = ab^{x_1}\dots ab^{x_m}$ and its two block-subwords, w_1' and w_2', satisfying $\frac{|w_1'|_b - 1}{|w_1'|_a} \geq \frac{|w_2'|_b + 1}{|w_2'|_a}$. Reasoning similarly as in Proposition 1, we obtain that $w_1' = (ab^e)^{y_1}\dots ab^{e-1}(ab^e)^{y_m}$, and $w_2' = ab^{e-1}(ab^e)^{y_p}\dots\ (ab^e)^{y_{p+r-1}}ab^{e-1}$. Also, $w_1 = ab^{e-1}w_1'$ and w_2 such that $w_2' = w_2ab^{e-1}$, are block-subwords of w, so that $\partial(w)$ contains the block-subwords $21^{y_1}\dots21^{y_m}$ and $21^{y_p}\dots21^{y_{p+r-1}}$. Hence we can see that all this corresponds to the situation that we had in the first part of the proof, so that reversing our arguments we conclude as required. □

Theorem 3 (efficient recursive algorithm checking cyclic solvability). *The following* PROLOG*-like algorithm checks in linear time the cyclic solvability of an e-block-word $w = ab^{x_1}ab^{x_2}\dots ab^{x_m}$.*

- ***if*** $x_1 = \dots = x_m$
  then *CyclicSolvable*(w)
- ***if*** $x_i = x_{i+1} \neq x_j = x_{j+1}$, *for some* i, j
  then $\neg$*CyclicSolvable*(w)
- ***if*** $w = (ab^e)^f(ab^{e-1}ab^e)^n(ab^{e-1})^g$, *for some* $f, g \in \{0, 1\}$ *and* $n \geq 1$
  then *CyclicSolvable*(w)
  else *CyclicSolvable*(w) = *CyclicSolvable*$(\partial(w))$

Proof. The algorithm checks the cyclic solvability of w by Theorem 2. It terminates in linear time since $|w| \geq 2 \cdot |\partial(w)|$. □

4 Relating Solvable Words and Cyclic Solvable Words

Here we present the main result of this paper, showing that all solvable words can be obtained by prefixing with x^k prefixes of cyclic solvable words.

Theorem 4. *$\{ x^k w \mid k \geq 0$ and w is a prefix of a cyclic solvable word$\}$ is the set of all solvable words.*

In order to prove the ($\supseteq$) inclusion (note that the reverse inclusion holds by Proposition 6), we give an *algebraic* characterization of the set of prefixes of cyclic solvable words. This is very similar to that of the cyclic solvable words in Theorem 1. We start with recalling an important result.

Theorem 5 (Theorem 2 in [5]). *A word w is solvable if, for any decomposition $w = \alpha y \beta x \gamma y \delta$, with $x \neq y$, we have $|y\beta|_x^y > |x\gamma|_x^y$.*

The above condition is equivalent to $\frac{|\beta|_y+1}{|x\gamma|_x} > \frac{|x\gamma|_y}{|\gamma|_x+1}$, and this is indeed very close to the conditions (2) and (3) in the definition of block-balanced words. We have found that these are exactly the prefixes of cyclic solvable words:

Theorem 6. *A word $w = ab^{x_1}ab^{x_2}\ldots ab^{x_m}a$ is a prefix of a cyclic solvable word iff $w' = ab^{x_1}ab^{x_2}\ldots ab^{x_m}$ is block-balanced.*

It is worth to note that the only differences between the statements in Theorem 1 are first Theorem 6 is that in the latter we need to introduce a final a in the word, in order to *signal* the completion of the previous block ab^{x_m}. On the other hand, we have the circular character of circular words, which 'forces' us to consider also block-subwords that can 'glue' the end of the word with its beginning.

As it is the case for cyclic solvable words, the characterization above remains valid when we only check contiguous block-subwords.

Theorem 7. *A word $w = ab^{x_1}ab^{x_2}\ldots ab^{x_m}a$ is a prefix of a cyclic solvable word iff $w' = ab^{x_1}ab^{x_2}\ldots ab^{x_m}$ is contiguously block-balanced.*

To prove the last two theorems we need several auxiliary results.

Proposition 6. *For any $e \geq 2$, no solvable word w contains the subword ab^eab^e before the subword $ab^{e-1}ab^{e-1}a$.*

Proof. Let us consider an occurrence of ab^eab^e before, and as close as possible, the occurrence of $ab^{e-1}ab^{e-1}a$. Then $w = \alpha(ab^eab^e)(ab^{e-1}ab^e)^k(ab^{e-1}ab^{e-1})a\delta$. Let y be the first a after α, $\beta = (b^eab^e)(ab^{e-1}ab^e)^{k-1}(ab^{e-1}ab^{e-1})$, x the last b in the subsequence $(ab^{e-1}ab^e)^k$, $\gamma = ab^{e-1}ab^{e-1}$, and the second y be the last a. Then we get $|a\beta|_a \cdot |b\gamma|_b = (2e-1)\cdot(2k+2)\cdot 2 = (k+1)\cdot(2e-1)\cdot 2 = |a\beta|_b \cdot |b\gamma|_a$, and thus $|a\beta|_a \cdot |b\gamma|_b \not> |a\beta|_b \cdot |b\gamma|_a$. Hence, by Fact 5 w is unsolvable and thus we obtain an obvious contradiction, finishing the proof. □

Corollary 2. *No solvable circular block-word has ab^eab^e and $ab^{e-1}ab^{e-1}$ as block-subwords.*

Proof. In such a case we could always 'put' the occurrence of ab^eab^e 'before' that of $ab^{e-1}ab^{e-1}$, due to the cyclic character of circular words.

Proposition 7. *A solvable word cannot have both ab^eab^e and $ab^{e-1}ab^{e-1}a$ as subwords, after any previous occurrence of b.* □

Proof. By Proposition 6, the only possibility is that we will have $ab^{e-1}ab^{e-1}a$ before ab^eab^e, possibly sharing the last a of the first, but then, reasoning as in the proof of Proposition 6, we should have $w = \alpha b(ab^{e-1}ab^{e-1})(ab^eab^{e-1})^k(ab^eab^e)\delta$. Taking as y the first b after α, $\beta = (ab^{e-1}ab^{e-1})(ab^eab^{e-1})^k$, as x the first a in the last block (ab^eab^e), $\gamma = b^eab^{e-1}$, and as second y the last b, we get $|b\beta|_b \cdot |a\gamma|_a = (2e-1)\cdot(2k+2)\cdot 2 \not> (2k+2)\cdot(4e-2) = |b\beta|_a \cdot |a\gamma|_b$, obtaining again the unsolvability of w. □

Remark 1. It is important to observe that the total duality we had in the case of cyclic solvable words, as shown by Proposition 5, is partially lost when we consider solvable (plain) words. In particular, from Proposition 6 we conclude that ab^2ab^2abab is not a solvable word, while $ababab^2ab^2$ is solvable, because it can be decomposed as $a(babab^2ab^2)$, and $babab^2ab^2ab$ (or $babab^2ab$) is cyclic solvable. However, when considering prefixes of cyclic solvable words, this duality remains valid, since it can be obtained as an immediate consequence of the following more general result.

Proposition 8. *Let $ab^{x_1}ab^{x_2}\dots ab^{x_m}$ be a block balanced e-block-word. Then $ab^{2e-1-x_1}ab^{2e-1-x_2}\dots ab^{2e-1-x_m}$ is block balanced.*

Proof. We only need to observe that $\frac{\sum_{i=j}^{k} x_i - 1}{k-j+1} < \frac{\sum_{i=l}^{m} x_i + 1}{m-l+1}$ is equivalent to $\frac{\sum_{i=j}^{k}(2e-1-x_i)+1)}{k-j+1} > \frac{\sum_{i=l}^{m}(2e-1-x_i)-1}{m-l+1}$, by simply observing that the following holds: $\sum_{i=j}^{k}(2e-1-x_i) = (2e-1)(k-j+1) - \sum_{i=j}^{k} x_i$. □

As we observed for cyclic solvable block-words, *conjugation* also preserves plain block balancing, and this allows to simplify some proofs, by reducing the number of cases, as in the proof of Theorem 8 below.

Theorem 8. *Let $w = ab^{x_1}ab^{x_2}\dots ab^{x_m}$ be a block balanced derivable word, with $\partial(w) = \partial_2(w)$. Then $\partial(w) = 1^{y_0}21^{y_1}2\dots 21^{y_{m+1}}$ is semi-derivable and considering $\mathrm{d}w = 21^{y_s}\dots 21^{y_t}$, where $s = 0$ if $y_0 = \min\{y_1,\dots,y_m\}+1$ and $s = 1$ otherwise; and $t = m+1$ if $y_{m+1} = \min\{y_1,\dots,y_m\}+1$ and $t = m$ otherwise, the block-subword $\phi_{1\to a,2\to b}(\mathrm{d}w)$ is block balanced.*

Proof. 1. Suppose that $y_{i_1} < y_{i_2} - 1$, where $i_1, i_2 \in \{1,\dots,m\}$. Then, the corresponding subwords in w, $w_1 = ab^{e-1}(ab^e)^{y_1}ab^{e-1}a$ and $w_2 = (ab^e)^{y_2}a$, are clearly not block balanced since $\frac{ey_1+2(e-1)+1}{y_1+2} \le \frac{ey_2-1}{y_2}$ is equivalent to $y_1 \le y_2 - 2$.

2. Exactly as above, once w contains a full sequence of blocks $(ab^e)^{f-1}$, neither y_0 nor y_{m+1} can be greater than f.

3. First notice that $\partial(w')$ can indeed contain a nonempty prefix 1^{y_0}, but at the moment we will ignore this, assuming that $y_0 = 0$. In the same way, we will remove the (possibly) *incomplete* final block $21^{y_{m+1}}$, only keeping it when $y_{m+1} = e$. Note that the obtained word $21^{y_1}\dots 21^{y_t}$ remains block balanced. Let us suppose that this is not the case. Then we have two subwords, $21^{y_j}\dots 21^{y_{j+m-1}}$ and $21^{y_k}\dots 21^{y_{k+r-1}}2$, with $\frac{\sum_{i=j}^{j+m-1} y_i - 1}{m} \ge \frac{\sum_{i=k}^{k+r-1} y_i + 1}{r}$.

To simplify, we write $T_1 = \sum_{i=j}^{j+m-1} y_i$ and $T_2 = \sum_{i=k}^{k+r-1} y_i$, so that we have $rT_1 - r \geq mT_2 + m$.
Let us consider the two subwords of w' that generate the two subwords of the derivative given above, $w_1 = ab^{e-1}(ab^e)^{y_j} \dots ab^{e-1}(ab^e)^{y_{j+m-1}}$ and $w_2 = ab^{e-1}(ab^e)^{y_k} \dots ab^{e-1}(ab^e)^{y_{k+r-1}}$. We take w'_1, where $w_1 = ab^{e-1}w'_1$, and $w'_2 = w_2 ab^{e-1}$. Hence $\frac{|w'_1|_b - 1}{|w'_1|_a} \geq \frac{|w'_2|_b + 1}{|w'_2|_a}$.
Since we have $|w'_1|_b = eT_1 + (m-1)(e-1)$, $|w'_1|_a = T_1 + (m-1)$, $|w'_2|_b = eT_2 + (r+1)(e-1)$, and $|w'_2|_a = T_2 + r + 1$, we obtained exactly the same situation as in the proof of Theorem 2.
And this gives $(|w'_1|_b - 1)|w'_2|_a \geq (|w'_2|_b + 1)|w'_1|_a \iff rT_1 - r \geq mT_2 + m$.
Finally, when the initial prefix is 1^e, the result remains valid for the whole word $21^{y_0} \dots 21^{y_t}$. Note that, although we have introduced an additional 2 at the beginning of the derivative in the case $y_0 = 0$, we have also 'removed' the prefix ab^{e-1} of w_1, and proceded with its *continuation* w'_1. Now, we simply avoid that complication, working with the full word w_1, and the rest of the proof works in the same way as before. □

Now we can present the proofs of Theorems 6 and 7:

Proof (Theorem 6). By the induction on the number of blocks of w. The base case (i.e., 0) is vacuous. For the inductive case, we first apply Proposition 8 if needed, so that we can assume $\partial(w) = \partial_2(w)$. Hence, we can apply Theorem 8 to conclude that $\phi_{1\to a,2\to b}(\mathrm{d}w)$ is also block-balanced, so that we can apply the induction hypothesis, getting that it is a prefix of a cyclic solvable word, cw. Now, we can consider $c = \int_{e,2} cw$, that contains w as a subword, since the removed prefix and suffix (if that was the case) would clearly fit into the blocks before and after $\mathrm{d}w$ in cw. Finally, to complete the proof, we just need to undo the conjugation, if that was applied at the beginning. By Proposition 8 and Proposition 5 conjugation preserves cyclic solvability. Hence it can be applied to prefixes of the form $w = ab^{x_1}ab^{x_2} \dots ab^{x_m}a$. □

Proof (Theorem 7). The result follows immediately from Theorem 6, using the arguments from the proof of Proposition 3. □

We show that the conditions characterizing solvable words in [5], recalled here as Theorem 5, are nearly equivalent with our block balancing property, as the following lemmata states:

Lemma 1

1. *Let w be a word and its decomposition $w = \alpha y \beta x \gamma y \delta$ disproves the solvability of w with the use of Theorem 5. Then $\beta = x\beta' y$, and $\delta = \epsilon$ or $\delta = x\delta'$.*
2. *Let $w = a^k w'$ and $w' = b^{x_0}ab^{x_1} \dots ab^{x_m}ab^{x_{m+1}}$, where $x_1, \dots, x_m \in \{e, e-1\}$ and $x_0 > 0$, for $e = \min\{x_1, \dots, x_m\} + 1 > 1$. Then:*
   (a) If $y = a$, then $y\beta = ab^{x_i} \dots ab^{x_{i+n_1-1}-1}$ and $x\gamma = bab^{x_{i+n_1}} \dots b^{x_{i+n_1+n_2-1}}$.
   (b) If $y = b$, then $y\beta = bab^{x_i} \dots ab^{x_{i+n_1-1}}$ and $x\gamma y = ab^{x_{i+n_1}} \dots b^{x_{i+n_1+n_2-1}}$.

Proof. (1) If $\beta = y\beta'$, then β' makes $|\beta'|_y < |\beta|_y$, and thus can be used instead of β. And, similarly, if $\beta = \beta' x$, then we can take $\gamma' = x\gamma$ instead of γ; while if $\delta = y\delta'$, we can take $\gamma' = \gamma y$ instead of γ.
(2) This is an application of part (1) to this particular class of words. □

Lemma 2. *Let $w = a^k w'$ and $w' = b^{x_0} a b^{x_1} \dots a b^{x_m} a b^{x_{m+1}}$, where $x_1, \dots, x_m \in \{e, e-1\}$ and $x_0 > 0$, for $e = \min\{x_1, \dots, x_m\} + 1 > 1$, and the decomposition $w = \alpha y \beta x \gamma y \delta$ disproves its solvability with the use of Theorem 5. Then:*

1. *If $y\beta = a b^{x_i} \dots a b^{x_{i+n_1-1}-1}$ and $x\gamma = b a b^{x_{i+n_1}} \dots b^{x_{i+n_1+n_2-1}}$, then $y\beta b \frown \gamma$, and so $a b^{x_i} \dots a b^{x_{m+1}}$ is not a prefix of a cyclic solvable word.*
   Conversely, if $u \frown v$ are sequences of blocks with $\frac{|u|_b - 1}{|u|_a} \geq \frac{|v|_b + 1}{|v|_a}$, then $\alpha u v \delta$ is unsolvable, for all α and δ.
2. *If $y\beta = b a b^{x_i} \dots a b^{x_{i+n_1-1}-1}$ and $x\gamma y = a b^{x_{i+n_1}} \dots b^{x_{i+n_1+n_2-1}}$, then $\beta \frown a\gamma b$, and so $a b^{x_i} \dots a b^{x_{m+1}}$ is not a prefix of a cyclic solvable word.*
   Conversely,if $u \frown v$ are sequences of blocks with $\frac{|u|_b + 1}{|u|_a} \leq \frac{|v|_b - 1}{|v|_a}$, then $\alpha b u v \delta$ is unsolvable, for all α and δ.

Proof. (1) $\frac{|a\beta|_a}{|a\beta|_b} = \frac{|u|_a}{|u|_b - 1} \leq \frac{|v|_a}{|v|_b + 1} = \frac{|b\gamma|_a}{|b\gamma|_b}$.
(2) $\frac{|b\beta|_b}{|b\beta|_a} = \frac{|u|_b - 1}{|u|_a} \leq \frac{|v|_b + 1}{|v|_a} = \frac{|a\gamma|_b}{|a\gamma|_a}$.
Note that these two cases are not totally symmetric, because in the second case introducing at least one b before u is needed to obtain an unsolvable word. □

And finally we can present the proof of our main theorem.

Proof (Theorem 4). Let $w = a^k w'$ and $w' = b^{x_0} a b^{x_1} \dots a b^{x_m} a b^{x_{m+1}}$, where $x_1, \dots, x_m \in \{e, e-1\}$ and $x_0 > 0$, for $e = \min\{x_1, \dots, x_m\} + 1 > 1$. We consider two possible cases:

Case 1: $k = 0$. Then $w = b^{x_0} \dots a b^{x_m} a b^{x_{m+1}}$. Suppose that the word $w' = a b^{x_1} \dots a b^{x_m} a b^{x_{m+1}}$ cannot be extended to a cyclic solvable word. Then, by Theorem 7, there exists a pair of contiguous block-subwords $u \frown v$ of w'. This gives us $w = \alpha b u v \delta$, for some words α and β, and we can apply one of the cases of Lemma 2, whether we have $\frac{|u|_b - 1}{|u|_a} \geq \frac{|v|_b + 1}{|v|_a}$ or $\frac{|u|_b + 1}{|u|_a} \leq \frac{|v|_b - 1}{|v|_a}$, getting in both cases that w would be unsolvable, against the hypothesis.

Case 2: $k > 0$. Then we have $w = a^k b^{x_0} a b^{x_1} \dots a b^{x_m} a b^{x_{m+1}}$, and applying Lemma 2(2) again, if $a b^{x_1} \dots a b^{x_m} a b^{x_{m+1}}$ would not be a prefix of a cyclic solvable word, then $b a b^{x_1} \dots a b^{x_m} a b^{x_{m+1}}$ would not be solvable, which cannot be the case. Therefore, $a b^{x_1} \dots a b^{x_m} a b^{x_{m+1}}$ must be a prefix of a cyclic solvable word, and then $b^{e-1} a b^{x_1} \dots a b^{x_m} a b^{x_{m+1}}$ is also a prefix of a cyclic solvable word.

Then, the only remaining case to check corresponds to $x_0 = e$, but then $w' = b^{x_0} \dots a b^{x_m} a b^{x_{m+1}}$ can be extended to a cyclic solvable word iff $aw = w' = a b^{x_0} \dots a b^{x_m} a b^{x_{m+1}}$ can be extended. But if this would not be the case, since w is assumed to be solvable, there should exist a decomposition disproving its solvability with $i = 0$. This corresponds to the case $y = a$. But then, by Lemma 2(2), we conclude again that w is unsolvable. □

4.1 An Efficient Algorithm to Detect Solvable Words

Having established that the extensions of prefixes of cyclic solvable words with an additional prefix of the form x^k are the only solvable words, we can now verify solvability. For this purpose we use the constructive proof of Theorem 8, combined with Theorem 2, that states that derivation both preserves and reflects cyclic solvability.

Theorem 9 (efficient recursive algorithm to check solvability). *The following* PROLOG*-like algorithm with clauses applied in indicated order checks in linear time the solvability of* w*.*

- ***if*** $w = x^k y w'$ *and* $k > 0$
  then $Solvable(w) = PrefixCyclicSolvable(yw')$
- ***if*** $w = \alpha x x \beta y y \gamma$ *and* $x \neq y$
  then $\neg PrefixCyclicSolvable(w)$
- ***if*** $w = x^{k_1} y^j x^{k_2}$*, for* $k_1, k_2 \in \mathbb{N}$ *and* $j \in \{0, 1\}$
  then $PrefixCyclicSolvable(w)$

Now $w = b^{x_0} a b^{x_1} a b^{x_2} \ldots a b^{x_m} a b^{x_{m+1}}$*, where* $m \geq 1$ *and* $x_1, \ldots, x_m \geq 1$ *and* $x_0, x_{m+1} \in \mathbb{N}$*.*

- ***if*** $x_i > x_j + 1$*, for some* $j \in \{1, \ldots, m\}$ *and* $i \in \{0, \ldots, m+1\}$
  then $\neg PrefixCyclicSolvable(w)$
- ***if*** $x_1 = \cdots = x_m$
  then $PrefixCyclicSolvable(w)$
- ***if*** $x_0 < x_j$*, for some* $j \in \{1, \ldots, m\}$
  then $PrefixCyclicSolvable(w) = PrefixCyclicSolvable(ab^{x_1} \ldots ab^{x_{m+1}})$
  else $PrefixCyclicSolvable(w) = PrefixCyclicSolvable(ab^{x_0} \ldots ab^{x_{m+1}})$
- ***if*** $x_{m+1} < x_j$*, for some* $j \in \{1, \ldots, m\}$
  then $PrefixCyclicSolvable(w) = PrefixCyclicSolvable(ab^{x_1} \ldots ab^{x_m} a)$
  else $PrefixCyclicSolvable(w) = PrefixCyclicSolvable(ab^{x_1} \ldots ab^{x_{m+1}} a)$
- ***if*** $w = (ab^e)^f (ab^{e-1}ab^e)^k (ab^{e-1})^g$*, for some* $f, g \in \{0, 1\}$ *and* $k \in \mathbb{N}$
  then $PrefixCyclicSolvable(w)$
  else $PrefixCyclicSolvable(w) = PrefixCyclicSolvable(\partial(w))$

Proof. Follows immediately from Theorems 2 and 8. Applying the latter we know that whenever the algorithm declares w not extendable to a cyclic solvable word, because its derivative was proved not to be such, the decision is sound. And if the algorithm, after some derivatives, finally reaches some trivial prefix of a cyclic solvable word, then the application of the former theorem proves that after the corresponding number of integrations, we would obtain a cyclic solvable word that extends the considered word w.

The algorithm terminates in linear time, since $|w| \geq 2 \cdot |\partial w|$. □

Let us consider $w = b^8(ab)^2ab^2(ab)^4ab^2(ab)^3ab^2(ab)^4ab^2(ab)^4ab^2ab$. After removing the prefix b^8, we obtain w' with $\partial(w') = 1^2 21^4 21^3 21^4 21^4 21$, which once the short prefix and suffix are removed, can be translated to $w'' = ab^4ab^3ab^4ab^4a$.

Now we have $\partial(w'') = 121^2$, which produces the simple word ab^2, which clearly is a prefix of a cyclic solvable word. Thus w is solvable.

One can show in an 'effective' way that the analysis was correct by constructing the primitives of the involved words, that will be cyclic solvable words extending them (see Fig. 2). In particular, we see first that 21^2 proves that $\partial(w'')$ is a prefix of $(1)(21^2)(21)$, while the first integration produces the cyclic solvable word $cw'' = ab^4ab^3ab^4$, by means of which we see that w'' is a prefix of $cw'' \cdot cw''$. And finally, a new integration produces the cyclic solvable word $cw' = (ab)^2ab^2(ab)^4ab^2(ab)^3ab^2(ab)^2$, by means of which we see that taking $w = b^8w'$, w' is a prefix of $cw' \cdot cw'$.

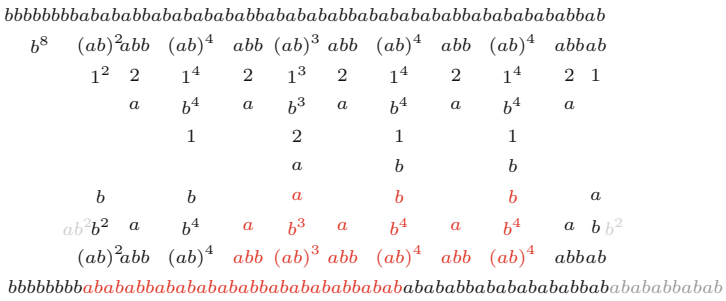


Fig. 2. Procedure of checking solvability and deriving a circular word for suffix.

5 Characterization of Reversible Binary Words

We start by recalling the definitions of strict reverses and reversible words. A *(strict) reverse* of a transition $t \in T$ in a net $N = (P, T, F, M_0)$ is a new transition $\overline{t}$ such that $F(p, \overline{t}) = F(t, p)$ and $F(\overline{t}, p) = F(p, t)$. Following [3], we now relate the reversibility of all transitions with the solvability of the reversed sequence.

A binary word $w = t_1 \dots t_n$ is *reversible* if the following flts is solvable:

$$\overline{TS}_w = (\{0, \dots, n\}, \{a, b\}, \{(i-1, t_i, i), (i, \overline{t_i}, i-1) \mid 0 < i \leq n\}, 0).$$

Fact 5. *A solvable binary word w is reversible iff w^{rev} is solvable.*

We continue by showing that any cyclic solvable word is reversible, and so any prefix of a cyclic solvable word is also reversible.

Proposition 9. *A binary word w is cyclic solvable iff w^{rev} is cyclic solvable.*

Proof. By Fact 3, any cyclic solvable word w can be generated by a net N_{AB}, with $A = |w|_a, B = |w|_b$. One can show that by reversing the arrows in N_{AB}, and starting from the same initial marking, we generate exactly w^{rev}, recovering again the initial marking, and thus proving that w^{rev} is cyclic solvable. □

Corollary 3. *Each cyclic solvable binary word is reversible.*

Proof. Follows immediately from Fact 5 and Proposition 9. □

We can further illustrate the last result by showing that in every $N_{AB} = (\{p_a, p_b\}, \{a, b\}, F, M_0) \in \mathcal{N}_{AB}$ one can introduce strict reverses without enlarging the set of reachable markings. We define $\overline{N}_{AB} = (\{p_a, p_b\}, \{a, \overline{a}, b, \overline{b}\}, \overline{F}, M_0)$, where the non-zero values of $\overline{F}$ are $\overline{F}(p_a, a) = \overline{F}(a, p_b) = \overline{F}(p_b, \overline{a}) = \overline{F}(\overline{a}, p_a) = A$, and $\overline{F}(p_b, b) = \overline{F}(b, p_a) = \overline{F}(p_a, \overline{b}) = \overline{F}(\overline{b}, p_b) = B$. Similarly as in N_{AB}, every reachable marking M of $\overline{N}_{AB}$ satisfies the same property as the initial marking, namely $M(p_a) + M(p_b) = A + B - 1$, and the net is reversible (i.e., from every marking one can reach the initial marking). As a consequence, in each reachable marking one can fire either a or b. Similarly, in each reachable marking, one can fire either $\overline{a}$ or $\overline{b}$. Suppose we can fire $\overline{a}$ at M. Then we cannot fire $\overline{b}$, hence we go from M_0 to M, firing a as the last transition. The case of the initial marking is treated similarly, thanks to the reversibility of N_{AB}.

The result above leads to a characterization of reversible words.

Proposition 10. *A binary word $w = a^k b\alpha a b^m$ is reversible iff both $w_s = b\alpha a b^m$ and $w_p = a^k b\alpha a$ are prefixes of cyclic solvable words.*

Proof. In order to be reversible, w must be solvable, and so w_s must be a prefix of a cyclic solvable word. Moreover, w^{rev} must also be solvable, so that w_p^{rev} must be a prefix of a cyclic solvable word. Then, by applying Proposition 9, w_p must be also a prefix of a cyclic solvable word. □

Corollary 4. *One can decide in linear time whether a binary flts can be reversed.*

Proof. Given a word $w = a^k b\alpha a b^m$, we can apply the algorithm in Theorem 9 to both $a^k b\alpha a$ and $b\alpha a b^m$.

Note: The result could also be obtained directly, by applying the characterization from Proposition 10, and so applying the algorithm from Theorem 9 to both $a^k b\alpha a$ and $b^m a \alpha^{rev} b$. □

It seems that the above characterization gives us some space to find reversible words that are not prefixes of cycle solvable words, but this can only happen in very few simple cases.

Theorem 10. *Apart from prefixes of cyclic solvable binary words, the only reversible words are those of the form $a^k(ba)^i b^m$ or $b^k(ab)^i a^m$, where $i \in \{0, 1\}$ or $k = 2 = m$.*

Proof. First, it is clear that all the 'exceptions' in the statement are indeed solvable and reversible. Moreover one can see that they must appear indeed as exceptional cases, since they do not correspond to prefixes of cyclic solvable words, out of a few trivial instances.

Now we concentrate on the words $w = x^n yuzt^m$ with $x \neq y, z \neq t \in \{a, b\}$, such that $w' = yuz$ contains either aa or bb. We assume the latter. Using the

results about the decomposition of solvable words, we know that all the blocks ab^i, and reversed blocks b^ia in w', must be either ab^{e-1} or ab^e (resp. $b^{e-1}a$ or b^ea), for some $e \geq 2$. Moreover, if $x = a$ then $n = 1$, and when $t = a$ then $m = 1$. While when $x = b$, the cases in which $m < e$ will immediately generate a prefix of a cyclic solvable word, since $w_p = x^n w'$ is such a word; and, similarly, when $t = b$. Hence, we can concentrate in the following on the case $m = e$. Next we distinguish the remaining cases, showing that they never produce reversible words which are not prefixes of cyclic solvable words:

- If $x = b$, then aw_p is a prefix of a cyclic solvable word, and then aw remains solvable. Now, using our characterization of solvable words, w must be a prefix of a cyclic solvable word. The case $t = b$ can be dealt with in a similar (symmetric) way.
- If $x = a$ and $t = a$, we consider the contiguous blocks of b's, b^{i_x} and b^{i_y}.
  - If $i_x = e$ (resp. $i_y = e$), then the cyclic word containing $w_s = w'y^m$ (resp. w_p) clearly contains one a before (resp. after) its occurrence, and so w itself is contained in that cyclic word.
  - If $i_x = e-1$ and $i_y = e-1$ and w is not a prefix of a cyclic solvable word, then w contains two mutually unbalanced contiguous block-subwords that totally cover w_p, since otherwise they would also prove that either w_s or w_p would not be the prefix of some cyclic solvable word, contradicting our assumption. But since the two blocks at the end of these block-subwords will be the same (ab^{e-1}), we could remove them getting two new unbalanced subwords that now do not cover w_p, something that we have already shown to be impossible.

As a result, if w is not a prefix of a cyclic solvable word, then $w' = yuz$ contains neither aa nor bb. Hence it is of the form $(ab)^i$ or $(ba)^i$. □

6 Conclusions

In this paper, we discussed three classes of binary words from the viewpoint of the solvability of the transition systems related to them. For each class, we described a linear algorithm which verifying its membership. Based on our results, one can show every cyclic solvable word is reversible, and every reversible word is solvable (moreover, the two implications cannot be reversed).

A natural direction for further research is to consider larger alphabets, and more sophisticated flts's, for example, those having the shape of directed rooted trees.

Acknowledgement. This research was supported by Cost Action IC1405. The first author was partially supported by the Spanish projects TRACES (TIN2015-67522-C3-3) and N-GREENS (S2013/ICE-2731).

References

1. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47967-4
2. Barylska, K., Best, E., Erofeev, E., Mikulski, Ł., Piątkowski, M.: Conditions for Petri net solvable binary words. In: Koutny, M., Desel, J., Kleijn, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency XI. LNCS, vol. 9930, pp. 137–159. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53401-4_7
3. Barylska, K., Erofeev, E., Koutny, M., Mikulski, Ł., Piątkowski, M.: Reversing transitions in bounded Petri nets. Fundam. Inform. **157**(1–4), 341–357 (2018)
4. Barylska, K., Koutny, M., Mikulski, Ł., Piątkowski, M.: Reversible computation vs. reversibility in Petri nets. Sci. Comput. Program. **151**, 48–60 (2018)
5. Best, E., Erofeev, E., Schlachter, U., Wimmel, H.: Characterising Petri net solvable binary words. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016. LNCS, vol. 9698, pp. 39–58. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39086-4_4
6. Erofeev, E., Barylska, K., Mikulski, Ł., Piątkowski, M.: Generating all minimal Petri net unsolvable binary words. In: Proceedings of PSC 2016, pp. 33–46 (2016)

Pattern Matching in Link Streams: A Token-Based Approach

Clément Bertrand[1(✉)], Hanna Klaudel[1], Matthieu Latapy[2], and Frédéric Peschanski[2]

[1] IBISC, Univ Evry, Université Paris-Saclay, 91025 Evry, France
clement.bertrand@univ-evry.fr
[2] LIP6, Sorbonne Université, Paris, France

Abstract. Link streams model the dynamics of interactions in complex distributed systems as sequences of links (interactions) occurring at a given time. Detecting patterns in such sequences is crucial for many applications but it raises several challenges. In particular, there is no generic approach for the specification and detection of link stream patterns in a way similar to regular expressions and automata for text patterns. To address this, we propose a novel automata framework integrating both timed constraints and finite memory together with a recognition algorithm. The algorithm uses structures similar to tokens in high-level Petri nets and includes non-determinism and concurrency. We illustrate the use of our framework in real-world cases and evaluate its practical performances.

Keywords: Timed pattern recognition · Finite-memory automata
Timed automata · Complex networks · Link streams

1 Introduction

Large-scale distributed systems involve a great number of remote entities (computer nodes, applications, users, etc.) interacting in real-time following very complex network topologies and dynamics. One classical way to observe the behavior of such complex system is to take snapshots of the system at given times and represent the global state as a very large and complex graph. The behavior of the system is then observed as a timed sequence of graphs. The algorithmic detection of patterns of behaviors in such large and dynamic graph sequences is a very complex, most often intractable, problem. The *link stream* formalism [11] has been proposed to model complex interactions in a simpler way. A link stream is a sequence of timestamped links (t, u, v), meaning that an interaction (e.g. message exchange) occurred between u and v at time t. The challenge is to develop analysis techniques that can be performed on the link streams directly, without having to build the underlying global graph sequence. The patterns of interests in link stream involve both structural and temporal aspects, which raises serious challenges regarding the description of such patterns and the design of detection

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 227–247, 2018.
https://doi.org/10.1007/978-3-319-91268-4_12

algorithms. The problems has been mostly approached from two different angles. First, recognition algorithms have been developed for specific patterns such as *triangles* in [12]. The focus is on the performance concerns, involving non-trivial algorithmic issues. At the other end of the spectrum, *complex event processing* (CEP) has been proposed as a higher-level formalism to describe more complex interaction patterns in generic event streams [1,19]. These generic works do not handle the specificity of the input streams, in particular the real-time and graph-related properties of link streams. Our objective is to develop an intermediate approach, generic enough to cover a range of interesting structural and temporal properties, while taking into account the specificities of the link streams abstraction.

Our starting point is that of regular expressions and finite state automata for textual patterns. We interpret link streams as (finite) words and develop a pattern language involving both structural and temporal features. We propose a new kind of hybrid automata, the *timed ν-automata*, as recognizers for this pattern language. They are build upon finite state automata (FSA) with both timed [2,3] and finite-memory [6,7,10] features. The patterns themselves can be specified by enriched regular expressions, and "compiled" to timed ν-automata. The problem of timed pattern matching has been addressed only quite recently in e.g. [14,15,17,18]. While our model bears some resemblance with these propositions, the main novelty is the study of pattern matching in the presence of real-time constraints *together with* finite memory. To our knowledge this has not been addressed in the literature.

One interesting aspect of the automata model we propose is that the recognition principles are based on a non-trivial *token game*. Indeed, our main inspiration comes from high-level Petri nets. Based on this formalism, we developed a prototype tool that we applied to real-world link streams analysis. Performance issues are raised but the results are encouraging. In particular, our experiments confirm the following key fact: timing properties often help in reducing the performance cost induced by storage of information in memory.

The outline of the paper is as follows: In Sect. 2 we introduce the principles of finding patterns in link streams. The automata model and pattern language are formalized in Sect. 3. The prototype tool and practical experiments are then discussed in Sect. 4.

2 Patterns in Link Streams

We consider *link streams* [11] defined as sequences of triples (t_i, u_i, v_i), meaning that we observe a link between nodes u_i and v_i at time t_i. Figure 1 (left) shows an example of a link stream that models interactions between nodes a, b, c and d. For example at time $t = 6$ a link from node d to node b is observed, which corresponds to a triple $(6, d, b)$ in the stream.

A *pattern* in such a link stream can be seen as a series of (directed) subgraphs observed in a given time frame. For example, at time $t = 15$ we observe the subgraph described on the right of Fig. 1. This graph has been formed in the

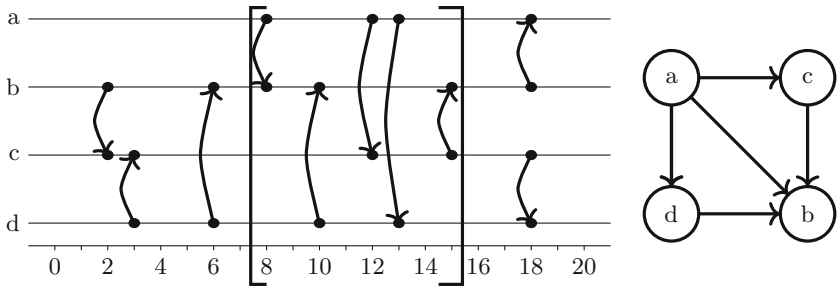


Fig. 1. A link stream (left) and its graph projection in time interval $[8, 15]$ (right).

depicted time frame of 7 s. One trivial way to detect such patterns is to build all the intermediate graphs and solve the subgraph isomorphism problem at each time step. This is however out of reach in most situations most notably because: (1) real-world link streams involve very large graphs, and (2) subgraph isomorphism is a NP-complete problem. Hence, in practice dedicated algorithms are developed for specific kinds of subgraphs. One emblematic example is the *triangle* for which specialized algorithms have been developed. A triangle is simply the establishment of a complete subgraph of three nodes, in a directed way. In network security this is a known trigger for attacks: two nodes that may be identified as "attackers" negotiate to "attack" a third node identified as the "target". Such a trigger can be observed in Fig. 1 (right) with a and d attackers targeting b. In real-world link streams, detecting such triangles is in fact not trivial, as explained for example in [12].

In this paper, our motivation is to develop a more generic approach able to handle not only such triangles but also other kinds of patterns: directed polygons, paths, alternations (e.g. links that appear periodically), etc. We also require the matching algorithms to be of practical use, hence with efficiency in mind. Our starting point is the theory of *finite-state automata* (FSA) and *regular expressions*. Indeed, if we ignore the timestamps, a link stream is similar to a finite word, each symbol being a directed link (a pair of nodes). For example in the time frame $(8, 15)$ we observe the following "word":

$$(a, b)(d, b)(a, c)(a, d)(c, b).$$

Based on such a view, we can use FSA as pattern recognizers and regular expressions as a high-level specification language. A regular pattern for the triangle example is as follows:

$$\Big(((a \to d) \mid (d \to a)) \cdot ((a \to b) \otimes (d \to b))\Big) \otimes (@ \to @)^*$$

This expression uses classical regular constructs such as concatenation $\cdot$, disjunction $|$, the *Kleene* star $*$ and shuffle $\otimes$. The symbol @ is used as a placeholder for any possible node, hence $(@ \to @)$ means "any possible link". Based on

such specification, it is easy to build a finite-state automaton to recognize the triangles in an untimed link stream very efficiently.

However, the "regular language" approach fails to capture the timing properties of link streams. What we need is a form of real-time pattern matching. Quite surprisingly, there are very few research works addressing this problematic, despite the broad success of timed automata [2] in general. An important starting point is the *timed regular expressions* formalism [3]. The basic principle is to interpret input words, hence link streams, as *timed event sequences*: a succession of either symbols or *delays* corresponding to a passage of time. Below is an example of a link stream as a timed event sequence:

$$(a,b)2(d,b)2(a,c)1(a,d)1(c,b).$$

A timed regular expression for the triangle pattern can then be specified, e.g.:

$$\Big(((a \to d) \mid (d \to a)) \cdot \langle (a \to b) \otimes (d \to b) \rangle_{[0,1]}\Big) \otimes (@ \to @)^*$$

The *delay* construction $\langle S \rangle_{[x,y]}$ says that the subpattern S must be detected in time interval $[x, y]$. For the triangle pattern it means that the nodes a and d are only observed as "attacking" target b if they simultaneously link to b in the time interval of one second.

Another fundamental aspect that we intend to capture in link stream patterns is that of *incomplete knowledge*. In classical and timed automata, symbols range over a fixed and finite alphabet. In link streams, this means that the nodes of the graphs must be known in advance, which is in general too strong an assumption. In an attack scenario, for example, we must consider an *open system*: it is very likely that only the target is known in advance, and the two attackers remain undisclosed.

The kind of pattern we intend to support is e.g.:

$$\Big((\sharp X \to \sharp Y) \cdot \langle (X \to b) \otimes (Y \to b) \rangle_{[0,1]}\Big) \otimes (@ \to @)^*.$$

In this pattern, the variables X and Y represent unknown nodes corresponding to two "attackers". The construction $\sharp X$ means that the input symbol (hence node) associated to X must be *fresh*, i.e., not previously encountered. In case of a match this node is associated to X and kept in memory. With the operator $X!$ (the dual of $\sharp X$), after matching a value associated to variable X, all the values associated to it are discarded (i.e. the associated set is cleared).

The sub-pattern $(\sharp X \to \sharp Y)$ describes a link between two fresh nodes. Note that since Y is matched after X, the freshness constraints impose that its associated node is distinct from the one of X. To match the sub-expression $(X \to b)$, the input must be a link from the node already associated with X in memory to node b. This is a potential attack on the target b.

To handle such dynamic matching, one must consider a (countably) infinite alphabet of unknown symbols. This has been studied in the context of *quasi-regular languages* and *finite memory automata* (FMA) [10]. In this paper, we

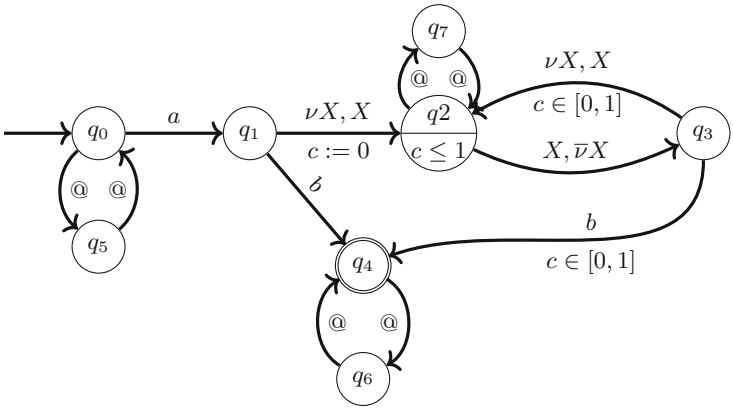


Fig. 2. Automaton for $(@ \to @)^* \otimes \Big((a \to b) \vee a \to \sharp X \cdot \langle (X! \to \sharp X)^* \cdot (X! \to b)\Big)\rangle_{[0,1]}$

build upon the model of ν-automata that we developed in a previous work [6,7]. It is a variant of FMA, which is tailor-made for the problem at hand. If compared to the classical FMA model, the ν-automata can be seen as a generalization to handle *freshness conditions* [13].

The resulting mixed model of *timed ν-automata* is quite capable in terms of expressiveness. The automaton formalism is a combination of both the timed constraint and clocks reset from timed automaton and the memory management of the ν-automaton. As an illustration, Fig. 2 depicts an automaton that detects in a link stream all the paths from a node a to a node b such that each link is established in at most one second. We suppose that the automaton is defined for the alphabet $\Sigma = \{a, b\}$, i.e., only the nodes a and b are initially known. The labels $\nu X, X$ and $X, \overline{\nu} X$ are the automata variants of the operators $\sharp X$ and $X!$ discussed previously. An example of an accepting input is:

$$(a, y)\ 0.1\ (y, z)\ 0.3\ (y, b).$$

Initially, in state q_0 the known symbol a is consumed while transiting to state q_1. The unknown symbol y is saved in the memory associated to variable X while transiting to state q_2. This only works because the symbol y is fresh, i.e., not previously encountered. The delay of 0.1 s is consumed in state q_2 while increasing the value of the clock c to 0.1. The state constraint $c \leq 1$ is still satisfied. The next input y may either lead to q_3 (because it was previously associated to X) or q_7 (because the symbol @ accepts any input). The recognition principle is non-deterministic so both possibilities will be tried:

– if the transition $q_2 \xrightarrow{X, \overline{\nu} X} q_3$ is taken, X is no longer associated to any symbol in q_3. The next input is the unknown symbol z. From q_3, only the transition $q_3 \xrightarrow[c \in [0,1]]{\nu X, X} q_2$ is enabled. In q_2 the variable X would be associated to z.

However, this path is doomed because the next (and last) link does not start from z. Then at the end of the input sequence the path leads to state q_2 which is not a terminal state.

– if the transition $q_2 \xrightarrow{@} q_7$ is taken then the value associated to X is not discarded and the input z leads back to the state q_2 through transition $q_7 \xrightarrow{@} q_2$. The input 0.3 increases the clock value to $c = 0.4$. The next input y may again lead either to q_7 or q_3 as in the previous case. In state q_3 the input b enables only the transition $q_3 \xrightarrow[c\in[0,1]]{b} q_4$, which leads to the final state q_4 (since $b \in \Sigma$).

We reach an accepting state because the clock value $c = 0.4$ is still under 1 s. On the other hand, if the second delay is not 0.3 but e.g., 1.0 then the link stream is not recognized because of a timeout in state q_2.

3 Automata Model and Recognition Principles

The automata model we propose can be seen as a layered architecture with: (1) a classical (non-deterministic) finite-state automata layer, (2) a timed layer (based on [3]) and (3) a memory layer (based on [7]). These layers are obviously dependent but there is a rather clean interface between them.

3.1 The Timed ν-Automata

Definition 1. *A* timed ν-automaton *is a tuple:*

$$A = (\underbrace{\Sigma, Q, q_0, F, \Delta}_{\textit{finite-state}}, \underbrace{C, \Gamma}_{\textit{timed}}, \underbrace{\mathcal{U}, V}_{\textit{memory}})$$

The basic structure is that of a finite-state automaton. We first assume a finite alphabet of *known symbols* denoted by Σ. The finite set Q is that of *locations*[1]. The initial location is q_0 and F is the set of final locations. The component Δ is the set of transitions (explained in details below).

This basic structure is extended for the timed constraints with a set C of *clocks* (ranging over $c_0, c_1, \ldots$) and a map Γ that associates to each location a set of timed constraints. A time constraint is a time interval of the form $[min, max] \in \mathbb{I} = [\mathbb{R}_{\geq 0}, (\mathbb{R}_{\geq 0} \cup \{+\infty\})]$ giving the minimum and maximum values of the clock so that the automaton can "live" in the given location. A transition can also be annotated with time constraints to restrict its firing. Note that the maximum value may be infinite, which means there is no time limit for crossing the transition. The only operations we need on intervals is that of intersection $I_1 \cap I_2$ and membership $c \in I$.

[1] The notion of a location here corresponds to a state in classical automata theory. We rather use the term state in the sense of *actual state* or *configuration* (as in FMAs [10]), i.e., an element of the state-space: a location together with a memory content and clock values.

The memory component is a finite set V of *variables* (ranging over $X, Y, \ldots$) for the memory constraints. Each variable will be associated to a (possibly empty) set of *unknown symbols* ranging over a countably infinite alphabet denoted by $\mathcal{U}$. These symbols are all the symbols that may appear in an input sequence, which are not in Σ. Unlike FMA, which are limited by the number of their registers, the ν-automata use variables of dynamic size, which allows to recognize words composed of an arbitrary number of distinct unknown symbols.

Definition 2. *A transition $t \in \Delta$ of a timed ν-automaton is of the form:*

$$q \xrightarrow[\gamma, \rho]{\nu, e, \overline{\nu}} q'$$

with q (res. q') the starting (resp. ending) location, $\nu \subset V$ a set of variable allocations, $\overline{\nu} \subset V$ a set of variable releases. The *event* e is either a symbol in the finite alphabet Σ, a use of a variable in V or an ε. The timed constraint γ is a guard function of type $C \to \mathbb{I}$, associating to each clock a unique time interval. Finally, ρ is the set of clocks to be reset to 0 while crossing the transition. To simplify the notation of transitions, the empty sets are omitted.

3.2 Dynamics

For a variable $X \in V$, an allocation is a set M_X of unknown symbols, a finite subset of $\mathcal{U}$, together with a flag. The flag may be $M_X^\bullet$ (read mode, default) or M_X° (write mode). In read mode, the only available operation is to check if an input symbol is already present in $M_X^\bullet$. In write mode, only a fresh symbol $\alpha \notin \bigcup_{X \in V} M_X$ may be added. An important property of the memory model is the following.

Definition 3. *A* token *is a pair $k = (k_{time}, k_{mem})$ with k_{time} a function from clocks to clock values, and k_{mem} a mapping from variables to sets of* allocations.

Definition 4. *A* configuration *of an automaton is a mapping S from locations Q to corresponding reachable clocks and memory valuations*[2].

We denote by $S(q)$ the set of tokens associated to location q.

Property 1. INJ (memory injectivity)
For any pair of distinct variables X, Y we have $M_X \cap M_Y = \emptyset$.

We denote by $\mathsf{INJ}(k)$ the fact that token k respects Property 1. Although most memory models do not work like this, this injectivity property is an essential feature of finite-memory automata models (cf. [10]).

The initial configuration of every timed ν-automaton contains the single token $(\{X \to \emptyset^\bullet | \forall X \in V\}, \{c \to 0 | \forall c \in C\})$ in the initial location. The recognition of a pattern in an input sequence in this setting is a non-deterministic

[2] We reuse the token notion of high-level Petri nets because it is quite similar conceptually. The configuration roughly corresponds to the marking of a Petri net.

process. It corresponds to the propagation of tokens over locations of the automaton representing the pattern. The input is accepted if after reading the whole input sequence there is at least one token in some final location. The token itself allows to retrieve the admissible clock values and memory content.

The core of the recognition principle is a partial function δ that takes a token, a transition, and an input symbol (ε if none) to produce either a new token to put into the destination location, or nothing ($\perp$) if the transition is not enabled.

Definition 5. *(update) Consider the transition* $t = q \xrightarrow[\gamma,\rho]{\nu,e,\overline{\nu}} q'$*, a token* $k = (k_{time}, k_{mem})$ *present in* q*, and* α *an input symbol. If we pose* $k'_{time} = \delta_{time}(t, k_{time})$ *and* $k'_{mem} = \delta_{mem}(t, k_{mem}, \alpha)$*, then the next token to put in location* q' *is:*

$$\delta(t, (k_{time}, k_{mem}), \alpha) = \begin{cases} (k'_{time}, k'_{mem}) \text{ if } k'_{time} \neq \perp \text{ and } k'_{mem} \neq \perp \\ \perp \text{ otherwise} \end{cases}$$

The next token only exists if neither the functions for time update δ_{time} or memory update δ_{mem} yield the undefined value $\perp$.

The time update function δ_{time} corresponds to the time model of [3].

Definition 6. *(time update) Let* C *be a set of clocks,* q *a location and* Γ_q *the time constraints function. The* δ_{time} *function is defined as follows:*

$$\delta_{time}(t, k_{time}) = \begin{cases} \perp \text{ if } \exists c \in C \setminus \rho,\ allow(c) \cap \Gamma_{q'}(c) = \emptyset & \text{(case 1.1)} \\ \quad \vee \exists c \in \rho,\ allow(c) = \emptyset \vee 0 \notin \Gamma_{q'}(c) & \text{(case 1.2)} \\ otherwise\ \{c \mapsto k'_c \mid c \in C\} & \text{(case 2)} \\ \quad \text{with } k'_c = \begin{cases} 0 \text{ if } c \in \rho \\ k_{time}(c) \text{ otherwise} \end{cases} & \end{cases}$$

where $allow(c) = k_{time}(c) \cap \gamma(c) \cap \Gamma_q(c)$ *and* γ *is the time constraint on* t*.*

If at least one of the non-reseted clocks c fails to satisfy either the transition guard or the locations constraints (case 1.1) then no token is produced. Another case of failure is if a reseted clock fails to satisfy the initial location constraint and transitions guards, or zero is not accepted in the destination location as the outgoing value of the clock (case 1.2). A token is otherwise produced (case 2), which simply consists in updating the clock to the correct value (either 0 if there is a reset for the clock, or to the value prescribed by the input token).

The principle of updating the memory is a little bit more complex. The memory part of the next token is computed by the memory update function δ_{mem} from the previous memory component depending on an input symbol α. The computation respects the following ordering: the allocation of the variables in set ν is performed before checking the consistency between the input and transition label, and before releasing the variables in the set $\overline{\nu}$.

Definition 7. *(memory update) Let V be a set of variables, and $\mathcal{U}$ an infinite set of unknown symbols. The δ_{mem} function is defined as follows:*

$$\delta_{mem}(t, k_{mem}, \alpha) = \begin{cases} \bot \text{ if } e \notin V \wedge \alpha \neq e & (c.1.1) \\ \quad \vee e \in V \wedge \alpha \notin \mathcal{U} & (c.1.2) \\ \quad \vee e \in V \setminus \nu \wedge k_{mem}(e) = M_e^\bullet \wedge \alpha \notin M_e & (c.1.3) \\ \quad \vee (e \in \nu \vee k_{mem}(e) = M_e^\circ) \wedge \exists Y,\ \alpha \in k_{mem}(Y) & (c.1.4) \\ \text{otherwise } \{X \mapsto k'_X \mid X \in V\} & \\ \qquad \text{with } k'_X = \begin{cases} \emptyset^\bullet, \text{ if } X \in \overline{\nu} & (c.2.1) \\ (M_X \cup \{\alpha\})^\bullet, \text{ if } X = e & (c.2.2) \\ M_X^\circ, \text{ if } X \in \nu & (c.2.3) \\ k_{mem}(X), \text{ otherwise} & (c.2.4) \end{cases} & \end{cases}$$

where $e \in V \cup \Sigma \cup \{\varepsilon\}$ denote the input enabling transition t and the sets ν and $\overline{\nu}$ denote respectively the sets of allocated and freed variables.

In the first four cases no token can be produced. If the transition label e is a known symbol in Σ, then the input α must exactly match else it is a failure (c.1.1). If otherwise e corresponds to a variable, then α must be an unknown symbol in $\mathcal{U}$ (c.1.2). A more subtle failure is (c.1.3) for a variable $e \in V$ in read mode. In this situation the input symbol must be already recorded in the memory associated to e. Complementarily, if the variable e is in write mode (or is put in write mode along the transition), then the input symbol must be *fresh* (c.1.4).

If the next token is produced then for each variable X the associated memory content M_X is updated as follows. If X is to be released (in set $\overline{\nu}$) then the memory is cleared and put in read mode (c.2.1). If it is not released and the variable is to be read (i.e. $X = e$) then α is added to the memory content (c.2.2). In (c.2.3) the variable is not read ($X \neq e$) but it is allocated (in set ν). In this situation the memory content is put in write mode. Otherwise (c.2.4) the memory is left unchanged for variable X.

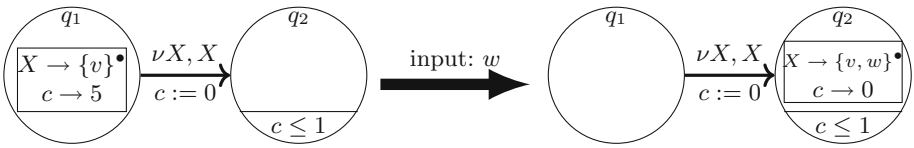


Fig. 3. Passing a transition of the automaton from Fig. 2 with input w

Example 1. Figure 3 illustrates the generation of a new token taking as an example of the transition $d = q_1 \xrightarrow[\rho=\{c\}]{\{X\},X,\{\}} q_2$ from the automaton in Fig. 2. Based on the token $(\{c \to 5\}, \{X \to \{v\}^\bullet\})$ in location q_1, the input w enables the transition d producing a new token in q_2, computed as follows:

– for the time component, the value of clock c verifies the transition constraints. In the arrival location q_2 the clock is reseted and the clock value 0 verifies the time invariant in q_2. The transition would be disabled without the reset because clock value 5 does not satisfy the invariant. Of course, the clock value in the arrival token is $c \rightarrow 0$.
– for the memory component, the transition d is only enabled when the input is an unknown symbol, because transition d is labeled with a variable. Since the alphabet Σ of known symbols is $\{a, b\}$, the symbol w is considered as unknown, i.e., $w \in \mathcal{U}$. Because the allocations are applied before checking the input, the variable X is allocated and then used to enable the transition. So the symbol w should be added to M_X in the newly generated token. However, it is only possible if the input is fresh. Since $M_X = \{v\}$ and X is the only variable, this freshness constraint is satisfied. Hence, the new token associates the memory $\{v, w\}^\bullet$ to X.

As both δ_{time} and δ_{mem} can compute a new value, a new token is generated for the location q_2. ◇

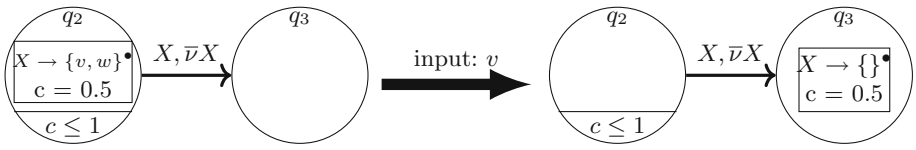


Fig. 4. Passing a transition of the automaton from Fig. 2 with input v

Example 2. Figure 4 presents another example of transition in the automaton of the Fig. 2. This example illustrates a case of memory evolution with δ_{mem}. Here the variable X is used as the trigger and then freed. The variable's freeing occurs simultaneously to reset of the clocks, after checking of guards. As X is not allocated during the transition and was neither allocated before, the transition is enabled only if the input is an unknown symbol and belongs to M_X. The input is actually the unknown symbol $v \notin \Sigma = \{a, b\}$. Furthermore, $v \in \{v, w\} = M_X$, so the transition may be passed and the variable X is cleared in the newly generated token. ◇

The important property of memory injectivity must be preserved through δ_{mem} to fulfill the freshness constraints.

Proposition 1. *(preservation of injectivity) Let k be a token such that $\mathit{INJ}(k)$, and suppose $k' = \delta(t, k, \alpha) \neq \bot$ for some transition t and input α. Then we have $\mathit{INJ}(k')$.*

Proof. In the token $k = (k_{\mathsf{time}}, k_{\mathsf{mem}})$ only the memory component k_{mem} is affected by the injectivity property. The main hypothesis $\mathsf{INJ}(k)$ means that $\forall X, Y\ (X \neq Y),\ k_{\mathsf{mem}}(X) \cap k_{\mathsf{mem}}(Y) = \emptyset$.

Suppose that the transition $t = q \xrightarrow[\gamma,\rho]{\nu,e,\overline{\nu}} q'$ produces token $k' = \delta(t, k, \alpha) = (k'_{\mathsf{time}}, k'_{\mathsf{mem}})$. We have to show $\mathsf{INJ}(k')$, i.e. $k'_{\mathsf{mem}}(X) \cap k'_{\mathsf{mem}}(Y) = \emptyset$ for the same pairs of distinct variables. In the definition of δ_{mem} (Definition 7) we are mostly concerned with cases (c.2.1) to (c.2.4) because we expect a token as output. The memory update depends on the value of the transition trigger e and is as follows:

- If e is not a variable: $e \in \varepsilon \cup \Sigma$, then the case c.2.2 of δ_{mem} cannot occur. So, in the token k' the variable domains are either empty (case c.2.1), or the same as in k (case c.2.3 or c.2.4). Given the hypothesis $\mathsf{INK}(k)$ and the fact that $\emptyset$ is the zero element of intersection, we trivially have $\mathsf{INK}(k')$ as expected.
- If e is a variable: $e \in V$ then the case c.2.2 occurs for exactly one variable of the generated token. As presented above, the variable domains generated with the cases c.2.1, c.2.3 and c.2.4 have empty intersections with each other. Only the domains generated by case c.2.2 must be handled with care. We have to consider two situations:
  - if $\alpha \in k_{mem}(e)$ then the set M_e is not modified, so the Property 1 is trivially verified;
  - or $\alpha \notin k_{mem}(e)$ then case c.1.4 ensures that α is absent in all the domains of the other variables. Thus, Property 1 is verified as well. □

Given a global configuration S and an input α, a new configuration is computed with function σ. It consists in producing all new tokens by the enabled transitions and propagating them through all the ε-transitions.

Definition 8. *(global update)*

$$\sigma(S, \alpha) = \begin{cases} \sigma_{closure}(S, \alpha) \text{ if } \alpha \in \mathbb{R}^+ & \text{(time delay)} \\ \sigma_{closure}(\sigma_{step}(S, \alpha), 0) \text{ otherwise} & \text{(symbol)} \end{cases}$$

The input may be either a known or unknown symbol, or a time delay. In case of a time delay the token should just be propagated through the ε-transitions, defined by $\sigma_{\mathsf{closure}}$ presented below. If the input is a symbol, then new tokens shall be produced through the non ε-transitions, which is expressed by σ_{step}, and then propagated through the ε-closure.

Definition 9. *(event handling)*

$$\sigma_{step}(S, \alpha) = \{q' \mapsto \{\delta(t, k, \alpha) \mid t = q \xrightarrow[\gamma,\rho]{\nu,e,\overline{\nu}} q' \wedge k \in S(q)\} \mid q' \in Q\}$$

The σ_{step} function simply consists in applying the local update function δ at all locations for all non ε-transitions. The tokens belonging to S are not kept in the new configuration.

The resulting tokens are propagated for ε transitions with a time delay fixed to 0. In the case of a time delay, the ε-closure is applied with this fixed delay

as argument. The function $\sigma_{\mathsf{closure}}$ thus handles the time propagation in the automaton. It produces all the possible next tokens in all locations reachable through an ε-path.

Definition 10. *(ε-closure)*

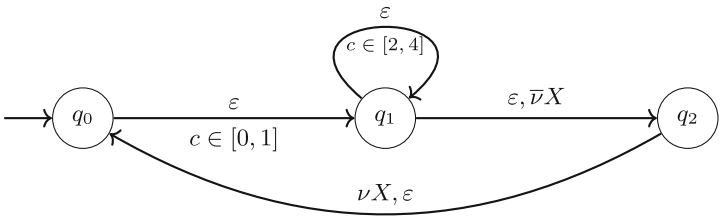
$$\begin{aligned}\sigma_{\mathit{closure}}(S,x) = \{\, q \mapsto \{k \mid\ & \exists q_0 \underbrace{\xrightarrow{\nu_1,\varepsilon,\overline{\nu_1}}}_{\gamma_1,\rho_1} q_1 \underbrace{\xrightarrow{\nu_2,\varepsilon,\overline{\nu_2}}}_{\gamma_2,\rho_2} \cdots q_{n-1} \underbrace{\xrightarrow{\nu_n,\varepsilon,\overline{\nu_n}}}_{\gamma_n,\rho_n} q \in \Delta \\ & \wedge \exists k_0 \in S(q_0),\ \exists x_0 + x_1 + \cdots + x_n = x \\ & \wedge \forall i \in [1,n],\ \exists k_i = \delta(t_i, \mathit{shift}(k_{i-1}, x_{i-1}), \varepsilon) \\ & \wedge k = \mathit{shift}(k_n, x_n), \forall c \in C, k_{\mathit{time}}(c) \in \Gamma_q(c)\} \\ \mid\ & q \in Q\}\end{aligned}$$

where $\mathsf{shift}((k_{\mathsf{time}}, k_{\mathsf{mem}}), x) = (\{c \to k_{\mathsf{time}}(c) + x | c \in C\}, k_{\mathsf{mem}})$.

The idea behind the above is to compute a decomposition of the given delay x as a sum $x_1 + \cdots + x_n$ corresponding to a specific way of waiting in the locations encountered on the ε-path. In fact, only the existence of the decomposition is required. Indeed, for two distinct decompositions the final value retained for a clock c in the next token is the same (either the delay added to the initial value, or 0 for a reseted clock).

Example 3. Figure 5 illustrates the propagation of tokens in an ε-closure, expressed by the function $\sigma_{\mathsf{closure}}$. The token will be propagated with the delay $\alpha = 4$ as an input to exhibit its effect. In this example, we begin with the configuration of step 0 containing only one token in location q_0.

The tokens of this automaton are composed of a variable X and a clock c. The initial configuration, in step 0, contains only one token k_0 in q_0. This token is initialized with X in read mode and a set containing the unknown symbols u and v. There is only one clock c initialized to 0. In step 1, the token k_0 is propagated through the transition $t_{01} = q_0 \xrightarrow[c\in[0,1]]{\varepsilon} q_1$, which generates the token k_1 in location q_1. Since t_{01} has no side-effect (clock or memory update), k_1 is a copy of k_0. In step 2, the token k_1 is propagated through the transition $t_{11} = q_1 \xrightarrow[c\in[2,4]]{\varepsilon} q_1$ generating the token k_1' in location q_1. The transition has no side-effect so the memory of k_1' is the same as the memory of k_1. However, to fulfill the time constraint $c \in [2,4]$, the value of c has to be at least 2. To cross the transition, the clocks values should consume some amount of the input delay α. In step 3, both k_1 and k_1' can be propagated through $t_{12} = q_1 \xrightarrow{\varepsilon,\overline{\nu}X} q_2$. This transition has as a side-effect to clear the variable X. So both the tokens k_2 and k_2' generated respectively from k_1 and k_1' have for variable X the value $\{\}^\bullet$ (an empty set of symbols in read mode). Step 4 consists in the propagation of tokens k_2 and k_2' through the transition $t_{20} = q_2 \xrightarrow{\nu X,\varepsilon} q_0$. This transition has as a side effect to allocate X. However, as t_{20} is an ε-transition, the set associated to X will not be modified and X will be in write mode on the generated tokens. In step 5 two tokens are generated in location q_1, but both come from the token



step	tokens in q_0	tokens in q_1	tokens in q_2
0	$k_0 : (X \to \{u, v\}^\bullet, c \to 0)$		
1	$k_0 : (X \to \{u, v\}^\bullet, c \to 0)$	$k_1 : (X \to \{u, v\}^\bullet, c \to 0)$	
2	$k_0 : (X \to \{u, v\}^\bullet, c \to 0)$	$k_1 : (X \to \{u, v\}^\bullet, c \to 0)$ $k_1' : (X \to \{u, v\}^\bullet, c \to 2)$	
3	$k_0 : (X \to \{u, v\}^\bullet, c \to 0)$	$k_1 : (X \to \{u, v\}^\bullet, c \to 0)$ $k_1' : (X \to \{u, v\}^\bullet, c \to 2)$	$k_2 : (X \to \{\}^\bullet, c \to 0)$ $k_2' : (X \to \{\}^\bullet, c \to 2)$
4	$k_0 : (X \to \{u, v\}^\bullet, c \to 0)$ $k_0' : (X \to \{\}^\circ, c \to 0)$ $k_0'' : (X \to \{\}^\circ, c \to 2)$	$k_1 : (X \to \{u, v\}^\bullet, c \to 0)$ $k_1' : (X \to \{u, v\}^\bullet, c \to 2)$	$k_2 : (X \to \{\}^\bullet, c \to 0)$ $k_2' : (X \to \{\}^\bullet, c \to 2)$
5	$k_0 : (X \to \{u, v\}^\bullet, c \to 0)$ $k_0' : (X \to \{\}^\circ, c \to 0)$ $k_0'' : (X \to \{\}^\circ, c \to 2)$	$k_1 : (X \to \{u, v\}^\bullet, c \to 0)$ $k_1' : (X \to \{u, v\}^\bullet, c \to 2)$ $k_1'' : (X \to \{\}^\circ, c \to 0)$ $k_1''' : (X \to \{\}^\circ, c \to 2)$	$k_2 : (X \to \{\}^\bullet, c \to 0)$ $k_2' : (X \to \{\}^\bullet, c \to 2)$
6	$k_0 : (X \to \{u, v\}^\bullet, c \to 4)$ $k_0' : (X \to \{\}^\circ, c \to 4)$	$k_1 : (X \to \{u, v\}^\bullet, c \to 4)$ $k_1'' : (X \to \{\}^\circ, c \to 4)$	$k_2 : (X \to \{\}^\bullet, c \to 4)$

Fig. 5. Example of ε-closure with an input delay 4

k_0'. As k_0'' has $c \to 2$, it cannot enable t_{01} because the clock constraint $c \in [0, 1]$ is not respected. The token with $c \to 0$ crosses t_{01} and the transition t_{11} (as in step 2) generating two tokens, k_1'' and k_1''', in q_1 with different clock values. After step 5 it is not possible to generate any new token in a location with a different value than the tokens already present in it. In step 6 the propagation is over and all the clocks are increased to $k_{0_{time}}(c) + \alpha = 0 + 4$.

However, only one token is kept at a location if several are generated with identical clock and memory valuations. The step 6 corresponds to the configuration returned by $\sigma_{\mathsf{closure}}$. ◇

Since there may be an infinite number of ε-paths from a given starting location q, the following is an important Property wrt. decidability.

Proposition 2. *For a given configuration S and time delay x, the function $\sigma_{\mathsf{closure}}$ can only produce a finite amount of tokens.*

Proof. To prove the proposition, we show that both the possible memory and clocks states are finite over the propagation through the ε-closure.

First, we prove that the number of memory states is finite. An ε-transition does not read any symbol. So, the only memory operations present in an ε-closure are the allocation ν and the freeing $\overline{\nu}$. Let X be a variable of initial valuation M_X^a, where M_X is the set associated to X and a the initial mode of X. Its reachable values in the ε-closure are:

- M_X^a in all ε-paths with no operations on X,
- M_X° in all ε-paths where X is only allocated,
- $\emptyset^\bullet$ in all ε-paths where the last memory operation used on X is a freeing $\overline{\nu}$,
- $\emptyset^\circ$ in all ε-paths where X was freed at least once and the last memory operation on X is an allocation ν.

As a consequence, if the tokens are composed of n variables, after the propagation in an ε-closure at most 4^n variations of each initial memory valuation can be generated.

To prove that the number of clock valuations is finite we recall that our time model is based on *timed pattern matching* [3], which implies that clock resets can only be on non ε-transitions. Thus, the final clock values after the propagation over an ε-closure are the initial values increased by a possible delay given as an input. The number of clock valuations is constant through the propagation.

As the number of memory states and clocks states are both finite, the number of combinations between them is finite too. An upper bound for the number of tokens generated in an ε-closure is $nb_{\text{tokens}} \cdot 4^{nb_{\text{vars}}} \cdot nb_{\text{states}}$, where nb_{tokens} is the number of tokens composing the initial configuration (as an upper bound for the number of memory valuations), nb_{vars} is the number of variables composing a token, and nb_{states} is the number of states composing the ε-closure. □

3.3 Pattern Language

The description of non-trivial patterns in link streams can become tedious if specified directly as automata. Indeed, even simple patterns can yield very large automata. We are looking for a more concise way to describe the patterns, in the spirit of regular expressions. We propose the language of *timed ν-expressions* to specify patterns for link streams.

The syntax of the core constructs is given in Table 1. The basic constructs are those of traditional regular expressions. The symbols are referring to known, unknown or arbitrary nodes. The link construct $n_1 \to n_2$ symbolizes a *non-breaking* connection between two nodes. The delay construct for time constraints is the same as in [3]. The constructs for memory management are based on variable occurrences (for unknown nodes), allocations and releases. The notation $\sharp\{X_1, \ldots, X_n\}e$ (resp. $e\{X_1, \ldots, X_n\}!$) means that the variables $X_1, \ldots, X_n$ are allocated (resp. released) before (resp. after) recognizing the subexpression e. The shuffle operator $\otimes$ is present in the language to ease the description of patterns with independent parts.

The semantics of the pattern language is given in terms of a generated timed ν-automaton. By lack of space, we do not describe the translation formally in

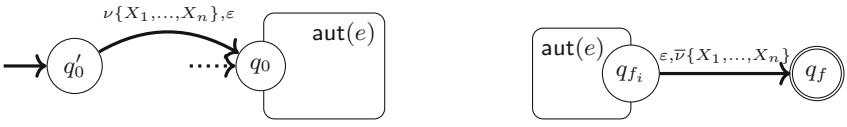
Table 1. The (core) pattern language

Node	$n, n_1, n_2 \ldots ::=$	k	(known node)
		X	(variable, unknown node)
		@	(arbitrary node)
Expression	$e, e_1, e_2, \ldots ::=$	n	(node)
		$n_1 \rightarrow n_2$	(link)
(regular)		$e_1 \,.\, e_2$	(concatenation)
		$e_1 \mid e_2$	(disjunction)
		$e_1 \otimes e_2$	(shuffle)
		e^*	(iteration)
(time)		$\langle e \rangle_{[x,y]}$	(delay[a])
(memory)		$\sharp\{X_1, \ldots, X_n\}e$	(allocation)
		$e\{X_1, \ldots, X_n\}!$	(release)

[a]Following [3] the expression inside a delay should not be empty.

this paper but only describe it informally. We intend to investigate the formal properties of the language (e.g. language equivalence) in a future work. Note that the translation is relatively straightforward. The translation rules for the regular expression constructs are the classical ones. A special case is the link expression $n_1 \rightarrow n_2$ that corresponds to a basic automaton with three states and two transitions, one for n_1 and the second for n_2. One important property is that this construction is non-breaking (e.g. it is atomic for the shuffle). For the delay construct a thorough explanation is given in [3]. It mostly remains to explain the translation of the memory operations.

Table 2. Automata for memory operators



aut($\sharp\{X_1, \ldots, X_n\}e$) aut($e\{X_1, \ldots, X_n\}!$)

Table 2 illustrates how the allocation and release operators are translated:

- The function aut: *expression* $\rightarrow$ *automaton* translates a timed ν-expression to the corresponding timed ν-automaton.
- The translation of $\sharp\{X_1, \ldots, X_n\}e$ gives rise to a new initial location q_0' and a ε-transition between q_0' and the initial location of the automaton generated from e, which allocates the variables $X_1, \ldots, X_n$. The new initial location of the automaton is q_0'. The translation of $e\{X_1, \ldots, X_n\}!$ leads to the creation of a new final location q_f and a new transition from each final location of

the automaton generated for e to q_f, each of them releasing the variables $X_1, \ldots, X_n$. The new unique final location is q_f.

In the experiments we often used the following derived constructs:

- allocation and use: $\sharp X \stackrel{\text{def}}{=} \sharp\{X\}X$
- use and release: $X! \stackrel{\text{def}}{=} X\{X\}!$
- allocation, use and release $\sharp X! \stackrel{\text{def}}{=} \sharp\{X\}X\{X\}!$

The whole translation has been implemented in a prototype tool described in the next section.

4 Experiments

Our main objective is to develop a practical pattern matching tool for link stream analysis. An early implementation of the tool is available online[3]. In this section we present early experiments with this prototype to real-world link streams.

For starters, the worst-case complexity of our pattern matching algorithm is exponential on the size of the link stream (the number of links). This complexity is reached for instance in the case depicted in Fig. 6, which is a "memory-only" scenario. If the input is a sequence of distinct symbols then the number of tokens associated to the unique location of the automaton will double each time a symbol is consumed. For instance, in the figure, the 8 tokens are associated to distinct versions of the variable U (the U_i's) after consuming the input $a\ b\ c$: one for each subset of the alphabet.

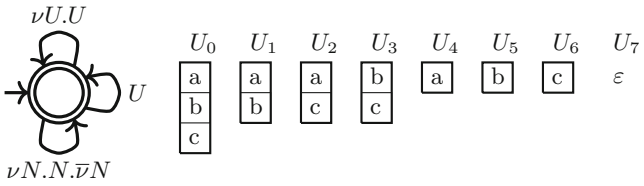


Fig. 6. A subset automaton after input $a\ b\ c$.

However, timed constraints most often improve the situation by removing expired tokens. Thus, in practice there are ways to avoid the worst-case scenarios. This is similar to the practical "regex" tools, which in general go well beyond regular expressions, also leading to exponential blowups in the worst case [4,5].

This makes experimental evaluation of our method particularly appealing to estimate its practical performances and applicability. In order to do so, we consider two link streams built from two different real-world datasets: (1) a recording of traffic routed by a large internet trans-Pacific router [8], and (2) a one month capture of tweets on Twitter France.

[3] The MaTiNa tool repository is at: https://github.com/clementber/MaTiNA.

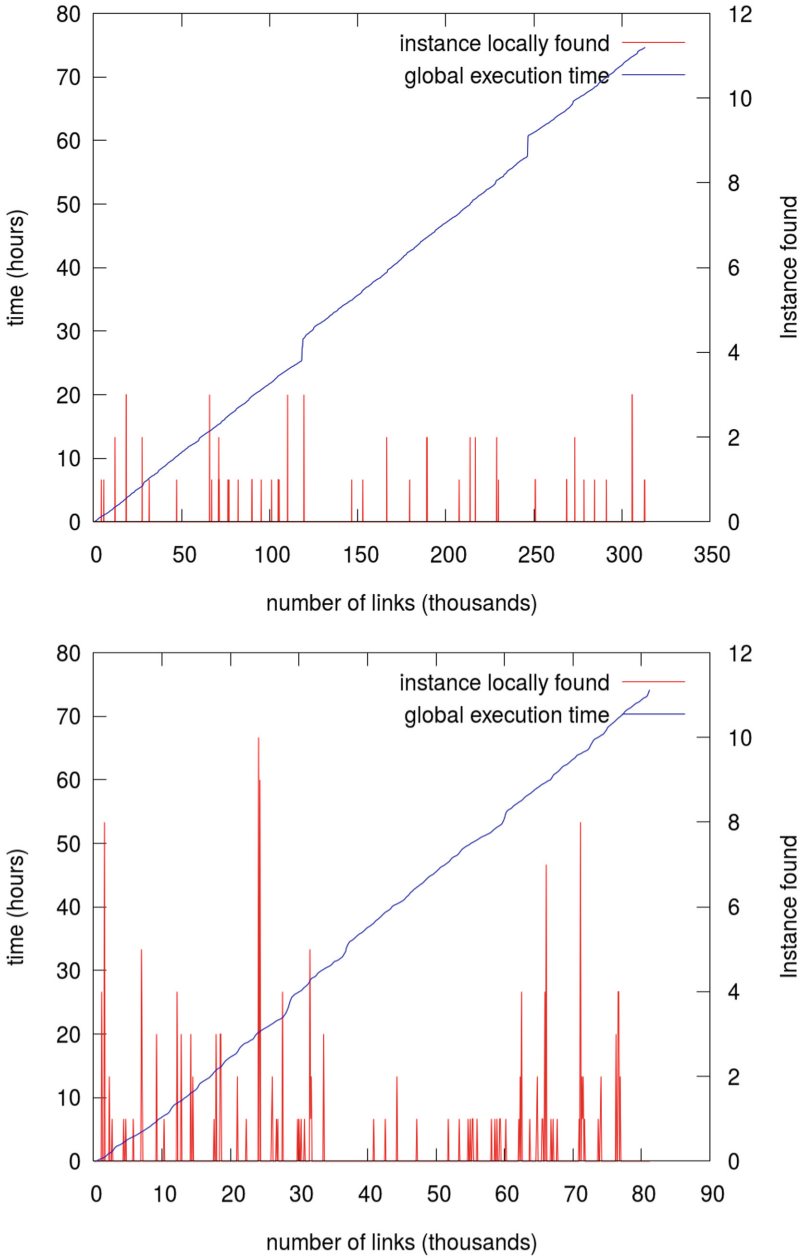


Fig. 7. DDoS pattern recognition with time frames $\delta = 0.01$ (top) and $\delta = 0.02$ (bottom).

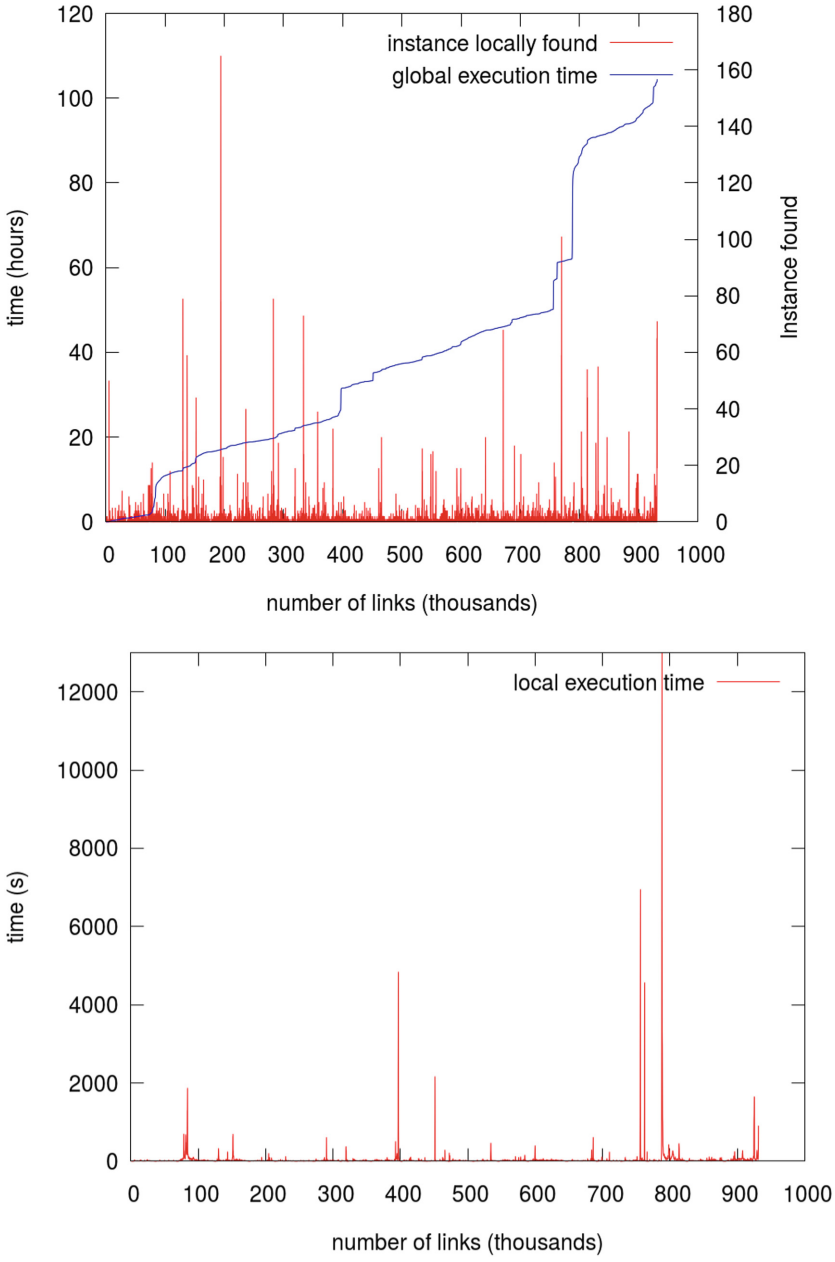


Fig. 8. Triangle detection in Twitter exchanges with running time and number of detected instances (top) and local running time (bottom).

In the case of internet traffic, our motivation is to detect potential coordinated attacks. To do so, we define a variant of the triangle pattern discussed in Sect. 2, namely 2×2 bicliques, i.e. squares, which [16] identified as meaningful

to this regard. Since there is approximately one link every 2 µs in the stream and the stream lasts for a whole day, it must be clear that we may not detect all untimed patterns in the stream. In this context, the time frame of an attack is in general quite sudden and precise, and so time is a crucial feature.

We present results for two different time frames in Fig. 7. It displays the total running time as a function of the number of processed links, together with the number of found instances of the pattern. As expected, the number of instances of the pattern increases with the time frame. Also, the tool processes less links in a given amount of time (85 h in this experiment). Although our implementation is not optimized at all, the linear time cost of the computations w.r.t. the number of processed links clearly appears.

Our second experiment targets communities of Twitter users. We consider tweets over a period of a month, leading to a stream of 1.3 million links[4]. The pattern we seek is an undirected complete graph between k users for a given k, i.e. cliques of size k occurring in a time frame of ten minutes. Figure 8 presents the results for $k = 3$, i.e. triangle detection. The running time experiences sharp increases at specific times, that correspond to peak periods in Twitter exchanges. This is confirmed by plotting the execution time at each step of the computation (right part of the figure). During such peaks of tweets, the tool has to store more data than usual, leading to a more costly processing of links. One way to improve this issue would be to consider a variable time rate by e.g. decomposing the link stream in distinct sub-streams processed with different time frame.

5 Conclusion

The language of *timed ν-expressions* we propose to specify patterns in link streams is heavily inspired by regular expressions, but enriched with timed and memory features. The language is rather low-level but with well-chosen derived constructs we think it is usable (and has been used) by domain experts. The language has a straightforward translation to the core outcome of our research: the *timed ν-automata* formalism and the corresponding recognition principles. Beyond the formalities, we developed a functional, and freely available, prototype that we experimented in a realistic setting. Non-trivial patterns have been detected on real-world link streams, with decent performances for such an early prototype. These early experiments give us confidence regarding the relevance of our approach.

For future work, we plan both theoretical investigations and more practical work at the algorithmic and implementation level. We also expect to broaden the application domains. In particular, since our detection is performed *online*, one potential area of application is that of monitoring open systems at runtime for e.g. security or safety properties. At the theoretical level, we plan to study the pattern language and its more precise relation to the automata framework. Since the semantics are based on a token game, the formalism is in a way closed

[4] The data come from the Politoscope project by the CNRS Institut des Systémes Complexes Paris Ile-de-France (https://politoscope.org).

to the Petri nets than it is from classical automata. Hence, interesting extensions of the formalism could be developed based on a high-level Petri net formalism, e.g. in the spirit of [9]. Our prototype tool uses a relatively naive interpreter for pattern matching. We plan to improve its performances by first introducing a compilation step. Moreover, there is an important potential for parallelization of the underlying token game.

References

1. Agrawal, J., Diao, Y., Gyllstrom, D., Immerman, N.: Efficient pattern matching over event streams. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 147–160 (2008)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
3. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. J. ACM **49**(2), 172–206 (2002)
4. Câmpeanu, C., Salomaa, K., Sheng, Y.: A formal study of practical regular expressions. Int. J. Found. Comput. Sci. **14**(6), 1007–1018 (2003)
5. Carle, B., Narendran, P.: On extended regular expressions. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 279–289. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00982-2_24
6. Deharbe, A., Peschanski, F.: The omniscient garbage collector: a resource analysis framework. In: ACSD 2014. IEEE Computer Society (2014)
7. Deharbe, A., Peschanski, F.: The omniscient garbage collector: a resource analysis framework. Research report, LIP6 UPMC Sorbonne Universités, France (2014). https://hal.archives-ouvertes.fr/hal-01626770
8. Fontugne, R., Borgnat, P., Abry, P., Fukuda, K.: MAWILab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In: ACM CoNEXT 2010 (2010)
9. Garg, V.K., Ragunath, M.T.: Concurrent regular expressions and their relationship to petri nets. Theor. Comput. Sci. **96**(2), 285–304 (1992)
10. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. **134**, 329–363 (1994)
11. Latapy, M., Viard, T., Magnien, C.: Stream graphs and link streams for the modeling of interactions over time. CoRR, abs/1710.04073 (2017)
12. Paranjape, A., Benson, A.R., Leskovec, J.: Motifs in temporal networks. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, pp. 601–610. ACM (2017)
13. Tzevelekos, N.: Fresh-register automata. In: Proceedings of the 38th Annual ACM SIGPLAN-SIGACT, POPL 2011, pp. 295–306. ACM (2011)
14. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Timed pattern matching. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 222–236. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10512-3_16
15. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Online timed pattern matching using derivatives. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 736–751. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_47
16. Viard, T., Fournier-S'niehotta, R., Magnien, C., Latapy, M.: Discovering patterns of interest in IP traffic using cliques in bipartite link streams. CoRR, abs/1710.07107 (2017)

17. Waga, M., Akazaki, T., Hasuo, I.: A Boyer-Moore type algorithm for timed pattern matching. In: Fränzle, M., Markey, N. (eds.) FORMATS 2016. LNCS, vol. 9884, pp. 121–139. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44878-7_8
18. Waga, M., Hasuo, I., Suenaga, K.: Efficient online timed pattern matching by automata-based skipping. In: Abate, A., Geeraerts, G. (eds.) FORMATS 2017. LNCS, vol. 10419, pp. 224–243. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65765-3_13
19. Zhang, H., Diao, Y., Immerman, N.: On complexity and optimization of expensive queries in complex event processing (2014)

Semantics and Expressiveness

Modeling Operational Semantics with Interval Orders Represented by Sequences of Antichains

Ryszard Janicki(✉)

Department of Computing and Software, McMaster University,
Hamilton, ON L8S 4K1, Canada
janicki@mcmaster.ca

Abstract. A representation of interval orders by sequences of antichains is discussed and its relationship to the Fishburn's representation by sequences of beginnings and endings is analyzed in detail. Using sequences of antichains to model operational semantics of elementary inhibitor nets is also discussed.

Keywords: Interval orders · Operational semantics
Elementary inhibitor nets · Sequences of antichains · Interval sequences

1 Introduction

Most operational[1] semantics of concurrent systems are defined either in terms of sequences (i.e. total orders) or step-sequences (i.e. stratified orders) [3,11,15,19]. It was argued by Wiener in [20] (and later more formally in [11]) that any execution that can be observed by a single observer must be an interval order. It implies that the most precise operational semantics is defined in terms of interval orders. However generating interval orders directly is problematic for most models of concurrency, as interval orders do *not* have a natural sequence representation, as both total and stratified orders have (plain sequences and step sequences respectively). For interval orders one might use either sequences of *beginnings* and *endings* of events involved (Fishburn Theorem [6]), or *sequences of appropriate antichains* (i.e. a kind of step sequences, but interpreted differently) [7,11]. The former approach lead to the ideas of *ST-traces* [18,19] and *interval sequences* [13], the latter is the subject of this paper. The problem with all approaches based on *beginnings* and *endings* is that sequence representations of interval orders given by Fishburn Theorem are not unique, we often have to include all representations, which results in relatively complex and cluttered models for even not so complex cases.

Partially supported by NSERC grant of Canada.

[1] 'Operational semantics' is not a generally agreed concept, in this paper this will be just a collection of all system runs (i.e. executions, observations) [3,11,15,19]. A different meaning is used in for example [4].

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 251–271, 2018.
https://doi.org/10.1007/978-3-319-91268-4_13

In this paper we will show a detailed relationship between interval sequences of [13] and sequences of antichains including simple algorithms that transform one into another. Based on this relationship we will use sequences of antichains to represent operational semantics of elementary nets with inhibitor arcs. This will provide an alternative operational semantics in terms of interval orders for these nets. We will show that this new semantics is consistent with the interval sequence semantics of [13].

2 Partial Orders and Sequences

We will now present some results and notations about partial orders, sequences and their mutual relationship that will be used in the rest of this paper.

A relation $<\subseteq X\times X$ is a (*strict*) *partial order* iff it is irreflexive and transitive, i.e. for all $a,c,b\in X$, $a\not< a$ and $a<b<c \implies a<c$. We also define:

$$a \frown_< b \stackrel{df}{\iff} \neg(a<b)\wedge\neg(b<a)\wedge a\neq b,$$

Note that $a\frown_< b$ means a and b are *incomparable* (w.r.t. $<$) elements of X.

In this paper we will deal mainly with interval and total orders, and often mention stratified orders.

Let $<$ be a partial order on a set X. Then:

1. $<$ is *total* if $\frown_<=\emptyset$. In other words, for all $a,b\in X$, $a<b \vee b<a \vee a=b$. For clarity, we will reserve the symbol $\lhd$ to denote total orders;
2. $<$ is *stratified* if $a \frown_< b \frown_< c \implies a\frown_< c \vee a=c$, i.e., the relation $\frown_<\cup id_X$ is an equivalence relation on X;
3. $<$ is *interval* if for all $a,b,c,d\in X$, $a<c \wedge b<d \implies a<d\vee b<c$.

It is clear from these definitions that every total order is stratified and every stratified order is interval. In this paper, most partial orders will be represented by Hasse diagrams [7]. Figure 1 illustrates the above definition. The orders $\lhd_1, \overset{po}{<_1}$ are total, the orders $<_3$ and $\overset{po}{<_1}$ are stratified, the order $<_1$ is interval and the order $<_2$ is not an interval order.

Finite total orders can uniquely be represented sequences, for example $\lhd_1$ from Fig. 1 is represented by the sequence $BaEaBbBcEbBdEcEd$, $\overset{po}{<_1}$ from the same figure by the sequence (of sets) $\{a\}\{b,c\}\{c,d\}$ (cf. [8,12]).

Finite stratified orders can uniquely be represented by step sequences. For example $<_3$ from Fig. 1 is represented by the step sequence $\{a,b\}\{c,d\}$ and $\overset{po}{<_2}$ from the same figure by the step sequence $\{A_1\}\{A_2,A_3\}\{A_4\}$, where $A_1=\{a,b\}$, $A_2=\{a,d\}$, $A_3=\{b,c\}$ and $A_4=\{c,d\}$.

The partial order $<_2$ of Fig. 1 is the simplest example of an order that is not interval order (cf. [7]).

For the interval orders, the name and intuition follow from Fishburn's Theorem:

Theorem 1 (Fishburn [6]). *A partial order $<$ on countable[2] set X is interval iff there exists a total order $\lhd$ on some T and two injective mappings with disjoint codomains $B, E : X \to T$ such that for all $x, y \in X$,*

1. $B(x) \lhd E(x)$,
2. $x < y \iff E(x) \lhd B(y)$. ◇

Usually $B(x)$ is interpreted as the beginning and $E(x)$ as the end of an *interval* x. To shorten our notations, we will omit parentheses and write Bx and Ex to denote $B(x)$ and $E(x)$ respectively. The intuition of Fishburn's theorem is illustrated in Fig. 1 with $<_1$ and $\lhd_1$. For all $x, y \in \{a, b, c, d\}$, we have $Bx \lhd_1 Ex$ and $x <_1 y \iff Ex \lhd_1 By$.

Each sequence (step sequence) of events represents a total (stratified) order of *enumerated events* in a natural way. For precise definitions see for example [8,12,15], here we will be using the following notation.

Notation

1. For each set of events Σ, let $\widehat{\Sigma} = \{a^i \mid a \in \Sigma, i \geq 1\}$ denote the set of *enumerated* events generated by Σ.
2. For each sequence $x \in \Sigma^*$ and each step sequence $z \in (2^\Sigma)^*$, let $\hat{x} \in \widehat{\Sigma}^*$ and $\hat{z} \in (2^{\widehat{\Sigma}})^*$ denote their *enumerated representations*.
   For example, if $x = abbaa$ then $\hat{x} = a^1b^1b^2a^2a^3$, and if $z = \{a, b\}\{a, b, c\}\{a\}$ then $\hat{z} = \{a^1, b^1\}\{a^2, b^2, c^1\}\{a^3\}$.
3. For every sequence $x \in \Sigma^*$, $\lhd_x$ is the *total order* defined by the enumerated sequence $\hat{x}$.
   For example: $\lhd_{abbaa} = a^1 \to b^1 \to b^2 \to a^2 \to a^3$.
4. For every step sequence $z \in (2^\Sigma)^*$, $\lhd_z$ is the *stratified order* defined by the enumerated step sequence $\hat{z}$.
   For example: $\lhd_{\{a,b\}\{a,b,c\}\{a\}} = \{a^1, b^1\} \to \{a^2, b^2, c^1\} \to \{a^3\}$.
5. For every sequence or step sequence x, let $\mathscr{A}_x$ denotes the set of all elements of Σ occurring in x.
   For example $\mathscr{A}_{abbaac} = \{a, b, c\}$, $\mathscr{A}_{\{a,b\}\{a,b,c\}\{a\}} = \{a, b, c\}$.
6. For every sequence or step sequence x and every $a \in \Sigma$, we will write $a \in x$ if $a \in \mathscr{A}_x$.
   For example $b \in abbac$, $b \in \{a, b\}\{b, c\}$.
7. For $x, y \in \Sigma^*$ we will write $x \subseteq y$ iff $y = x_1 x x_2$ for some $x_1, x_2 \in \Sigma^*$. If $x \subseteq y$, we will say that x is a *subsequence* of y.
   For example $bbc \subseteq abbcb$.
8. For every sequence or step sequence x and every $A \subseteq \Sigma$, $x \cap A$ denote the projection of x onto A, i.e. a sequence or step-sequence derived from x by erasing all elements of $\Sigma \setminus A$ (cf. *erasing homomorphism* [9]).
   For example if $x = abbcbdda$, $y = \{a, b, d\}\{a, b, c, e\}\{a, c\}$ and $A = \{a, b\}$, then $x \cap A = abbba$, $y \cap A = \{a, b\}\{a, b\}\{a\}$. ◇

We will often write a instead of a^1 if this will not lead to any confusion.

[2] For uncountable X it is additionally required that the equivalence relation $\sim_<$ defined as $a \sim_< b \iff \forall c \in X.(c < a \Leftrightarrow c < b) \wedge (a < c \Leftrightarrow b < c)$ has countably many equivalence classes [6]. But in this paper we need only a simpler version for countable X, cf. [11].

3 Interval Sequences

In this section we recall the concept of interval sequences as presented in [13]. Conceptually the interval sequences are close to older ST-traces of [18], the difference is that ST-traces have been defined in the framework of Petri nets, while interval sequences do not assume any model of a system.

For every finite set (of events) Σ, the set

$$\mathscr{BE}_\Sigma = \{Ba \mid a \in \Sigma\} \cup \{Ea \mid a \in \Sigma\},$$

is the set of all beginnings and endings of events in Σ.

Many sequences from $\mathscr{BE}^*_\Sigma$ represent interval orders, for example $\lhd_{BaBbEaEb} = \lhd_{BbBaEaEb} = \lhd_{BaBbEbEa} = \lhd_{BbBaEbEa}$ is an interval order $a \frown_< b$, but not every sequence from $\mathscr{BE}^*_\Sigma$ can be interpreted as an interval order, for example $BaBcBb$ represents no interval order.

- We say that a string $x \in \mathscr{BE}^*_\Sigma$ is an *interval sequence* iff

$$\forall Ba, Ea \in x.\ x \cap \{Ba, Ea\} \in (BaEa)^+.$$

- We will use $\mathsf{InSeq}(\Sigma)$ to denote the set of all interval sequences of $\mathscr{BE}^*_\Sigma$.

For example for $\Sigma = \{a, b, c\}$ a sequence $x_1 = BaBbEbEaBcBaBbEcEbEaBaEa$ is in $\mathsf{InSeq}(\Sigma)$, as $x_1 \cap \{Ba, Ea\} = BaEaBaEaBaEa$, $x_1 \cap \{Bb, Eb\} = BbEbBbEb$ and $x_1 \cap \{Bc, Ec\} = BcEc$; but sequences $x_2 = EaBbEbBa$ or $x_3 = BbEbBaEc$ are not in $\mathsf{InSeq}(\Sigma)$, since $x_2 \cap \{Ba, Ea\} = EaBa$ and $x_3 \cap \{Ba, Ea\} = Ba$.

Definition 1 **([13]).** Let $x \in \mathsf{InSeq}(\Sigma)$, and let $\blacktriangleleft_x$ be a relation on $\widehat{\Sigma}$, defined by:

$$a^i \blacktriangleleft_x b^j \iff Ea^i \lhd_x Bb^j.$$

By Theorem 1, the relation $\blacktriangleleft_x$ is an interval order, and it is called the *interval order generated by the sequence x of beginnings and ends*. ◇

For example if $x = BaEaBbBcEbBdEcEd$ then $\blacktriangleleft_x$ is the interval order $<_1$ from Fig. 1.

4 Sequences of Antichains

In this section we will show how interval orders can uniquely be represented by appropriate sequences of antichains.

Definition 2 **([7,11]).** Let $<$ be a partial order on X (of any kind).

1. A set $A \subseteq X$ is a *maximal antichain* of $<$ if and only if

$$(\forall a, b \in A.\ a \frown_< b \vee a = b) \wedge (\forall a \notin A. \exists b \in A.\ a < b \vee b < a).$$

The set of all maximal antichains of $<$ will be denoted by $\mathbb{A}_<$.

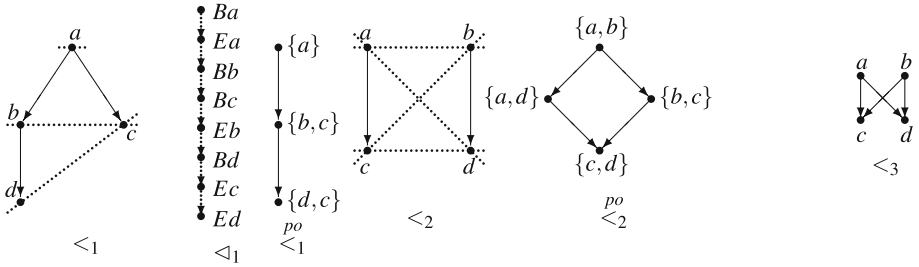


Fig. 1. Illustrations of partial orders definitions and Theorems 1 and 2. The order $<_1$ is interval, while $<_2$ is not. Antichains are connected with dotted lines. The total order $\lhd_1$ is a Fishburn's representation of the interval order $<_1$. A stratified order $<_3$ is represented by a step sequence $\{a, b\}\{c, d\}$.

2. The relation $\overset{po}{<} \subseteq \mathbb{A}_< \times \mathbb{A}_<$, defined as

$$A \overset{po}{<} B \iff A \neq B \wedge (\forall a \in A \setminus B. \forall b \in B \setminus A.\ a < b)$$

is called a *principal order* of $<$ (see [11] for more details). ◇

For the partial orders $<_1$ and $<_2$ in Fig. 1, $\overset{po}{<_1}$ and $\overset{po}{<_2}$ are their principal orders respectively. Both $\overset{po}{<_1}$ and $\overset{po}{<_2}$ are partial orders. It turns out this property holds in general, principal orders are always partial orders of maximal antichains. Moreover we can always recover the partial order $<$ from its principal order $\overset{po}{<}$.

Proposition 1 ([11]). *Let $<$ be any partial order on X.*

1. *The relation $\overset{po}{<}$ is a partial order.*
2. $\forall a, b \in X.\ a < b \iff a \neq b \wedge (\forall A, B \in \mathbb{A}_<.\ a \in A \wedge b \in B \implies A \overset{po}{<} B)$. ◇

Maximal antichains and principal orders are also convenient tools for classifying partial orders.

Theorem 2 ([7,11]). *A partial order $<$ is an interval order if and only if its principal order $\overset{po}{<}$ is a total order (of maximal antichains).* ◇

Theorem 2 is illustrated in Fig. 1. The order $<_1$ is interval and the principal order $\overset{po}{<_1}$ is total while the order $<_2$ is not interval and the principal order $\overset{po}{<_2}$ is not total. As a simple consequence of Theorem 2 we have the following corollary.

Corollary 1. *A partial order $<$ is stratified if and only if all maximal antichains are equivalence classes of $\frown_<$.* ◇

- When $\overset{po}{<}$ is a total order, it can be represented as an appropriate sequence of antichains of $<$. We may identify this sequence representation with the total order $\overset{po}{<}$ and write $\overset{po}{<} = A_1 \ldots A_n$, if convenient.

Note that A_i's are different antichains, i.e. $A_i \subseteq A_j$ if $i = j$, hence we have $A_i \overset{po}{<} A_j$ iff i is smaller than j.

- For every sequence of antichains $s = A_1 \dots A_n$, the *interval order* that the sequence s represents will be denoted by $\angle_s$.

For example $\angle_{\{a\}\{b,c\}\{c,d\}}$ equals $<_1$ of Fig. 1. The notation $\angle_s$ does make sense only if s is a sequence of antichains, it cannot be applied to just arbitrary step sequence s.

Not every step sequence cannot be interpreted as a sequence of antichains. For example neither $\{a,b\}\{a,b\}$ nor $\{a,c\}\{a,b\}\{b,c\}$ can be interpreted as sequences of antichains, for different reasons. Each antichain is different, but the step sequence $\{a,b\}\{a,b\}$ contains two identical steps. While different antichains may have common elements, common elements may belong only to consecutive antichains. In the sequence $\{a,c\}\{a,b\}\{b,c\}$, the element c belongs to two steps that are separated by another step.

Definition 3. A step sequence $A_1 \dots A_k$ of elements of X is a *sequence of antichains* if and only if the following conditions are satisfied:

1. For all A_i, A_j, $A_i \subseteq A_j \iff i = j$.
2. If $a \in A_i$ and $a \notin A_{i-1}$ then $a \notin A_j$ for all $j < i - 1$.
3. If $a \in A_i$ and $a \notin A_{i+1}$ then $a \notin A_j$ for all $j > i + 1$. ◇

The condition (1) is a simple consequence of the fact that A_i's are maximal antichains. The conditions (2) and (3) say that any element of Σ may occur only is a strictly consecutive subsequence of the sequence $A_1 \dots A_k$, i.e. we have the pattern $\underbrace{A_1 \dots A_{r-1}}_{a \notin A_i} \underbrace{A_r \dots A_s}_{a \in A_i} \underbrace{A_{s+1} \dots A_k}_{a \notin A_i}$.

- For every finite set X, let $\mathbb{SA}(X)$ denote the set of all sequences of antichains that can be built from the elements of X.

For $X = \widehat{\Sigma}$ we additionally assume that if $A_1 \dots A_k \in \mathbb{SA}(\widehat{\Sigma})$ then for each $a \in \Sigma$:

1. $a^i \in A_s, a^j \in A_t$ and $s < t$ implies $i \leq j$,
2. $a^{i+1} \in A_s$ implies $a^i \in A_t$ for some $t < s$.

From Definitions 2 and 3, and Proposition 1, we have the following result.

Proposition 2. *1. If $s = A_1 \dots A_k$ is a sequence of antichains then there is the unique interval order $<_s$ such that $\overset{po}{<_s} = A_1 \dots A_k$.*
2. For every sequence of antichains s, we have $s = \overset{po}{\angle_s}$. ◇

For interval orders *simultaneity* is usually described by overlapping, i.e. a and b are considered simultaneous if Bb occurs before Ea, or Ba occurs before Eb (cf. [2]). Simultaneity is not transitive (as opposed to stratified orders) and maximal antichains represent maximal sets of simultaneous events.

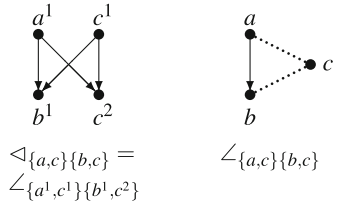


Fig. 2. Partial orders generated by a sequence of sets $\{a,b\}\{b,c\}$, once interpreted as a step sequence -the order $\lhd_{\{a,c\}\{b,c\}}$, and once interpreted as a sequence of antichains - the order $\angle_{\{a,c\}\{b,c\}}$

Note that while for every step sequence $x \in (2^{\Sigma})^*$, $\widehat{x}$ can be interpreted as a sequence of antichains, as every stratified order is interval, not every sequence of antichains is equal $\widehat{x}$ for some step sequence $x \in (2^{\Sigma})^*$. For example $\{a^1,c^1\}\{b^1,c^1\}$ is a sequence of antichains, but there is no step sequence $x \in (2^{\{a,b,c\}})^*$ such that $\widehat{x} = \{a^1,c^1\}\{b^1,c^1\}$. This case is illustrated in Fig. 2. Since for $y = \{a,c\}\{b,c\}$, $\widehat{y} = \{a^1,c^1\}\{b^1,c^2\}$, then $\lhd_y = \lhd_{\widehat{y}} = \lhd_{\{a,c\}\{b,c\}}$. Clearly $\lhd_{\{a,c\}\{b,c\}} = \angle_{\{a^1,c^1\}\{b^1,c^2\}}$ and it is a stratified order, but $\angle_{\{a,c\}\{b,c\}}$ is a different, interval but not stratified, order.

The basic advantage of sequences and step sequences is that their elements may not be unique, $x_1 = aaa$ is a correct sequence and $x_2 = \{a,b\}\{a,b\}\{b,c\}$ is a correct step sequence. The uniqueness required for construction of partial orders is added later by enumeration, i.e. $\widehat{x_1} = a^1a^2a^3$ and $\widehat{x_2} = \{a^1,b^1\}\{a^2,b^2\}\{b^3,c^1\}$; we use enumerated sequence or step sequence $\widehat{x}$ to produce $\lhd_x$. On the other hand, sequences of antichains require all elements that appear in them to be unique. Hence we are forced to define sequences of antichains over $\widehat{\Sigma}$ instead of just Σ. This fact can cause some problems when the sequences of antichains are used to represent operational semantics of concurrent systems.

5 Interval Sequences vs Sequences of Antichains

We will now analyze the mutual relationship between interval sequences and sequences of antichains.

Let z be an interval sequence over Σ, i.e. $z \in \mathsf{InSeq}(\Sigma)$.

- We define $\mathsf{sa}(x)$, a *sequence of antichains generated by* z, as $\mathsf{sa}(z) \in \mathbb{SA}(\widehat{\Sigma})$ such that $\blacktriangleleft_z = \angle_{\mathsf{sa}(z)}$.

For example, if $z = BcBaEaBbEcEb$ then $\mathsf{sa}(x) = \{a,c\}\{b,c\}$ as $\blacktriangleleft_z = \angle_{\{a,c\}\{b,c\}}$.

Let s be a sequence of antichains over Σ, i.e. $s \in \mathbb{SA}(\widehat{\Sigma})$.

- We define $\mathsf{isq}(s)$, *a set of interval sequences generated by* s, as

$$\mathsf{isq}(s) = \{x \mid x \in \mathsf{InSeq}(\Sigma) \wedge \angle_s = \blacktriangleleft_x\}.$$

For example, for $s = \{a, c\}\{b, c\}$, we have $\mathsf{isq}(s) = \{BaBcEaBbEbEc, BaBcEa$ $BbEcEb, BcBaEaBbEbEc, BcBaEaBbEcEb\}$.

The definitions of $\mathsf{sa}(z)$ and $\mathsf{isq}(s)$ are indirect, they use the equality $\angle_s = \blacktriangleleft_z$. We will now provide a direct relationship that does not use the partial orders equality $\angle_s = \blacktriangleleft_z$.

Let $z \in \mathsf{InSeq}(\widehat{\Sigma})$ for some Σ, and $x \subseteq z$. Clearly $x \in \mathscr{BE}_\Sigma$, but it may not belong to $\mathsf{InSeq}(\widehat{\Sigma})$. Moreover, since $z \in \mathsf{InSeq}(\widehat{\Sigma})$ is an *enumerated* sequence, *all its elements are distinct*. We will now show how a subsequence x of z may define some antichains of $\blacktriangleleft_z$.

- We define
$$\mathsf{AC}(x) = \{a^i \mid Ba^i \in x\} \setminus \{a^i \mid Ea^i \in x\}.$$

Clearly if x is an interval sequence, i.e. $x \in \mathsf{InSeq}(\widehat{\Sigma})$ then $\mathsf{AC}(x) = \emptyset$.

Lemma 1. *1. For each $x \subseteq z$, if $\mathsf{AC}(x) \neq \emptyset$ then $\mathsf{AC}(x)$ is an antichain of $\blacktriangleleft_z$.*
2. For every maximal antichain A of $\blacktriangleleft_z$ there is $x \subseteq z$ such that $A = \mathsf{AC}(x)$.

Proof

(1) Suppose $a^i, b^j \in \mathsf{AC}(x)$ and $a^i \blacktriangleleft_z b^j$. Hence $Ba^i, Bb^j \in x$ and $Ba^i \lhd_z Ea^i \lhd_z Bb^j$. But this implies $Ea^i \in x$, a contradiction.
(2) Assume $A = \{a_1, \ldots, a_k\}$ is a maximal antichain of $\blacktriangleleft_z$. Without loss of generality we may assume $Ba_1 \lhd_z \ldots \lhd_z Ba_k$. Let x_A be subsequence of z that starts with Ba_1 and ends with Ba_k. Since A is an antichain, there is no Ea_i, $i = 1, \ldots, k$ in x_A, so $A \subseteq \mathsf{AC}(x_A)$. Suppose $b \in \mathsf{AC}(x_A) \setminus A$. Hence we have $Ba_1 \lhd_z Bb \lhd_z Ba_k$ for some $Bb \in z$ and $Bb \notin A$. If $Ba_k \lhd_z Eb$, then the set $A \cup \{b\}$ is also an antichain, a contradiction as A is a maximal antichain of $\blacktriangleleft_z$. The other case is $Ba_1 \lhd_z Bb \lhd_z Eb \lhd_z Ba_k$, but then $b \notin \mathsf{AC}(x_A)$, so a contradiction again. Hence $A = \mathsf{AC}(x_A)$. □

Clearly not every non-empty $\mathsf{AC}(x)$ is a maximal antichain, and usually there are different x_1 and x_2 such that $\mathsf{AC}(x_1) = \mathsf{AC}(x_2)$.

A sequence $x \subseteq z$ is AC-*complete in* z if the following properties are satisfied:

1. $x = Ba_1 x_0 Ba_k$ or $x = Ba_1$ for some $Ba_1, Ba_k \in z$,
2. $\forall Bb \in z.\ Bb \lhd_z Ba_1 \implies Eb \lhd_z Ba_k$,
3. $\forall Bb \in z.\ Ba_k \lhd_z Bb \implies \exists Ea_i.\ Ba_k \lhd_z Ea_i \lhd_z Bb$.

Intuitively, a sequence x is AC-complete in z if it is the *minimal* sequence such that $\mathsf{AC}(x)$ is a *maximal* antichain of $\blacktriangleleft_z$. Note that if x is a maximal prefix of z built from events beginnings only, i.e. $x = Ba_1 \ldots Ba_k$ and $z = xEa_i y$, then x is AC-complete.

Consider a sequence $z = \overbrace{\underbrace{BfBaBb}_{x_1} EaBcEbBdBe}^{x_3} \overbrace{EdEcEeEf}^{x_2}$. It has three AC-complete subsequences: $x_1 = BfBaBb$, $x_2 = BfBaBbEaBc$ and $x_3 =$

$BfBaBbEaBcEbBdBe$. The sequence $z' = \overbrace{Ba\underbrace{Bb}_{x'_2}Ea}^{x'_1}\overbrace{Bc}^{x'_3}EbBdBeEdEcEe$, has also three AC-complete subsequences: $x'_1 = BaBb$, $x'_2 = BbEaBc$, and $x'_3 = BcEbBdBe$.

We will now show that AC-complete subsequences of z correspond to the maximal antichains of the interval order $\blacktriangleleft_z$.

Proposition 3. *1. If $x \subseteq z$ is AC-complete, then $\mathsf{AC}(x)$ is a maximal antichain of $\blacktriangleleft_z$.*
2. For every maximal antichain A of $\blacktriangleleft_z$ there is an AC-complete x subsequence of z such that $A = \mathsf{AC}(x)$.

Proof

(1) By Lemma 1(1), $\mathsf{AC}(x)$ is an antichain of $\blacktriangleleft_z$. Suppose $x = Ba_1x_0Ba_k$ and $\mathsf{AC}(x) \cup \{b\}$ is also an antichain of $\blacktriangleleft_z$ and $b \notin \mathsf{AC}(x)$. Since $b \notin \mathsf{AC}(x)$ then either $Ba_1 \lhd_z Bb \lhd_z Eb \lhd_z Ba_k$, or $Bb \lhd_z Ba_1$, or $Bb \lhd_z Ba_k$. In the first case $b \blacktriangleleft_z a_k$, so $\mathsf{AC}(x) \cup \{b\}$ is not an antichain. Since x is AC-complete, for the second case we have $Eb \lhd_z Ba_k$, so $b \blacktriangleleft_z a_k$ again, for the third case $Ea_i \lhd_z Bb$ for some $Ea_i \in x$, so $a_i \blacktriangleleft_z b$. Hence in both cases $\mathsf{AC}(x) \cup \{b\}$ is not an antichain. Thus $\mathsf{AC}(x)$ is a maximal antichain.
(2) It suffices to show that x_A from the proof of Lemma 1(2) is AC-complete. The first condition of AC-completeness is clearly satisfied. Suppose $Bb \lhd_z Ba_1$ for some $Bb \in z$. Since $\mathsf{AC}(x)$ is a maximal antichain we clearly have $b \blacktriangleleft_x a_i$ for some a_i, so $Eb \lhd Ba_i \lhd Ba_k$, so the condition (2) also holds. Similarly for condition (3). □

We can easily introduce a total ordering of AC-complete subsequences of z that will correspond to the total ordering of the maximal antichains of z.

Let x, y be AC-complete subsequences of z. We will write

$$x <_z y \iff \forall Ba^i \in \mathscr{A}_x \setminus \mathscr{A}_y . \forall Bb^j \in \mathscr{A}_y .\ Ea^i \lhd_z Bb^j.$$

Proposition 4. *Let $x, y \subseteq z$ be AC-complete sequences. Then*

$$x <_z y \iff \mathsf{AC}(x) \overset{po}{\blacktriangleleft}_z \mathsf{AC}(y).$$

Proof. Since x, y are AC-complete then $\mathsf{AC}(x)$ and $\mathsf{AC}(y)$ are maximal antichains. Moreover,

$$(\forall Ba^i \in \mathscr{A}_x \setminus \mathscr{A}_y . \forall Bb^j \in \mathscr{A}_y .\ Ea^i \lhd_z Bb^j) \iff (\forall a^i \in \mathsf{AC}(x) \setminus \mathsf{AC}(y) . \forall b^j \in \mathsf{AC}(y).\ a^i \blacktriangleleft_z b^j).$$

Hence $x <_z y \iff \mathsf{AC}(x) \overset{po}{\blacktriangleleft}_z \mathsf{AC}(y)$. □

For $z = BfBaBbEaBcEbBdBeEdEcEeEf$ and its three B-complete subsequences $x_1 = BfBaBb$, $x_2 = BfBaBbEaBc$ and $x_3 = BfBaBbEaBcEbBdBe$,

we have $x_1 <_z x_2 <_z x_c$, $\mathsf{AC}(x_1)=\{a,b,f\}$, $\mathsf{AC}(x_2)=\{b,c,f\}$, $\mathsf{AC}(x_3)=\{d,e,f\}$, and $\{a,b,f\} \overset{po}{\blacktriangleleft}_z \{b,c,f\} \overset{po}{\blacktriangleleft}_z \{d,e,f\}$, i.e. $\mathsf{AC}(x_1) \overset{po}{\blacktriangleleft}_z \mathsf{AC}(x_2) \overset{po}{\blacktriangleleft}_z \mathsf{AC}(x_3)$.

For $z' = BaBbEaBcEbBdBeEdEcEe$ and its three B-complete subsequences $x'_1 = BaBb$, $x'_2 = BbEaBc$ and $x'_3 = BcEbBdBe$, we have $x'_1 <_z x'_2 <_z x'_3$, $\mathsf{AC}(x'_1) = \{a,b\}$, $\mathsf{AC}(x'_2) = \{b,c\}$, $\mathsf{AC}(x'_3) = \{c,d,e\}$, and $\{a,b\} \overset{po}{\blacktriangleleft}_z \{b,c\} \overset{po}{\blacktriangleleft}_z \{c,d,e\}$, i.e. $\mathsf{AC}(x'_1) \overset{po}{\blacktriangleleft}_z \mathsf{AC}(x'_2) \overset{po}{\blacktriangleleft}_z \mathsf{AC}(x'_3)$.

In general it is not obvious how to derive a sequence of antichains $\mathsf{sa}(z)$ from a given interval sequence z. A simple algorithm below is based on classical idea of matching left braces, i.e. Ba's, with appropriate right braces, i.e. Ea's (cf. [5]), and implicitly employs the concept of AC-complete subsequences.

The variable j, the sets A_j, $A^{(i)}$ and the sequences of sets s_j are not parts of the algorithm, they were included as they will be used in the proof of the correctness of this algorithm.

Algorithm 1. Input: $n > 0$, $z = \alpha_1 \dots \alpha_n \in \mathsf{InSeq}(\Sigma)$
i:*Integer*; *A*:*set*; *s*:*sequence_of_sets*; *mode*:{'B','E'};
$i \leftarrow 0$; $A \leftarrow \emptyset$; $s \leftarrow \varepsilon$; *mode* $\leftarrow$ 'B'; % $j \leftarrow 1$
for $i = 1$ to n do
  if $\alpha_i = Ba$ then do $A \leftarrow A \cup \{a\}$; *mode* = 'B' od; (1)
  if $\alpha_i = Ea \wedge$ *mode* = 'B' then do $s \leftarrow sA$; % $A_j \leftarrow A$; $s_j \leftarrow s$; $j \leftarrow j+1$; (2a)
    $A \leftarrow A \setminus \{a\}$; *mode* = 'E' od; (2b)
  if $\alpha_i = Ea \wedge$ *mode* = 'E' then do $A \leftarrow A \setminus \{a\}$; *mode* = 'E' od; (3)
    % $A^{(i)} \leftarrow A$;
  od
Output: $\mathsf{sa}(z) = s$ % $\mathsf{sa}(z) = s_{j-1} = A_1 \dots A_{j-1}$ $\diamond$

For the interval sequence $z = BaBbEaBcEbBdBeEdEcEe$, the above algorithm returns the sequence of antichains $\mathsf{sa}(z) = \{a,b\}\{b,c\}\{c,d,e\}$ and clearly $\angle_{\mathsf{sa}(z)} = \blacktriangleleft_z$.

Time complexity of Algorithm 1 is $O(n \log n)$ if the set A is implemented as any kind of balanced tree [5]. This complexity is $O(n^2)$ if the set A is implemented as a list or an array [5]. For the list or array implementation, the time complexity of $A \cup \{a\}$ in line (1) is actually $O(1)$ but unfortunately the complexity of $A \setminus \{a\}$ in lines (2) and (3) is $O(n)$.

While AC-complete subsequences do not appear explicitly in Algorithm 1, they are essential part of its proof of correctness.

Proposition 5. *For each* $z = \alpha_1 \dots \alpha_n \in \mathsf{InSeq}(\Sigma)$ *and* $\mathsf{sa}(z)$ *derived from* z *by Algorithm 1, we have:* $\mathsf{sa}(z) \in \mathbb{SA}(\widehat{\Sigma}) \ \wedge \ \angle_{\mathsf{sa}(z)} = \blacktriangleleft_z$.

Proof. Let h_j be the value of i when the value of j has changed from $j-1$ to j (see line (2a) of the algorithm). Define $z_j = \alpha_1 \dots \alpha_{h_j}$ and consider $z_1 = \alpha_1 \dots \alpha_{h_1}$. Since z_1 is a maximal prefix of z built from Ba's only, it is AC-complete and $A_1 = \mathsf{AC}(z_1)$ is a maximal antichain. Hence $s_1 \in \mathbb{SA}(\widehat{\Sigma})$ and $\angle_{s_1} = \blacktriangleleft_{z_1}$.

Assume $s_j = A_1 \dots A_j \in \mathbb{SA}(\widehat{\Sigma})$ and $\angle_{s_j} = \blacktriangleleft_{z_j}$ and consider the case of $j+1$. The sequence s_j has been constructed in line (2a) just before j has been changed to $j+1$. At the end of line (2a) we still have $A = A_j$.

The lines (2b) and (3) delete from A the events that have ended so no longer belong to the next antichain.

We have to consider two cases, Case 1, when $j+1$ is the biggest value of j, i.e. line (1) is executed that last time for $i = h_{j+1} - 1$, and Case 2 when the value $j+2$ takes place.

(*Case* 1) In this case the algorithm produces $\mathsf{sa}(z) = A_1 \dots A_j \in \mathbb{SA}(\widehat{\Sigma})$ and by the induction assumption we have, $\angle_{\mathsf{sa}(z)} = \blacktriangleleft_z$. Moreover, the lines (2b) and (3) delete the events that have ended from A. Since z is an interval sequence, A becomes empty at the end of loop 'for', i.e. $A^{(n)} = \emptyset$, which ends this case.

(*Case* 2) Let g_k be the value of i when the line (1) is executed first time when the current value of j is k. We have $i = g_{j+1} - 1$ when the line (3) is executed for the last time for the current $j+1$. Consider $A^{(g_{j+1}-1)}$. It contains all elements of A_j that have not executed their endings yet, i.e. $A^{(g_{j+1}-1)} = A_j \cap A_{j+1}$.

Let $i_0 = 1$ and for $j \geq 1$ let i_j be the smallest k such that $Ba = \alpha_k$ and $Ba \in A^{(g_j-1)}$. New Ba's are added to A in the loop 'for' from $i = g_j$ to $i = h_{j+1} - 1$, and $A^{(h_{j+1}-1)} = A_{j+1}$. Define $x_{j+1} = \alpha_{i_j} \dots \alpha_{h_{j+1}-1} \subseteq z$. Clearly $\mathsf{AC}(x_j) = A_{j+1}$. We will show that x_{j+1} is a AC-complete subsequence of z. The first element of x_{j+1}, α_{i_j} is Ba type by its definition, while the last element, $\alpha_{h_{j+1}-1}$ is Ba type by the definition of h_{j+1}, so the condition (1) is satisfied.

Let $x_j = Ba_{j_1} \dots Ba_{j_k}$ be the AC-complete subsequence of z such that $\mathsf{AC}(x_j) = A_j$. We clearly have $Ba_{j_1} \lhd_z Ba_{j_k} \lhd_z \alpha_{h_{j+1}-1}$, so the condition (2) is satisfied for x_{j+1} for all $Bb \lhd_z Ba_{j_1}$. Consider the case $Ba_{j_1} \lhd_z Bb \lhd_z \alpha_{i_j}$. Hence $Bb \in x_j$ and $b \in A_j$ but $b \notin A_{j+1}$. This means that b was deleted from A in either line (2b) or (3) for $i < h_{j+1} - 1$, so $Eb = \alpha_i$ for $i < h_{j+1} - 1$. Thus the condition (2) of AC-completeness is always satisfied. The condition (3) follows from the fact that $\alpha_{h_{j+1}}$ is type Ea. This means x_{j+1} is AC-complete. Clearly $x_j <_z x_{j+1}$ which implies $s_{j+1} = A_1 \dots A_j A_{j+1} \in \mathbb{SA}(\widehat{\Sigma})$ and $\angle_{s_{j+1}} = \blacktriangleleft_{z_{j+1}}$. □

What about an inverse problem? For a given sequence of antichains s, how can we derive an interval sequence z such that both s and z define the same interval order?

Let $s = A_1 \dots A_m$ be a sequence of antichains on $\widehat{\Sigma}$, i.e. $s \in \mathbb{SA}(\widehat{\Sigma})$.

For each $a \in s$, we define:

$$\mathit{first}_s(a) = A_i \text{ if } a \in A_i \text{ and either } i = 1 \text{ or } a \notin A_{i-1}, \text{ and}$$
$$\mathit{last}_s(a) = A_i \text{ if } a \in A_i \text{ and either } i = n \text{ or } a \notin A_{i+1}.$$

For each A_i, we define:

$$\mathsf{B}_s(A_i) = \{Ba \mid \mathit{first}_s(a) = A_i\} \text{ and } \mathsf{E}_s(A_i) = \{Ea \mid \mathit{last}_s(a) = A_i\}.$$

Also, for every set X, let $\mathit{perm}(X)$ denote the set of all permutations of the elements of X.

- For every sequence of antichains $s = A_1 \dots A_m \in \mathbb{SA}(\widehat{\Sigma})$, its set of all *interval sequence representations*, $\mathsf{isr}(s)$, is defined as:

$$\mathsf{isr}(s) = perm(\mathsf{B}_s(A_1))perm(\mathsf{E}_s(A_1)) \dots perm(\mathsf{B}_s(A_m))perm(\mathsf{E}_s(A_m)).$$

For example, for $s = \{a, c\}\{b, c\}$, we have $\mathsf{isr}(s) = \{BaBcEaBbEbEc, BaBcEa BbEcEb, BcBaEaBbEbEc, BcBaEaBbEcEb\}$.

Note that the definition of $\mathsf{isr}(s)$ involves neither interval order $\angle_s$ nor interval orders $\blacktriangleleft_z$ for $z \in \mathsf{isr}(s)$.

Proposition 6. *For every sequence of antichains* $s \in \mathbb{SA}(\widehat{\Sigma})$*:* $\mathsf{isq}(s) = \mathsf{isr}(s)$.

Proof. Let $x \in \mathsf{isr}(s)$, and $a \in s$. Note that either $first_s(a) = last_s(a)$ or $first_s(a) \overset{po}{\angle_s} last_s(a)$, which means $Ba \lhd_x Ea$. Now suppose $a \angle_s b$. Since $a \in last_s(a)$ and $b \in first_s(b)$, from Proposition 1(2), we have $last_a(a) \overset{po}{\angle_s} first_s(b)$. But $\overset{po}{\angle_s}$ is a *total* order of maximal antichains, so $last_a(a) \overset{po}{\angle_s} first_s(b)$ if and only if $x = \dots Ea \dots Bb \dots$, so $a \angle_s b \iff Ea \lhd_x Bb$, i.e. $x \in \mathsf{isq}(s)$.

Let $x \in \mathsf{isq}(s)$. Hence for all $a, b \in s$ we have $Ba \lhd_x Ea$ and $a \angle_s b \iff Ea \lhd_x Bb$. Suppose $x \notin \mathsf{isr}(s)$. For every $y \in \mathsf{isr}(s)$ we can write $x = vx_1$ and $y = vy_1$. Let y_0 be such element of $\mathsf{isq}(s)$ that the length of the prefix v is maximal.

We have to consider four cases:

Case 1. $x = u\ Ba\ u_x$, $y_0 = u\ Bb\ u_{y_0}$. Hence $x = u\ Ba\ v_1\ Bb\ v_2$ and $y_0 = u\ Bb\ z_1\ Ba\ z_2$. Suppose $z_1 = s\ Ec\ s_1$, i.e. $y_0 = u\ Bb\ s\ Ec\ s_1\ Ba\ z_2$, which means $Ec \lhd_{y_0} Ba$ i.e. $c \angle_s a$. Since Ec does not appear in u, we also have $x = u\ Ba\ t\ Ec\ t_1$, which means $Ba \lhd_x Ec$, i.e. $\neg(c \angle_s a)$, a contradiction. This means $z_1 = Ba_1 \dots Ba_m$, so $y_0 = v\ BbBa_1 \dots Ba_m Ba\ z_2$. But $y_0 \in \mathsf{isr}(s)$, so by the definition of $\mathsf{isr}(s)$, we have that $y_1 = vBaBbBa_1 \dots Ba_m\ z_2 \in \mathsf{isr}(s)$, so u is not maximal, as uBa is a prefix of both x and y_1. Therefore the Case 1 cannot happen.

Case 2. $x = u\ Ea\ u_x$, $y_0 = u\ Eb\ u_{y_0}$. Hence $x = u\ Ea\ v_1\ Eb\ v_2$ and $y_0 = u\ Eb\ z_1\ Ea\ z_2$. Suppose $z_1 = s\ Bc\ s_1$, i.e. $y_0 = u\ Eb\ s\ Bc\ s_1\ Ea\ z_2$, which means $Bc \lhd_{y_0} Ea$ i.e. $\neg(a \angle_s c)$. Since Bc does not appear in u, we also have $x = u\ Ea\ t\ Bc\ t_1$, which means $Ea \lhd_x Bc$, i.e. $a \angle_s c$, a contradiction. This means $z_1 = Ea_1 \dots Ea_m$, so $y_0 = v\ EbEa_1 \dots Ea_m Ea\ z_2$. But $y_0 \in \mathsf{isr}(s)$, so from the definition of $\mathsf{isr}(s)$, we have that $y_1 = vEaEbEa_1 \dots Ea_m\ z_2 \in \mathsf{isr}(s)$, so u is not maximal, as uEa is a prefix of both x and y_1. Therefore the Case 2 cannot happen either.

Case 3. $x = u\ Ba\ u_x$, $y_0 = u\ Eb\ u_{y_0}$. Hence $x = u\ Ba\ v_1\ Eb\ v_2$, which means $Ba \lhd_x Eb$, i.e. $\neg(b \angle_s a)$, and $y_0 = u\ Eb\ z_1\ Ba\ z_2$, which means $Eb \lhd_{y_0} Ba$. i.e. $b \angle_s a$, a contradiction, so the Case 3 is not valid.

Case 4. $x = u\ Ea\ u_x$, $y_0 = u\ Bb\ u_{y_0}$. Hence $x = u\ Ea\ v_1\ Bb\ v_2$, which means $Ea \lhd_x Bb$, i.e. $a \angle_s b$, and $y_0 = u\ Bb\ z_1\ Ea\ z_2$, which means $Bb \lhd_{y_0} Ea$. i.e. $\neg(a \angle_s b)$, a contradiction, so the Case 4 is not valid too. □

Assume that $|A_1| + \ldots + |A_m| = n$. Then constructing *an element* of $\mathsf{isr}(s)$, directly from the definition of $\mathsf{isr}(s)$, is $O(mn)$ since constructing $\mathsf{B}_s(A_i)$ is $O(n)$ and constructing $\mathsf{E}_s(A_i)$ is also $O(n)$.

6 Operational Semantics and Sequences of Antichains

In this section we will show how an operational semantics of elementary Petri nets with inhibitor arcs can be expressed in terms of sequences of antichains, i.e. interval orders.

6.1 Elementary Nets with Inhibitor Arcs

Elementary nets with inhibitor arcs [12,14] are very simple. They are just classical *elementary nets* of [17], i.e. one-safe place-transition nets without self-loops, extended with inhibitor arcs[3]. Nevertheless they can easily express complex behaviours involving 'not later than' cases [3,12,14,15], priorities, various versions of simultaneities, etc. [10,13,19].

An inhibitor net is a tuple $N = (P, T, F, I, m_0)$, where P is a set of *places*, T is a set of *transitions*, P and T are disjoint, $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*, $I \subseteq P \times T$ is a set of *inhibitor arcs*, and $m_0 \subseteq P$ is the *initial marking*. An inhibitor arc $(p, e) \in I$ means that e can be enabled only if p is not marked. In diagrams (p, e) is indicated by an edge with a small circle at the end. Any set of places $m \subseteq P$ is called a *marking*.

For every $x \in P \cup T$, the set ${}^\bullet x = \{y \mid (y, x) \in F\}$ denotes the *input* nodes of x and the set $x^\bullet = \{y \mid (x, y) \in F\}$ denotes the *output* nodes of x. The set $x^\circ = \{y \mid (x, y) \in I \cup I^{-1}\}$ is the set of nodes connected by an inhibitor arc to x. The dot-notation extends to sets in the natural way, e.g. the set $X^\bullet$ comprises all outputs of the nodes in X. We assume that for every $t \in T$, both ${}^\bullet t$ and $t^\bullet$ are non-empty and disjoint. These requirements do not always appear in the literature, but following [16,17] we use them for two reasons. Firstly because they are quite natural, and secondly because they allow us to avoid many unnecessary technicalities (cf. [17]). Additionally, both of ${}^\bullet t$ and $t^\bullet$ must have an empty intersection with t°. Figure 3 shows two examples of elementary inhibitor nets, N and N^1.

The *firing sequences semantics*, the simplest operational semantics, is defined in almost the same way as any other kind of Petri nets. The only difference is that for the inhibitor nets, a transition can be enabled only if no place with which it is joined by an inhibitor arc is marked.

Formally, a transition t is *enabled* at marking m if ${}^\bullet t \subseteq m$ and $(t^\bullet \cup t^\circ) \cap m = \emptyset$.

An enabled t can *occur* leading to a new marking $m' = (m \setminus^\bullet t) \cup t^\bullet$, which is denoted by $m[t\rangle m'$.

[3] *Inhibitor arcs* allow a transition to check for an *absence* of a token. In principle they allow 'test for zero', an operator the standard Petri nets do not have. They were introduced in [1].

- A *firing sequence* from the marking m_1 to m_{k+1} is any sequence of transitions $t_1 \ldots t_k$ for which there are markings $m_2, \ldots, m_k$ satisfying:

$$m_1[t_1\rangle m_1[t_2\rangle m_2 \ldots m_k[t_k\rangle m_{k+1}.$$

  In such case we write: $m_1[t_1 \ldots t_k\rangle m_{k+1}$.
- The set of all firing sequences from the marking m to the marking m' is defined as:

$$\mathsf{FS}_N(m{\rightarrow}m') = \{x \in T^* \mid m[x\rangle m'\}.$$

For the net N from Fig. 3, we have $\mathsf{FS}_N(\{s_1, s_2\}{\rightarrow}\{s_4, s_5\}) = \{abc, cab\}$, and corresponding total orders $<_N^{tot1}, <_N^{tot2}$.

Similarly we can define *firing step sequences*, $\mathsf{FSS}_N(m{\rightarrow}m')$. The only difference is that sets of mutually independent transitions can be fired simultaneously. Let $A \subseteq T$ be a non-empty set such that for all distinct $t, r \in A$, we have $(t^\bullet \cup^\bullet t) \cap (r^\bullet \cup^\bullet r) = \emptyset$.

Every such set A is called a *step*, and a step A is a *step enabled* at marking m if ${}^\bullet A \subseteq m$ and $(A^\bullet \cup A^\circ) \cap m = \emptyset$. We also denote $m[A\rangle m'$, where $m' = (m \backslash {}^\bullet A) \cup A^\bullet$.

A *firing step sequence* from the marking m_1 to m_{k+1} is any sequence of non-empty sets of transitions $A_1 \ldots A_k$ for which there are markings $m_2, \ldots, m_k$ satisfying:

$$m_1[A_1\rangle m_1[A_2\rangle m_2 \ldots m_k[A_k\rangle m_{k+1}.$$

In such case we may write: $m_1[A_1 \ldots A_k\rangle m_{k+1}$.

The set of all firing step sequences from the marking m to the marking m' is defined as follows:

$$\mathsf{FSS}_N(m{\rightarrow}m') = \{x \in (\mathscr{P}(T) \setminus \emptyset)^* \mid m[x\rangle m'\}.$$

For the net N, we have $\mathsf{FSS}_N(\{s_1, s_2\}{\rightarrow}\{s_4, s_5\}) = \{\{a\}\{b\}\{c\}, \{c\}\{a\}\{b\}, \{a, c\}\{b\}\}$, and corresponding stratified orders $<_N^{tot1}, <_N^{tot2}, <_N^{strat}$. The interval order $<_N^{int}$ *cannot be represented* by plain step sequences. Moreover the step sequence $\{a\}\{b, c\}$ does *not* belong to $\mathsf{FSS}_N(\{s_1, s_2\}{\rightarrow}\{s_4, s_5\})$, so the stratified order $<_{\neg N}^{strat}$ is not generated by the net N.

As it was aptly stated in [19]: "*If transitions have a beginning and an end, a system state cannot adequately be described by a marking alone; instead, it consists of a marking together with some transitions that have started, but have not finished yet*". One way of describing such system state is the concept of ST-marking, proposed in [18] and explored among others in [19]. Another, slightly simpler for elementary inhibitor nets, way is to use the model recently proposed in [13]. The idea of this approach is briefly illustrated in Fig. 3. If inhibitor arcs are not involved, to represent transitions by their beginnings and ends we might just replace each transition $\boxed{t}$ by the net $\boxed{Bt} \rightarrow (t) \rightarrow \boxed{Et}$ as proposed for example by Zuberek in [21] for Timed Petri nets. Each inhibitor arc must be replaced by two when transformation is made and this construction is explained in detail in [13]. Within this model, the interval order semantics of the net N in Fig. 3

is fully represented by the firing sequence semantics of the net N^1. Assuming that we can 'hold a token' in transitions and holding a token in c overlap with holding tokens in a and b, the net N can generate the interval order $<_N$, which is represented for example by an interval sequence $BaBcEaBbEbEc$ which is a firing sequence of the net N^1.

In this section we will extend the model of [13], so the sequence $\{a^1, c^1\}\{b^1, c^1\}$ can be derived for the net N of Fig. 3, and this is the *sequence of antichains* that defines the interval order $<_N$. The net N^1 in Fig. 3 is to illustrate intuitions and motivations only. Both in [13] and here, all the concepts will be defined in terms of standard inhibitor nets (cf. [12,14]), i.e. the net N. While the inhibitor arc (s_3, Bc) in the net N^1 is rather intuitively evident, the need for the inhibitor arc (b, Bc) might not be so obvious. However when the inhibitor arc (b, Bc) is deleted, the interval sequence $BaEaBbBcEcEb$ is a firing sequence of this new net, say N^2, and $\blacktriangleleft_{BaEaBbBcEcEb} = <_{\neg N}^{strat}$, so this new net N^2 is not equivalent to N. Moreover, we also have $\blacktriangleleft_{BaEaBcBbEcEb} = <_{\neg N}^{strat}$, while the interval sequence $BaEaBcBbEcEb$ is *not* a firing sequence of N^2, so the transformation of N into N^2 is not sound. More detailed discussion can be found in [13].

6.2 Interval Sequence Semantics

We will now briefly recall *interval sequence semantics* of elementary inhibitor nets as proposed in [13]. The key idea is to allow tokens not only in places but in transitions as well. A token in a transition t could be interpreted as 't is active', and removing all tokens from ${}^\bullet t$ and placing one token in t can be interpreted as an execution of Bt, while removing the token from t and placing tokens in $t^\bullet$ can be interpreted as executing Et. The whole definition is given below. It creates a basic structure that we can use to formally define 'holding a token' operational semantics (but without explicit notion of time) in terms of interval sequences.

Let $N = (P, T, F, I, m_0)$ be a given elementary nets with inhibitor arcs.

- For each $t \in T$ we define Bt - the beginning of t and Et - the end of t, and the set
$$\mathscr{T} = \{Bt \mid t \in T\} \cup \{Et \mid t \in T\}.$$
The elements of $\mathscr{T}$ are called *BE-transitions*.
- For each $t \in T$ we define:
$$\begin{array}{ll} {}^\bullet Bt = {}^\bullet t, & Bt^\bullet = \{t\}, \\ {}^\bullet Et = \{t\}, & Et^\bullet = t^\bullet, \\ Bt^\circ = t^\circ \cup (t^\circ)^\bullet, & Et^\circ = \emptyset. \end{array}$$
- We say that a set $m \subseteq P \cup T$ is an *extended marking* if: $m \cap ({}^\bullet m \cup m^\bullet) = \emptyset$.
- A BE-transition $\tau \in \mathscr{T}$ is *enabled* at *extended marking* $m \subseteq P \cup T$ if
$${}^\bullet \tau \subseteq m \text{ and } (\tau^\bullet \cup \tau^\circ) \cap m = \emptyset.$$

- An enabled τ can *occur* leading to a *new extended marking*

$$m' = (m \setminus^\bullet \tau) \cup \tau^\bullet,$$

  which is denoted by: $m[\![\tau\rangle\!\rangle m'$.
- An *extended firing sequence* from the extended marking m_1 to the extended marking m_{k+1} is any sequence of BE-transitions $\tau_1 \ldots \tau_k$ for which there are extended markings $m_2, \ldots, m_k$ satisfying:

$$m_1[\![\tau_1\rangle\!\rangle m_2 \ldots m_k[\![\tau_k\rangle\!\rangle m_{k+1}.$$

  In such case we write: $m_1[\![\tau_1 \ldots \tau_k\rangle\!\rangle m_{k+1}$.
- The set of all *firing interval sequences* from the marking $m \subseteq P$ to the marking $m' \subseteq P$ is defined as:

$$\mathsf{FIS}_N(m{\rightarrow}m') = \{x \in \mathscr{T}^* \mid m[\![x\rangle\!\rangle m'\}$$

It is not immediately obvious that the definition of $\mathsf{FIS}_N(m{\rightarrow}m')$ is sound and complete. This would require the set $\mathsf{FIS}_N(m{\rightarrow}m')$ to satisfy the following two properties

– every element of $\mathsf{FIS}_N(m{\rightarrow}m')$ must be an interval sequence, and
– since all total order representations of a given interval order are considered equivalent and none is preferred, if $x \in \mathsf{FIS}_N(m{\rightarrow}m')$, then $\blacktriangleleft_x = \blacktriangleleft_y$ should imply $y \in \mathsf{FIS}_N(m{\rightarrow}m')$.

The following two results from [13] show that the above two properties are satisfied.

Proposition 7 ([13])

1. *For all markings* $m, m' \subseteq P$, *we have* $\mathsf{FIS}_N(m{\rightarrow}m') \subseteq \mathsf{InSeq}(\mathcal{T}^*)$.
2. *For every* $x \in \mathsf{FIS}_N(m{\rightarrow}m')$ *and every* $y \in \mathscr{T}^*$, *if* $\blacktriangleleft_x{=}\blacktriangleleft_y$ *then* $y \in \mathsf{FIS}_N(m{\rightarrow}m')$. ◇

The proof of Proposition 7(2) is actually quite complex (cf. [13], Appendix A). For the net N from Fig. 3, we have $\mathsf{FIS}_N(\{s_1, s_2\}{\rightarrow}\{s_4, s_4\}) = \mathsf{FS}_{N^1}(\{s_1, s_2\}{\rightarrow}\{s_4, s_4\})$

$$= \left\{ \begin{array}{l} BcEcBaEaBbEb, BaBcEcEaBbEb, BaBcEaEcBbEb, BcBaEcEaBbEb, \\ BcBaEaEcBbEb, BaBcEaBbEbEc, BaBcEaBbEcEb, BcBaEaBbEbEc, \\ BcBaEaBbEcEb, BaEaBbEbBcEc \end{array} \right\}.$$

We will now very briefly discuss the relationship between the interval sequence model of [13] and ST-trace model of [18,19]. In the model of [13] the system states are represented by *extended markings* and in the model of [18,19] by *ST-markings*. For a given net (with or without inhibitor arcs), an *ST-marking* is a pair (m_{ST}, c_{ST}), where $m_{ST} \subseteq P$ is a current marking of N and $c_{ST} \subseteq T$ is the set of *currently firing* transitions. Formally, for every extended marking m, the pair $m_{ST}(m) = (m \cap P, m \cap T)$, is an ST-marking.

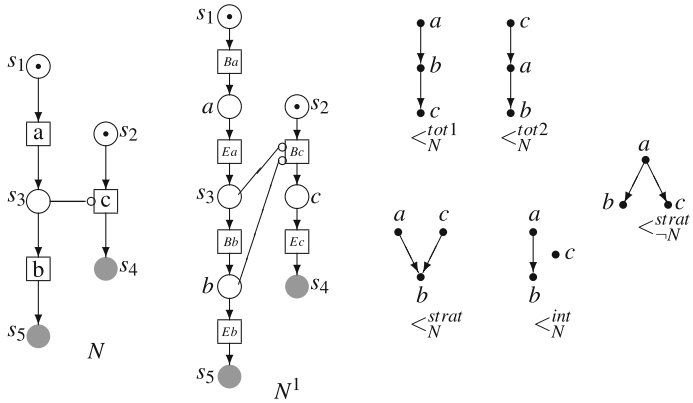


Fig. 3. Interval orders semantics of an inhibitor net. The stratified order $<^{strat}_{\neg N}$ is *not* generated by N or N^1.

An extended firing sequence τ such that $m[\![\tau\rangle\!\rangle m'$, can also be interpreted as a *ST-trace* from the ST-marking $m_{ST}(m)$ to the ST-marking $m_{ST}(m')$. Despite the name *trace*, ST-traces are just *sequences* (more precisely, prefixes of interval sequences) of the elements of $\mathscr{T}$, i.e. Bt's and Et's, not sets of (equivalent w.r.t. some rules) sequences as Mazurkiewicz, step or interval traces [10,13,15].

6.3 Operational Semantics with Sequences of Antichains

We will now extend the interval sequences semantics given above, so we can derive explicitly appropriate sequences of antichains, as $\{a,c\}\{b,c\}$ from the net N of Fig. 3. The basic concept in our approach is the idea of a *path* from one marking to another. This idea can be traced back to at least [18], most likely even earlier as a folklore concept.

Let $m, m' \subseteq P$.

- A ***path*** x from m to m' is a sequence of elements of $2^{P\cup T} \cup \mathscr{T}$,

$$x = m\tau_1 m_1 \tau_2 \ldots m_n \tau_n m',$$

  such that: $m[\![\tau_1\rangle\!\rangle m_1 [\![\tau_2\rangle\!\rangle \ldots m_n [\![\tau_n\rangle\!\rangle m'$.
- ***In such case we will write***: $m\{\!\{x\rangle\!\rangle m'$.
  For the net N of Fig. 3, x_0, y_0 given below are paths:

$$x_0 = \{s_1,s_2\}Ba\{a,s_2\}Bc\{a,c\}Ea\{s_3,c\}Bb\{b,c\}Eb\{s_5,c\}Ec\{s_4,s_5\},$$

$$y_0 = \{s_1,s_2\}Ba\{a,s_2\}Bc\{a,c\}Ea\{s_3,c\}Ec\{s_3,s_4\}Bb\{b,s_4\}Ec\{s_4,s_5\}.$$

- A path $m\tau_1 m_1 \tau_2 \ldots m_{n-1}\tau_n m'$ is *elementary* if for all $i = 1,\ldots,n$, $m_i \cap T \neq \emptyset$.

Clearly every path is a composition of elementary paths (the end of preceding elementary path is the beginning of successor elementary path).

The path x_0 is elementary while y_0 is not as $\{s_3, s_4\} \in y_0$, and y_0 is a composition of elementary paths $\{s_1, s_2\}Ba\{a, s_2\}Bc\{a, c\}Ea\{s_3, c\}Ec\{s_3, s_4\}$ with $\{s_3, s_4\}Bb\{b, s_4\}Ec\{s_4, s_5\}$.

- Let $x = m\tau_1 m_1 \tau_2 \dots m_{n-1}\tau_n m_n$ where $m' = m_n$ be a path from m to m'. We define $\widehat{x}$, an *enumerated path* of x in three steps.

1. First we construct an enumerated version of the sequence $z_x = \tau_1 \dots \tau_n$. Assume that $\widehat{z_x} = \tau_1^{i_1} \dots \tau_n^{i_n}$.
2. We now construct a sequence $\tilde{x}$ as follows:

$$\tilde{x} = m\tau_1^{i_1} m_1 \tau_2^{i_2} \dots m_{n-1}\tau_n^{i_n} m'.$$

   For the path $x_0 = \{s_1, s_2\}Ba\{a, s_2\}Bc\{a, c\}Ea\{s_3, c\}Bb\{b, c\}Eb\{s_5, c\}Ec\{s_4, s_5\}$:
   $\tilde{x_0} = \{s_1, s_2\}Ba^1\{a, s_2\}Bc^1\{a, c\}Ea^1\{s_3, c\}Bb^1\{b, c\}Eb^1\{s_5, c\}Ec^1\{s_4, s_5\}$.
3. We define *enumerated markings* in the following way: $\widehat{m} = m$, $\widehat{m'} = m'$ and for all $k = 1, \dots, n$,
   (a) $\widehat{m_k} \cap P = m_k \cap P$, i.e. elements of P remain unchanged,
   (b) $\forall a \in T.\ a^i \in \widehat{m_k} \iff a \in m_k \wedge \tilde{x} = x_1 Ba^i x_2 m_k x_3 Ea^i x_4$ for some subsequences x_1, x_2, x_3, x_4 of $\tilde{x}$, i.e. we set a^i for all a between Ba^i and Ea^i in $\tilde{x}$.

   We now set

$$\widehat{x} = \widehat{m}\tau_1^{i_1}\widehat{m_1}\tau_2^{i_2} \dots \widehat{m_{n-1}}\tau_n^{i_n}\widehat{m'}.$$

   The sequence $\widehat{x}$ is an ***enumerated path*** generated by x.

For the path x_0 from (2) above $\widehat{x_0} = \{s_1, s_2\}Ba^1\{a^1, s_2\}Bc^1\{a^1, c^1\}Ea^1\{s_3, c^1\}Bb^1\{b^1, c^1\}Eb^1\{s_5, c^1\}Ec^1\{s_4, s_5\}$.

For each path x we define $\mathsf{sap}(x)$, a ***sequence of antichains*** generated by the *path* x, as (see Notation 8):

$$\mathsf{sap}(x) = \widehat{x} \cap \widehat{T}.$$

In principle, $\mathsf{sap}(x)$ is just a projection of $\widehat{x}$ onto the set $\widehat{T}$. For the net N of Fig. 3 and the path $x_0 = \{s_1, s_2\}Ba\{a, s_2\}Bc\{a, c\}Ea\{s_3, c\}Bb\{b, c\}Eb\{s_5, c\}Ec\{s_4, s_5\}$, we have: $\mathsf{sap}(x_0) = \{a^1, c^1\}\{b^1, c^1\}$, and $\angle_{\mathsf{sap}(x_0)} = <_N^{int}$.

We will also write $m \,\|\mathsf{sap}(x)\rangle\!\rangle\, m'$ if $m \{\!\{x\}\!\rangle m'$ for some markings $m, m' \subseteq P$.

We will say that a sequence of antichains $s = A_1 \dots A_m$ is a *generalized step*, if

$$A_i \cap A_{i+1} \neq \emptyset \text{ for } i = 1, \dots, m-1.$$

Intuitively, if s is a generalized step, at each time point at least two events are overlapping. For example $\mathsf{sap}(x_0) = \{a^1, c^1\}\{b^1, c^1\}$ is a generalized step.

Lemma 2. *If x is an elementary path then $\mathsf{sap}(x)$ is a generalized step.*

Proof. Suppose $A_k \cap A_{k+1} = \emptyset$. This means in x, for each $a^i \in A_k$, $b^j \in A_{k+1}$, Ea^i must precede Bb^j so there must be a generalized marking $m_l \subseteq P$ which resides in x between Ea^i's and Bb^j's. Hence x is not elementary. □

The set of all *sequences of antichains* from the marking m to the marking m' is defined as

$$\mathsf{SA}_N(m{\rightarrow}m') = \{\mathsf{sap}(x) \mid m \,\|\mathsf{sap}(x)\rangle\!\rangle m'\} = \{\mathsf{sap}(x) \mid m\langle\!\langle x\rangle\!\rangle m'\}.$$

For the net N of Fig. 3 we have: $\mathsf{SA}_N(\{s_1,s_2\}{\rightarrow}\{s_4,s_5\}) = \{\{a^1\}\{b^1\}\{c^1\}, \{c^1\}\{a^1\}\{b^1\}, \{a^1,c^1\}\{b^1\}, \{a^1,c^1\}\{b^1,c^1\}\}$, and $\{\angle_s \mid s \in \mathsf{SA}_N(\{s_1,s_2\}{\rightarrow}\{s_4,s_5\}) = \{<_N^{tot1}, <_N^{tot2}, <_N^{strat}, <_N^{int}\}$, where $\angle_{\{a^1\}\{b^1\}\{c^1\}} = <_N^{tot1}$, $\angle_{\{c^1\}\{a^1\}\{b^1\}} = <_N^{tot2}$, $\angle_{\{a^1,c^1\}\{b^1\}} = <_N^{strat}$ and $\angle_{\{a^1,c^1\}\{b^1,c^1\}} = <_N^{int}$.

We will now present the main result of this section, that connects interval sequences semantics proposed in [13] with the semantics in terms of interval sequences proposed in this section. In principle this result follows from the fact that for each path $x = m\tau_1 m_1 \tau_2 \ldots m_{n-1}\tau_n m'$, we have $m\langle\!\langle x\rangle\!\rangle m' \iff m[\![\tau_1 \ldots \tau_n\rangle\!\rangle m' \iff m \parallel \mathsf{sap}(x)\rangle\!\rangle m'$, however the proof will also require Proposition 7(2).

Theorem 3. *For all $m, m' \subseteq P$,*

1. $\mathsf{SA}_N(m{\rightarrow}m') = \{\mathsf{sa}(z) \mid z \in \mathsf{FIS}_N(m{\rightarrow}m')\}$,
2. $\mathsf{FIS}_N(m{\rightarrow}m') = \bigcup_{s\in\mathsf{SA}_N(m\rightarrow m')} \mathsf{isq}(s)$.
3. $\{\blacktriangleleft_x \mid x \in \mathsf{FIS}_N(m{\rightarrow}m')\} = \{\angle_s \mid s \in \mathsf{SA}_N(m{\rightarrow}m')\}$.

Proof

(1) Let $s = A_1 \ldots A_t \in \mathsf{SA}_N(m{\rightarrow}m')$. Hence there is a path x such that $s = \mathsf{sap}(x) = \widehat{x} \cap \widehat{T}$ and $m\langle\!\langle x\rangle\!\rangle m'$. Since $x = m\tau_1 m_1 \tau_2 \ldots m_{n-1}\tau_n m'$ then $\widehat{x} = \widehat{m}\tau_1^{l_1}\widehat{m_1}\tau_2^{l_2}\ldots\widehat{m_{n-1}}\tau_n^{l_n}\widehat{m'}$. Since x is a path, we also have $m[\![\tau_1\rangle\!\rangle m_1[\![\tau_2\rangle\!\rangle \ldots m_{n-1}[\![\tau_n\rangle\!\rangle m'$. Assume $A_i = \widehat{m_{j_i}}$ for $i = 1,\ldots,t$, and define $z_x = \tau_1^{i_1}\tau_2^{i_2}\ldots\tau_n^{i_n}$. We will show that $\mathsf{sa}(z_x) = s$. Note that for each $a \in A_i = \widehat{m_{j_i}}$, Ba must precede $\widehat{m_{j_i}}$ in $\widehat{x}$ and Ea follow must $\widehat{m_{j_i}}$, i.e. $Ba \lhd_{\widehat{x}} \widehat{m_{j_i}} \lhd_{\widehat{x}} Ea$. Let $\tau_{k_i}^{i_{k_1}} \in \widehat{x}$ be such Ba that $a \in \widehat{m_{j_i}}$ for each $b \in \widehat{m_{j_i}}\setminus\{a\}$, $Ba = \tau_{k_i}^{i_{k_1}} \lhd_{\widehat{x}} Bb$. Define $x_i = \tau_{k_i}^{i_{k_1}} \ldots \tau_{j_i}^{l_{j_i}}$. Clearly $\tau_{j_i}^{l_{j_i}}\widehat{m_{j_i}} \subseteq \widehat{x}$, and by a reasoning very similar to that in the proof of Proposition 5 we can show that x_i is an AC-complete subsequence of z_x such that $x_i <_{z_x} x_j \iff i < j$ and $\mathsf{AC}(x_i) = A_i$. Hence $\mathsf{sa}(z_x) = s$, i.e. $\mathsf{SA}_N(m{\rightarrow}m') \subseteq \{\mathsf{sa}(z) \mid z \in \mathsf{FIS}_N(m{\rightarrow}m')\}$.
Let $z = \tau_1 \ldots \tau_n \in \mathsf{FIS}_N(m{\rightarrow}m')$, $\widehat{z} = \tau_1^{i_1}\tau_2^{i_2}\ldots\tau_n^{i_n}$, and let $x_z = m\tau_1 m_1 \tau_2 \ldots m_{n-1}\tau_n m_n$, with $m' = m_n$, be an appropriate path such that $m\langle\!\langle x_z\rangle\!\rangle m' \iff m[\![\tau_1 \ldots \tau_n\rangle\!\rangle m'$. This implies $\widehat{x_z} = \widehat{m}\tau_1^{l_1}\widehat{m_1}\tau_2^{l_2}\ldots\widehat{m_{n-1}}\tau_n^{l_n}\widehat{m'}$. Assume that $\mathsf{sa}(z) = A_1 \ldots A_t$, and $x_1,\ldots,x_l$ are AC-complete subsequences of $\widehat{z}$ such that $\mathsf{AC}(x_i) = A_i$ for $i = 1,\ldots,t$. Suppose $x_i = \tau_{k_i}^{i_{k_1}} \ldots \tau_{j_i}^{l_{j_i}}$. Clearly $\tau_{j_i}^{l_{j_i}}\widehat{m_{j_i}} \subseteq \widehat{x_z}$ and, because x_j is AC-complete, we also have $\widehat{m_{j_i}} = \mathsf{AC}(x_j) = A_j$. But this means $\mathsf{sa}(z) = \mathsf{sap}(x_z)$, i.e. $\{\mathsf{sa}(z) \mid z \in \mathsf{FIS}_N(m{\rightarrow}m')\} \subseteq \mathsf{SA}_N(m{\rightarrow}m')$.

(2) Let $z \in \mathsf{FIS}_N(m{\rightarrow}m')$. By (1) above, $\mathsf{sa}(z) \in \mathsf{SA}_N(m{\rightarrow}m')$, and clearly $z \in \mathsf{isq}(\mathsf{sa}(z))$. Hence $\mathsf{FIS}_N(m{\rightarrow}m') \subseteq \bigcup_{s\in\mathsf{SA}_N(m\rightarrow m')} \mathsf{isq}(s)$.
Let $s \in \mathsf{SA}_N(m{\rightarrow}m')$ and $z \in \mathsf{isq}(s)$, i.e. $\angle_s = \blacktriangleleft_z$. Let $x_s = m\tau_1 m_1 \tau_2 \ldots m_{n-1}\tau_n m'$ be a path such that $s = \mathsf{sap}(x)$. Clearly $z_s = \tau_1 \ldots \tau_n \in \mathsf{FIS}_N(m{\rightarrow}m')$ and $\angle_s = \blacktriangleleft_{z_s}$. Hence $\blacktriangleleft_z = \blacktriangleleft_{z_s}$ so by Proposition 7(2), $z \in \mathsf{FIS}_N(m{\rightarrow}m')$, i.e. $\bigcup_{s\in\mathsf{SA}_N(m\rightarrow m')} \mathsf{isq}(s) \subseteq \mathsf{FIS}_N(m{\rightarrow}m')$.

(3) Let $x \in \mathsf{FIS}_N(m{\rightarrow}m')$. Consider $\blacktriangleleft_x$. By (2) there is $s \in \mathsf{SA}_N(m{\rightarrow}m')$ such that $x \in \mathsf{isq}(s)$, so $\angle_s = \blacktriangleleft_x$, i.e. $\{\blacktriangleleft_x \mid x \in \mathsf{FIS}_N(m{\rightarrow}m')\} \subseteq \{\angle_s \mid s \in \mathsf{SA}_N(m{\rightarrow}m')\}$.
Let $s \in \mathsf{SA}_N(m{\rightarrow}m')\}$. By (1) there is $x \in \mathsf{FIS}_N(m{\rightarrow}m')$ such that $s = \mathsf{sa}(x)$; so $\angle_s = \blacktriangleleft_x$, which means $\{\angle_s \mid s \in \mathsf{SA}_N(m{\rightarrow}m')\} \subseteq \{\blacktriangleleft_x \mid x \in \mathsf{FIS}_N(m{\rightarrow}m')\}$. □

Theorem 3 states that for elementary inhibitor nets, the interval sequence semantics of [13] and the antichains sequence semantics of this paper are equivalent. One can be derived from another and they both generate the same set of appropriate interval orders. The advantage of antichains sequences is that each interval order is represented by exactly one sequence of antichains.

7 Final Comments

A detailed relationship between interval sequences of [13] and sequences of antichains, including simple algorithms that transform one into another, has been analyzed. While Fishburn Theorem and representation by sequences of antichains have been known for many years [7,11], this paper appears to be the first one that provides a detailed analysis of their mutual relationship. An operational semantics of elementary nets with inhibitor arcs was defined in terms of sequences of antichains, and was proved to be consistent with the operational interval sequence semantics proposed in [13]. A similar sequences of antichains semantics of nets can be derived from ST-traces of [18,19], but the model would be much more complex.

The results of this paper can be regarded as a promising starting point for more advanced model of *behavioural* semantics, equivalent to the interval traces of [13] or step traces of [10], but with sequences of antichains as models of system runs.

References

1. Agerwala, T., Flynn, M.: Comments on capabilities, limitations and "correctness" of Petri nets. Comput. Architect. News **4**(2), 81–86 (1973)
2. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**, 832–843 (1983)
3. Baldan, P., Busi, N., Corradini, A., Pinna, G.M.: Domain and event structure semantics for Petri nets with read and inhibitor arcs. Theor. Comput. Sci. **323**, 129–189 (2004)

4. Bidoit, M., Hennicker, R., Wirsing, M.: Characterizing behavioural semantics and abstractor semantics. In: Sannella, D. (ed.) ESOP 1994. LNCS, vol. 788, pp. 105–119. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-57880-3_7
5. Cormen, T.H., Leiserson, C.E., Rivest, D.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
6. Fishburn, P.C.: Intransitive indifference with unequal indifference intervals. J. Math. Psychol. **7**, 144–149 (1970)
7. Fishburn, P.C.: Interval Orders and Interval Graphs. Wiley, New York (1985)
8. Grabowski, J.: On partial languages. Fundam. Inform. **4**(2), 427–498 (1981)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Automata Theory, Languages, and Computation. Addison-Wesley, Boston (2001)
10. Janicki, R., Kleijn, J., Koutny, M., Mikulski, Ł.: Step traces. Acta Inform. **53**, 35–65 (2016)
11. Janicki, R., Koutny, M.: Structure of concurrency. Theor. Comput. Sci. **112**, 5–52 (1993)
12. Janicki, R., Koutny, M.: Semantics of inhibitor nets. Inf. Comput. **123**(1), 1–16 (1995)
13. Janicki, R., Yin, X.: Modeling concurrency with interval orders. Inf. Comput. **253**, 78–108 (2017)
14. Kleijn, J., Koutny, M.: Process semantics of general inhibitor nets. Inf. Comput. **190**, 18–69 (2004)
15. Kleijn, J., Koutny, M.: Formal languages and concurrent behaviours. In: Bel-Enguix, G., Jiménez-López, M.D., Martín-Vide, C. (eds.) New Developments in Formal Languages and Applications. SCI, vol. 113, pp. 125–182. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78291-9_5
16. Nielsen, M., Rozenberg, G., Thiagarajan, P.S.: Behavioural notions for elementary net systems. Distrib. Comput. **4**, 45–57 (1990)
17. Rozenberg, G., Engelfriet, J.: Elementary net systems. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 12–121. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_14
18. van Glabbeek, R., Vaandrager, F.: Petri net models for algebraic theories of concurrency. In: de Bakker, J.W., Nijman, A.J., Treleaven, P.C. (eds.) PARLE 1987. LNCS, vol. 259, pp. 224–242. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-17945-3_13
19. Vogler, W.: Partial order semantics and read arcs. Theor. Comput. Sci. **286**(1), 33–63 (2002)
20. Wiener, N.: A contribution to the theory of relative position. In: Proceedings of the Cambridge Philosophical Society, vol. 17, pp. 441–449 (1914)
21. Zuberek, W.M.: Timed Petri nets and preliminary performance evaluation. In: Proceedings of the 7th Annual Symposium on Computer Architecture, La Baule, France, pp. 89–96 (1980)

One Net Fits All

A Unifying Semantics of Dynamic Fault Trees Using GSPNs

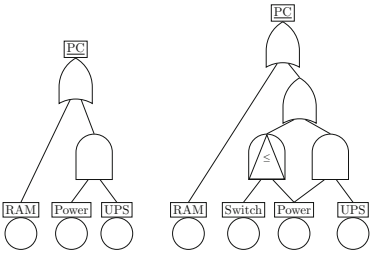
Sebastian Junges[1], Joost-Pieter Katoen[1,2], Mariëlle Stoelinga[2,3], and Matthias Volk[1(✉)]

[1] Software Modeling and Verification, RWTH Aachen University, Aachen, Germany
matthias.volk@cs.rwth-aachen.de
[2] Formal Methods and Tools, University of Twente, Enschede, Netherlands
[3] Department of Software Science, Radboud University Nijmegen, Nijmegen, Netherlands

Abstract. Dynamic Fault Trees (DFTs) are a prominent model in reliability engineering. They are strictly more expressive than static fault trees, but this comes at a price: their interpretation is non-trivial and leaves quite some freedom. This paper presents a GSPN semantics for DFTs. This semantics is rather simple and compositional. The key feature is that this GSPN semantics unifies *all* existing DFT semantics from the literature. All semantic variants can be obtained by choosing appropriate priorities and treatment of non-determinism.

1 Introduction

Fault trees (FTs) [1] are a popular model in reliability engineering. They are used by engineers on a daily basis, are recommended by standards in e.g., the automotive, aerospace and nuclear power industry. Various commercial and academic tools support FTs; see [2] for a survey. FTs visualise how combinations of components faults (their leaves, called basic events) lead to a system failure. Inner tree nodes (called gates) are like logical gates in circuits such as AND and OR. The simple FT in Fig. 1(a) models that a PC fails if either the RAM, or both power and UPS fails.



(a) FT (b) Dynamic FT

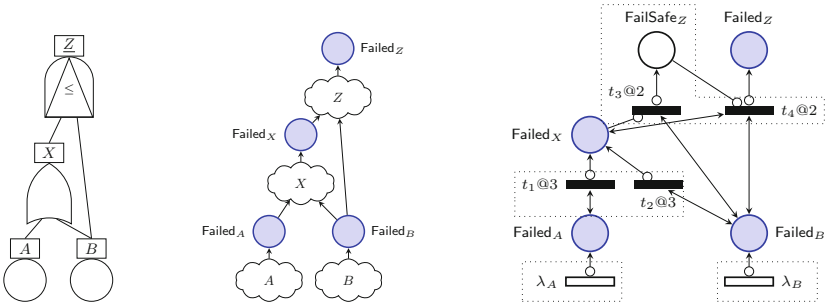
Fig. 1. Fault tree examples

Standard FTs appeal due to their simplicity. However, they lack expressive power to faithfully model many aspects of realistic systems such as spare components, redundancies, etc. This deficiency is remedied by *Dynamic Fault Trees*

This work is supported by the CDZ project CAP, the DFG RTG 2236 "UnRAVeL", the STW project 154747 SEQUOIA, and the EU project SUCCESS.

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 272–293, 2018.
https://doi.org/10.1007/978-3-319-91268-4_14

(DFTs, for short) [3]. They involve a variety of new gates such as spares and functional dependencies. These gates are dynamic as their behaviour depends on the failure history. For instance, the DFT in Fig. 1(b) extends our sample FT. If the power fails while the switch is operational, the system can switch to the UPS. However, if the power fails after the switch failed, their parent PAND-gate causes the system to immediately fail[1]. The expressive power of DFTs allows for modelling complex failure combinations succinctly. This power comes at a price: the interpretation of DFTs leaves quite some freedom and the complex interplay between the gates easily leads to misinterpretations [4]. The DFT in Fig. 2(a) raises the question whether B's failure first causes X to fail which in turn causes Z to fail, or whether B's failure is first propagated to Z making it impossible for Z to fail any more? These issues are not just of theoretical interest. Slightly different interpretations may lead to significantly divergent reliability measures and give rise to distinct underlying stochastic (decision) processes.



(a) DFT (b) Basic scheme (c) Simplified resulting GSPN

Fig. 2. Compositional semantics of DFTs using GSPNs

This paper defines a *unifying* semantics of DFTs using generalised stochastic Petri nets (GSPNs) [5,6]. The use of GSPNs to give a meaning to DFTs is not new; GSPN semantics of (dynamic) fault trees have received quite some attention in the literature [7–10]. Many DFT features are naturally captured by GSPN concepts, e.g., the failure of a basic event can be modelled by a timed transition, the instantaneous failure of a gate by an immediate transition, and places can be exploited to pass on failures. This work builds upon the GSPN-based semantics in [7]. The appealing feature of our GSPN semantics is that it *unifies* various existing DFT semantics, in particular various state-space based meanings using Markov models [11–13], such as continuous-time Markov Chains (CTMC), Markov automata (MA) [14], a form of continuous-time Markov decision process, or I/O interactive Markov chain (IOIMC) [15]. The key is that we capture all these distinct interpretations by a *single* GSPN. The structure of the net is the same for all possible meanings. Only two net features vary: the transition priorities and the partitioning of immediate transitions. The former

[1] A PAND-gate fails if all its children fail in a left-to-right order.

steer the ordering of how failures propagate through a DFT, while the latter control the possible ways in which to resolve conflicts (and confusion) [16].

Table 1. Semantic differences between supported semantics

	Monolithic CTMC [11]	IOIMC [12]	Monolithic MA [13]	Orig. GSPN [7]	New GSPN
Tool support	Galileo [17]	DFTCalc [18]	Storm [19]	--	--
Underlying model	CTMC	IMC [15]	MA [14]	GSPN/CTMC [5,6]	GSPN/MA [16]
Priority gates	$\leq$	$<$	$\leq$	$<$	$\leq$ and $<$
Nested spares	Not supported	Late claiming	Early claiming	Not supported	Early claiming
Failure propagation	Bottom-up	Arbitrary	Bottom-up	Arbitrary	Bottom-up
FDEP forwarding	First	Interleaved	Last	Interleaved	First
Non-determinism	Uniform	True (everywhere)	True FDEP	Uniform	true (PAND, SPARE)

The benefits of a unifying GSPN are manifold. First and foremost, it gives insights in the choices that DFT semantics from the literature—and the tools realising these semantics—make. We show that already three DFT aspects distinguish them all: failure propagation, forwarding in functional dependencies, and non-determinism, see the last three rows in Table 1. Mature tool-support for GSPNs such as SHARPE [20], SMART [21], GreatSPN [22] and its editor [23] can be exploited for all covered DFT semantics. Thirdly, our *compositional* approach, with simple GPSNs for each DFT gate, is easy to extend with more gates. The compositional nature is illustrated in Fig. 2. The occurrence of an event like the failure of a DFT node is reflected by a dedicated (blue) place. The behaviour of a gate is represented by immediate transitions (solid bars) and auxiliary (white) places. Failing BEs are triggered by timed transitions (open bars).

Our framework allows for expressing different semantics by a mild variation of the GSPN; e.g., whether B's failure is first propagated to X or to Z can be accommodated by imposing different transition priorities. The paper supports a rich class of DFTs as indicated in Table 2. The first column refers to the framework, the next four columns to existing semantics from the literature, and the last column to a new instantiation with mild restrictions, but presumably more intuitive semantics. The meaning of the rows is clarified in Sect. 2.2.

Related Work. The semantics of DFTs is naturally expressed by a state-transition diagram such as a Markov model [11–13]. Support of nested dynamic

Table 2. Syntax supported by different semantics

DFT feature	Framework	Monolithic CTMC	IOIMC	Monolithic MA	Orig. GSPN	New GSPN
Share SPAREs	✓	✓	✓	✓	✗	✓
SPARE w/subtree	✓	✗	✓	✓	✗	✓
Shared primary	✓	✗	✓	✗	✗	✓
Priority gates	PAND/POR	PAND	PAND	PAND/POR	PAND	PAND/POR
Downward FDEPs	✓	✗	✓	✓	✗	✗
SEQs on gates	✗	✓	✗	✓	✗	✗
PDEP	✓	✗	✗	✓	✗	✓

gates is an intricate issue, and the resulting Markov model is often complex. To overcome these drawbacks, semantics using higher-order formalisms such as Bayesian Networks [24,25], Boolean logic driven Markov processes [26,27] or GSPNs [7,9] have been proposed. DFT semantics without an underlying state-space have also been investigated, cf. e.g., [28,29]. These semantics often consider restricted classes of DFTs, but can circumvent the state-space explosion. Fault trees have been expressed or extracted from domain specific languages for reliability analysis such as Hip-HOPS, which internally may use Petri net semantics [30]. For a preliminary comparison, we refer to [1,4]. Semantics for DFTs with repairs [8], or maintenance [31] are more involved [32], and not considered in this paper.

Organisation of the Paper. Section 2 introduces the main concepts of GSPNs and DFTs. Section 3 presents our compositional translation from DFTs to GSPNs for the most common DFT gate types. It includes some elementary properties of the obtained GSPNs and reports on prototypical tool-support. Section 4 discusses DFT semantics from the literature based on the unifying GSPN semantics. Section 5 concludes and gives a short outlook into future work. An extended version containing proofs and translations for more DFT gates can be found in [33].

2 Preliminaries

2.1 Generalised Stochastic Petri Nets

This section summarises the semantics of GSPNs as given in [16]. The GSPNs are (as usual) Petri nets with timed and immediate transitions. The former

model the failure of basic events in DFTs, while the latter represent the instantaneous behaviour of DFT gates. Inhibitor arcs ensure that transitions do not fire repeatedly, to naturally model that components do not fail repeatedly. Transition weights allow to resolve possible non-determinism. Priorities will (as explained later) be the key to distinguish the different DFT semantics; they control the order of transition firings for, e.g., the failure propagation in DFTs. Finally, partitions of immediate transitions allow for a flexible treatment of non-determinism.

Definition 1 (GSPN). *A* generalised stochastic Petri net (GSPN) G *is a tuple* $(P, T, I, O, H, m_0, W, \Pi\mathsf{Dom}, \Pi, \mathcal{D})$ *where*

- P *is a finite set of* places.
- $T = T_i \cup T_t$ *is a finite set of* transitions*, partitioned into the set* T_i *of* immediate transitions *and the set* T_t *of* timed transitions.
- $I, O, H \colon T \to (P \to \mathbb{N})$*, the* input-*,* output- *and* inhibition-multiplicities *of each transition, respectively.*
- $m_0 \in M$ *is the* initial marking *with* $M = P \to \mathbb{N}$ *the set of* markings.
- $W \colon T \to \mathbb{R}_{>0}$ *are the* transition-weights.
- $\Pi\mathsf{Dom}$ *is the* priority domain *and* $\Pi \colon T \to \Pi\mathsf{Dom}$ *the* transition-priorities.
- $\mathcal{D} \in 2^{T_i}$*, a* partition *of the immediate transitions.*

For convenience, we write $G = (\mathcal{N}, W, \Pi\mathsf{Dom}, \Pi, \mathcal{D})$ and $\mathcal{N} = (P, T, I, O, H, m_0)$. The definition is as in [16] extended by priorities and with a mildly restricted (i.e., marking-independent) notion of partitions. An example GSPN is given in Fig. 2(c). Places are depicted by circles, transitions by open (solid) bars for timed (immediate) transitions. If $I(t,p) > 0$, we draw a directed arc from place p to transition t. If $O(t,p) > 0$, we draw a directed arc from t to p. If $H(t,p) > 0$, we draw a directed arc from p to t with a small circle at the end. The arcs are labelled with the multiplicities. For all gates in the main text, all multiplicities are one (and are omitted). Some gates in [33] require a larger multiplicity. Transition weights are prefixed with a w, transition priorities with an @, and may be omitted to avoid clutter.

We describe the GSPN semantics for $\Pi\mathsf{Dom} = \mathbb{N}$, and assume in accordance with [6] that for all $t \in T_t : \Pi(t) = 0$ and for all $t \in T_i : \Pi(t) = c > 0$. Other priority domains are used in Sect. 4. The semantics of a GSPN are defined by its *marking graph* which constitutes the state space of a MA. In each marking, a set of transitions are enabled.

Definition 2 (Concession, enabled transitions, firing). *The set* $\mathsf{conc}(m)$ *of* conceded transitions *in* $m \in M$ *is:*

$$\mathsf{conc}(m) = \{t \in T \mid \forall p \in P : m(p) \geq I(t)(p) \wedge m(p) < H(t)(p)\}$$

The set $\mathsf{enabled}(m)$ *of* enabled transitions in m *is:*

$$\mathsf{enabled}(m) = \mathsf{conc}(m) \cap \{t \in T \mid \Pi(t) = \max_{t \in \mathsf{conc}(m)} \Pi(t)\}$$

The effect of firing $t \in \mathsf{enabled}(m)$ on $m \in M$ *is a marking* $\mathsf{fire}(m,t)$ *such that:*

$$\forall p \in P : \mathsf{fire}(m,t)(p) = m(p) - I(t)(p) + O(t)(p).$$

Example 1. Consider again the GSPN in Fig. 2(c). Let $m \in M$ be a marking with $m(\mathsf{Failed}_B) = 1$ and $m(p) = 0$ for all $p \in P \setminus \{\mathsf{Failed}_B\}$. Then the transitions t_2 and t_3 have concession, but only t_2 is enabled. Firing t_2 on m leads to the marking m' with $m'(\mathsf{Failed}_B) = 1 = m'(\mathsf{Failed}_X)$, and $m'(p) = 0$ for $p \in \{\mathsf{Failed}_A, \mathsf{Failed}_Z, \mathsf{FailSafe}_Z\}$.

If multiple transitions are enabled in a marking m, there is a *conflict* which transition fires next. For transitions in different partitions, this conflict is resolved non-deterministically (as in non-stochastic Petri nets). For transitions in the same partition the conflict is resolved probabilistically (as in the GSPN semantics of [6]). Let $C = \mathsf{enabled}(m) \cap D$ be the set of enabled transitions in $D \in \mathcal{D}$. Then transition $t \in C$ fires next with probability $\frac{W(t)}{\sum_{t' \in C} W(t')}$. If in a marking only timed transitions are enabled, in the corresponding state, the sojourn time is exponentially distributed with exit rate $\sum_{t' \in C} W(t')$. If a marking enables both timed and immediate transitions, the latter prevail as the probability to fire a timed transition immediately is zero.

A Petri net is *k-bounded* for $k \in \mathbb{N}$ if for every place $p \in P$ and for every reachable marking $m(p) \leq k$. Boundedness of a GSPN is a sufficient criterion for the finiteness of the marking graph. A k-bounded GSPN has a *time-trap* if its marking graph contains a cycle $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} m$ such that for all $1 \leq i \leq n$, $t_i \in T_i$. The absence of time-traps is important for analysis purposes.

2.2 Dynamic Fault Trees

This section, based on [13], introduces DFTs and their nodes, and gives some formal definitions for concise notation in the remainder of the paper. The DFT semantics are clarified in depth in the main part of the paper.

Fault trees (FTs) are directed acyclic graphs with typed nodes. Nodes without successors (or: *children*), are *basic events* (BEs). All other nodes are *gates*. BEs represent system components that can fail. Initially, a BE is *operational*; it *fails* according to a negative exponential distribution. A gate fails if its *failure condition* over its children is fulfilled. The key gates for static fault trees (SFTs) are typed AND and OR, shown in Fig. 3(b) and (c). These gates fail if all (AND) or at least one (OR) children have failed, respectively. Typically, FTs express for which occurrences of BE failures, a specifically marked node (*top-event*) fails.

(a) BE (b) AND (c) OR (d) PAND (e) POR (f) PDEP (g) SEQ (h) SPARE

Fig. 3. Node types in ((a)–(c)) static and (all) dynamic fault trees.

SFTs lack an internal state—the failure condition is independent of the history. Therefore, SFTs lack expressiveness [2,4]. Several extensions commonly

referred to as *Dynamic Fault Trees* (DFTs) have been introduced to increase the expressiveness. The extensions introduce new node types, shown in Fig. 3(d)–(h); we categorise them as *priority gates*, *dependencies*, *restrictors*, and *spare gates*.

Priority Gates. These gates extend static gates by imposing a condition on the ordering of failing children and allow for order-dependent failure propagation. A *priority-and* (PAND) fails if all its children have failed in order from left to right. Figure 4(a) depicts a PAND with two children. It fails if A fails before B fails. The *priority-or* (POR) [29] only fails if the leftmost child fails before any of its siblings do. The semantics for simultaneous failures is discussed in Sect. 3.2. If a gate cannot fail any more, e.g., when B fails before A in Fig. 4(a), it is *fail-safe*.

Dependencies. Dependencies do not propagate a failure to their parents, instead, when their *trigger* (first child) fails, they update their *dependent events* (remaining children). We consider *probabilistic dependencies* (PDEPs) [24]. Once the trigger of a PDEP fails, its dependent events fail with probability p. Figure 4(b) shows a PDEP where the failure of trigger A causes a failure of BE B with probability 0.8 (provided it has not failed before). *Functional dependencies* (FDEPs) are PDEP with probability one (we omit the p then).

Restrictors. Restrictors limit possible failure propagations. *Sequence enforcers* (SEQ s) enforce that their children only fail from left to right. This differs from priority-gates which do not prevent certain orderings, but only propagate if an ordering is met. The AND SF in Fig. 4(c) fails if A and B have failed (in any order), but the SEQ enforces that A fails prior to B. In contrast to Fig. 4(a), SF is never fail-safe. Another restrictor is the MUTEX (not depicted) which ensures that exactly one of its children fails.

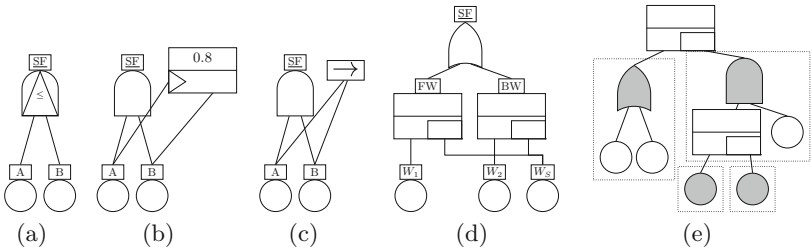


Fig. 4. Simple examples of dynamic nodes [13].

Spare Gates. Consider the DFT in Fig. 4(d) modelling (part of) a motor bike with a spare wheel. A bike needs two wheels to be operational. Either wheel can be replaced by the spare wheel, but not both. The spare wheel is less likely to fail until it is in use. Assume the front wheel fails. The spare wheel is available

and used, but from now on, it is more likely to fail. If any other wheel fails, no spare wheel is available any more, and the parent SPARE fails.

SPAREs involve two mechanisms: *claiming* and *activation*. Claiming works as follows. SPAREs *use* one of their children. If this child fails, the SPARE tries to *claim* another child (from left to right). Only operational children that have not been claimed by another SPARE can be claimed. If claiming fails—modelling that all spare components have failed—the SPARE fails. Let us now consider activation. SPAREs may have (independent, i.e., disjoint) sub-DFTs as children. This includes nested SPAREs, SPAREs having SPAREs as children. A *spare module* is a set of nodes linked to each child of the SPARE. This child is the *module representative*. Figure 4(e) gives an example of spare modules (depicted by boxes) and the representatives (shaded nodes). Here, a spare module contains all nodes which have a path to the representative without an intermediate SPARE. Every leaf of a spare module is either a BE or a SPARE. Nodes outside of spare modules are *active*. For each active SPARE and used child v, the nodes in v's spare module are activated. Active BEs fail with their active failure rate, all other BEs with their passive failure rate.

DFTs Formally. We now give the formal definition of DFTs.

Definition 3 (DFT). *A* Dynamic Fault Tree $\mathcal{F}$ *(DFT) is a tuple* (V, σ, Tp, top)*:*

- V *is a finite set of* nodes*.*
- $\sigma\colon V \to V^*$ *defines the (ordered)* children *of a node.*
- $Tp\colon V \to \{BE\} \cup \{AND, OR, PAND, \dots\}$ *defines the* node-type*.*
- $top \in V$ *is the* top event*.*

For node $v \in V$, we also write $v \in \mathcal{F}$. If $Tp(v) = K$ for some $K \in \{\mathsf{BE}, \mathsf{AND}, \dots\}$, we write $v \in \mathcal{F}_K$. We use $\sigma(v)_i$ to denote the i-th child of v and v_i as shorthand.

We assume (as all known literature) that DFTs are *well-formed*, i.e., (1) The directed graph induced by V and σ is acyclic, i.e., the transitive closure of the parent-child order is irreflexive, and (2) Only the leaves have no children.

For presentation purposes, for the main body we restrict the DFTs to *conventional DFTs*, and discuss how to lift the restrictions in [33].

Definition 4 (Conventional DFT). *A DFT is* conventional *if*

1. *Spare modules are only shared via their (unique) representative. In particular, they are disjoint.*
2. *All children of a SEQ are BEs.*
3. *All children of an FDEP are BEs.*

Restriction 1 restricts the DFTs syntactically and in particular ensures that spare modules can be seen as a single entity w.r.t. claiming and activation. Lifting this restriction to allow for non-disjoint spare modules raises new semantic issues [4]. Restriction 2 ensures that the fallible BEs are immediately deducible. Restriction 3 simplifies the presentation, in Sect. 4.4 we relax this restriction.

3 Generic Translation of DFTs to GSPNs

The goal of this section is to define the semantics of a DFT $\mathcal{F}$ as a GSPN $\mathcal{T}_{\mathcal{F}}$. We first introduce the notion of GSPN templates, and present templates for the common DFT node types such as BE, AND, OR, PAND, SPARE, and FDEP in Sect. 3.2. (Other node types such as PDEP, SEQ, POR, and so forth are treated in [33].) Sect. 3.3 presents how to combine the templates so as to obtain a template for an entire DFT. Some properties of the resulting GSPNs are described in Sect. 3.4 while tool-support is shortly presented in Sect. 3.5.

3.1 GSPN Templates and Interface Places

Recall the idea of the translation as outlined in Fig. 2. We start by introducing the set $\mathcal{I}_{\mathcal{F}}$ of *interface places*:

$$\mathcal{I}_{\mathcal{F}} = \{\mathsf{Failed}_v, \mathsf{Unavail}_v, \mathsf{Active}_v \mid v \in \mathcal{F}\} \cup \{\mathsf{Disabled}_v \mid v \in \mathcal{F}_{\mathsf{BE}}\}$$

The places $\mathcal{I}_{\mathcal{F}}$ manage the communication for the different mechanisms in a DFT. A token is placed in Failed_v once the corresponding DFT gate v fails. On the failure of a gate, the tokens in the failed places of its children are not removed as a child may have multiple parents. Inhibitor arcs connected to Failed_v prevent the repeated failure of an already failed gate. The $\mathsf{Unavail}_v$ places are used for the claiming mechanism of SPAREs, Active_v manages the activation of spare components, while $\mathsf{Disabled}_v$ is used for SEQs.

Every DFT node is translated into some *auxiliary places*, transitions, and arcs. The arcs either connect interface or auxiliary places with the transitions. For each node-type, we define a template that describes how a node of this type is translated into a GSPN (fragment).

To translate contextual behaviour of the node, we use *priority variables* $\boldsymbol{\pi} = \{\boldsymbol{\pi}_v \mid v \in \mathcal{F}\}$. Transition priorities are functions over the priority variables $\boldsymbol{\pi}$, i.e., $\Pi \colon T \to \mathbb{N}[\boldsymbol{\pi}]$. These variables are instantiated with concrete values in Sect. 4, yielding priorities in $\mathbb{N}$. This section does not exploit the partitioning of the immediate transitions; the usage of this GSPN ingredient is deferred to Sect. 4. Put differently, for the moment it suffices to let each immediate transition constitute its (singleton) partition.

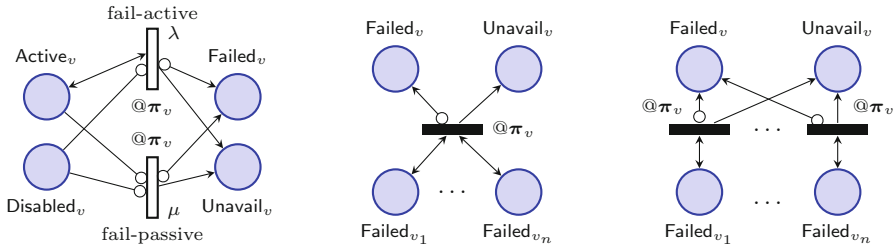
Definition 5 (GSPN-Template). *The GSPN* $\mathcal{T} = (\mathcal{N}, W, \mathbb{N}[\boldsymbol{\pi}], \Pi, \mathcal{D})$ *is a* ($\boldsymbol{\pi}$-parameterised) template over $\mathcal{I} \subseteq P$. *The* instantiation *of* $\mathcal{T}$ *with* $\boldsymbol{c} \in \mathbb{N}^n$ *is the GSPN* $\mathcal{T}[\boldsymbol{c}] = (\mathcal{N}, W, \mathbb{N}, \Pi', \mathcal{D})$ *with* $\Pi'(t) = \Pi(t)(\boldsymbol{c})$ *for all* $t \in T$.

The instantiation replaces the n priority variables by their concrete values.

3.2 Templates for Common Gate Types

We use the following notational conventions. Gates have n children. Interface places $\mathcal{I}$ are depicted using a blue shade; their initial marking is defined by the initialisation template, cf. Sect. 3.3. Other places have an initial token if it is drawn in the template. Transition priorities are indicated by @ and the priority function, e.g., $@\boldsymbol{\pi}_v$. The role of the priorities is discussed in detail in Sect. 4.

Basic Events. Figure 5(a) depicts the template $\mathsf{templ}_{\mathsf{BE}}(v)$ of BE v. It consists of two timed transitions, one for active failure and one for passive failure. Place Failed_v contains a token if v has failed. The inhibitor arcs emanating Failed_v prevent both transitions to fire once the BE has failed. A token in $\mathsf{Unavail}_v$ indicates that v is unavailable for claiming by a SPARE. If Active_v holds a token, the node fails with the active failure rate λ, otherwise it fails with the passive failure rate μ which typically is $c{\cdot}\lambda$ with $0 < c \leq 1$. The place $\mathsf{Disabled}_v$ contains a token if the BE is not supposed to fail. It is used in the description of the semantics of, e.g., SEQ in [33].



(a) BE (b) AND (c) OR

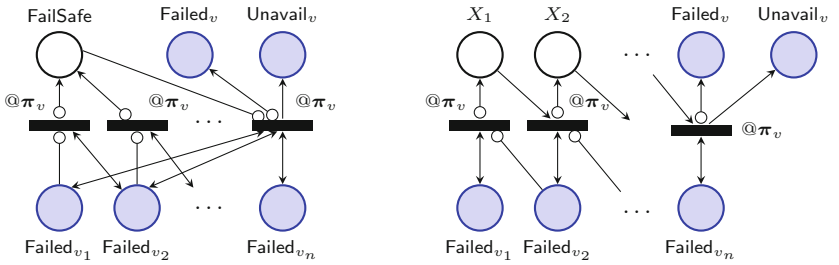
Fig. 5. GSPN templates for basic events and static gates

AND and OR. Figure 5(b) shows the template $\mathsf{templ}_{\mathsf{AND}}(v)$ for the AND gate v. A token is put in Failed_v as soon as the places Failed_{v_i} for all children v_i contain a token. Place Failed_v is thus marked if v has failed. Firing the (only) immediate transition puts tokens in Failed_v and $\mathsf{Unavail}_v$, and returns the tokens taken from Failed_{v_i}. Similar to the BE template, an inhibitor arc prevents the multiple execution of the failed-transition once v failed. The template for an OR gate is constructed analogously, see Fig. 5(c). The failure of one child suffices for v to fail; thus each child has a transition to propagate its failure to Failed_v.

PAND. We distinguish two versions [11] of the priority gate PAND: *inclusive* (denoted $\leq$) and *exclusive* (denoted $<$).

The *inclusive* $\mathsf{PAND}_{\leq}$ v fails if all its children failed in order from left to right while *including simultaneous failures* of children. Figure 6(a) depicts its template. If child v_i failed but its left sibling v_{i-1} is still operational, the $\mathsf{PAND}_{\leq}$ becomes fail-safe, as reflected by placing a token in FailSafe. The inhibitor arc of FailSafe now prevents the rightmost transition to fire, so no token can be put in Failed_v any more. If all children failed from left-to-right and $\mathsf{PAND}_{\leq}$ is not fail-safe, the rightmost transition can fire modelling the failure of the $\mathsf{PAND}_{\leq}$.

The *exclusive* $\mathsf{PAND}_{<}$ v is similar but *excludes the simultaneous failure* of children. Its template is shown in Fig. 6(b) and uses the auxiliary places $X_1, \ldots, X_{n-1}$ which indicate if the previous child failures agree with the strict failure order. A token is placed in X_i if a token is in X_{i-1} and the child v_i has



(a) Inclusive PAND$_{\leq}$ (b) Exclusive PAND$_{<}$

Fig. 6. GSPN templates for inclusive and exclusive PAND

just failed but its right sibling v_{i+1} is still operational. A token can only be put in Failed$_v$ if the rightmost child fails and X_{n-1} contains a token. If the child v_i violates the order, the inhibitor arc from its corresponding transition prevents to put a token in X_{i-1}. This models that PAND$_{<}$ becomes fail-safe.

The behaviour of both PAND variants crucially depends on whether children fail simultaneously or strictly ordered. The moment children fail depends on the order in which failures propagate, and is discussed in detail in Sect. 4.1.

SPARE. We depict the template templ$_{\mathsf{SPARE}}(v)$ for SPARE in two parts: Claiming[2] is depicted in Fig. 7, activation is shown in Fig. 8.

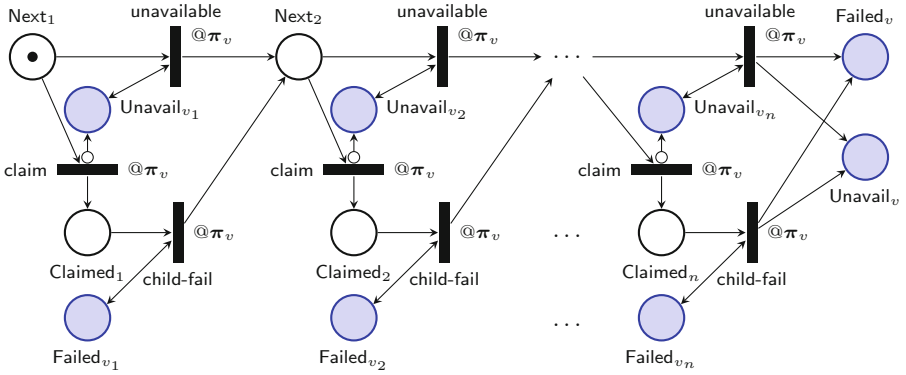


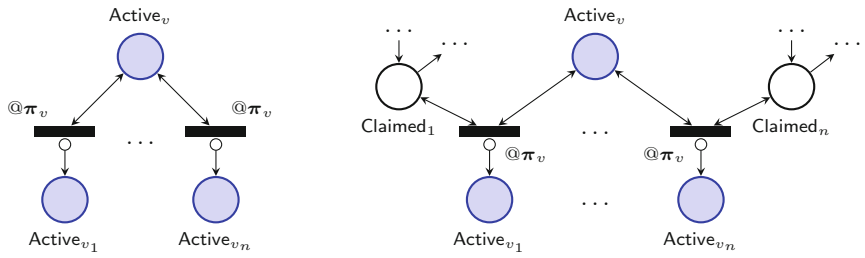
Fig. 7. GSPN template for SPARE, the claiming mechanism

Claiming. templ$_{\mathsf{SPARE}}(v)$ has two sorts of auxiliary places for each child i: Next$_i$ and Claimed$_i$. A token in Next$_i$ indicates that the spare component v_i is the next in line to be considered for claiming. Initially, only Next$_1$ is marked as the

[2] We consider *early* claiming; the concept of *late* claiming is described in [33].

primary child is to be claimed first. A token in $\mathsf{Claimed}_i$ indicates that SPARE v has currently claimed the spare component v_i. This token moves (possibly via $\mathsf{Claimed}_i$) through places Next_i and ends in Failed_v if all children are unavailable or already claimed. The claiming mechanism considers the Unavail places of the children. If $\mathsf{Unavail}_i$ is marked, the i-th spare component cannot be claimed as either the i-th child has failed or it has been claimed by another SPARE. In this case, the transition unavailable fires and the token is moved to Next_{i+1}. Then, spare component $i+1$ has to be considered next.

An empty place $\mathsf{Unavail}_i$ indicates that the i-th spare component is available. The SPARE can claim it by firing the claim transition. This results in tokens in $\mathsf{Claimed}_i$ and $\mathsf{Unavail}_i$, marking the spare component unavailable for other SPAREs. If a spare component is claimed (token in $\mathsf{Claimed}_i$) and it fails, the transition child-fail fires, and the next child is considered for claiming.



(a) Gate (b) SPARE

Fig. 8. GSPN template extensions for the activation mechanism of DFT elements

Activation. When an active SPARE claims a spare component c, all nodes in the spare module (the subtree) M_c become active, i.e., BEs in M_c now fail with their active (rather than passive) failure rate, and SPAREs in M_c propagate the activation downwards. The GSPN extensions for the activation mechanism are given in Fig. 8. The activation in SPAREs is depicted in Fig. 8(b). If a token is in $\mathsf{Claimed}_i$ indicating that the SPARE claimed the ith-child, and the SPARE itself is active, the transition can fire and places a token in Active_{v_i} indicating that the ith-child has become active. Other gates simply propagate the activation to their children as depicted in Fig. 8(a).

FDEP. Figure 9 depicts the template $\mathsf{templ}_{\mathsf{FDEP}}(v)$ for FDEP v; the generalized PDEP is discussed in [33]. If the first child of the FDEP fails, the dependent children fail too. Thus, if Failed_{v_1} is marked, then all transitions can fire and place tokens in the Failed places of the children indicating the failure propagation to dependent nodes. There is no arc to Failed_v as the FDEP itself cannot fail.

FDEPs introduce several semantic problems for DFTs, cf. [4]. This leads to different semantic interpretations which can be captured in our GSPN translation by different values for the priority variables $\boldsymbol{\pi}_v$; as elaborated in Sect. 4.

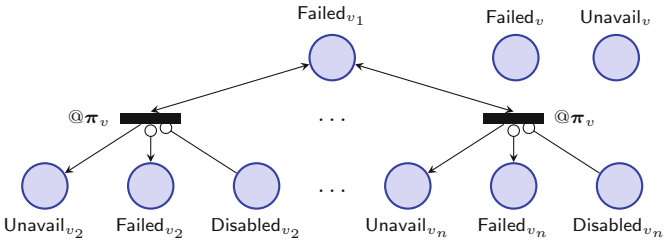


Fig. 9. GSPN template for FDEP

3.3 Gluing Templates

It remains to describe how the GSPN templates for the DFT elements are combined. We define the merging of templates. A more general setting is provided via graph-rewriting, cf. [7].

Definition 6 (Merging Templates). *Let* $\mathcal{T}_i = (\mathcal{N}_i, W_i, \mathbb{N}[\boldsymbol{\pi}], \Pi_i, \mathcal{D}_i)$ *for* $i = 1, 2$ *be* $\boldsymbol{\pi}$*-parameterised templates over* $P_1 \cap P_2 = \mathcal{I}$*. The* merge *of* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *is the* $\boldsymbol{\pi}$*-parameterised template over* $\mathcal{I}$*,* $\mathsf{merge}(\mathcal{T}_1, \mathcal{T}_2) = (\mathcal{N}, W, \mathbb{N}[\boldsymbol{\pi}], \Pi, \mathcal{D})$ *with*

- $P = P_1 \cup P_2$
- $T = T_1 \uplus T_2$, $I = I_1 \uplus I_2$, $O = O_1 \uplus O_2$, $H = H_1 \uplus H_2$
- $m_0 = m_{0,1} + m_{0,2}$
- $W = W_1 \uplus W_2$, $\Pi = \Pi_1 \uplus \Pi_2$, $\mathcal{D} = \mathcal{D}_1 \uplus \mathcal{D}_2$.

An n-ary merge of templates over $\mathcal{I}_\mathcal{F}$ is obtained by concatenation of the binary merge. As the (disjoint) union on sets is associative and commutative, so is the merging of templates. Let $\mathsf{merge}(\mathbb{T} \cup \mathcal{T})$, where $\mathbb{T}$ is a finite non-empty set of templates over some $\mathcal{I}$ and $\mathcal{T}$ is a template over $\mathcal{I}$, denote $\mathsf{merge}(\mathcal{T}, \mathsf{merge}(\mathbb{T}))$.

The GSPN translation converts each DFT node v into the corresponding GSPN using its type-dependent template $\mathsf{templ}_{Tp(v)}$.

Definition 7 (Template for a DFT). *Let DFT* $\mathcal{F} = (V, \sigma, Tp, top)$ *and* $\{\mathsf{templ}_{Tp(v)}(v) \mid v \in \mathcal{F}\}$ *be the set of templates over* $\mathcal{I}_\mathcal{F}$ *each with priority-variable* $\boldsymbol{\pi}_v$*. The* GSPN template $\mathcal{T}_\mathcal{F}$ *for DFT* $\mathcal{F}$ *with places* $P \supset \mathcal{I}_\mathcal{F}$ *is defined by* $\mathcal{T}_\mathcal{F} = \mathsf{merge}\big(\{\mathsf{templ}_{Tp(v)}(v) \mid v \in \mathcal{F}\} \cup \{\mathsf{templ}_{init}\}\big)$.

Initialisation Template. The initialisation template $\mathsf{templ}_{\mathsf{init}}$, see Fig. 10, is ensured to fire once and first, and allows to change the initial marking, e.g., already initially failed DFT nodes. This construct allows to fit the initial marking to the requested semantics without modifying the overall translation. The leftmost transition fires initially, and places a token in Active_{top}. The transition models starting the top-down activation propagation from the top-level node. Furthermore, a token is placed in the place Evidence, enabling the setting of *evidence*, i.e., already failed DFT nodes. If $\{e_1, \ldots, e_n\} \subseteq \mathcal{F}_{\mathsf{BE}}$ is the set of already failed BEs, firing the rightmost transition puts a token in each Failed_{e_i} for all already failed BE e_i.

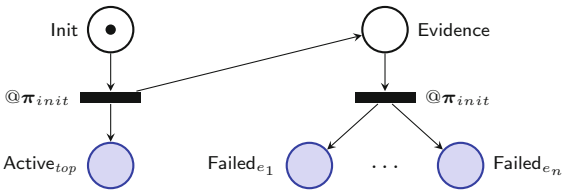


Fig. 10. GSPN template for initialisation

3.4 Properties

We discuss some properties of the obtained GSPN $\mathcal{T}_\mathcal{F}$ for a DFT $\mathcal{F}$. Details can be found in [33].

The size of $\mathcal{T}_\mathcal{F}$ is linear in the size of $\mathcal{F}$. Let $\sigma_{\max} = \max_{v \in \mathcal{F}} |\sigma(v)|$ be the maximal number of children in $\mathcal{F}$. The GSPN $\mathcal{T}_\mathcal{F}$ has no more than $6 \cdot |V| \cdot \sigma_{\max} + 2$ places and immediate transitions, and $2 \cdot |\mathcal{F}_{\mathsf{BE}}|$ timed transitions.

Transitions in $\mathcal{T}_\mathcal{F}$ fire at most once. Therefore, $\mathcal{T}_\mathcal{F}$ does not contain time-traps. Tokens in the interface places Failed_v, Active_v and $\mathsf{Unavail}_v$ are never removed. For such a place p and any transition t, $O(p)(t) \leq I(p)(t)$. Typically, the inhibitor arcs of interface places prevent a re-firing of a transition. In $\mathsf{templ}_{\mathsf{PAND}_<}(v)$, $\mathsf{templ}_{\mathsf{SPARE}}(v)$ and $\mathsf{templ}_{\mathsf{init}}$ tokens move from left to right, and no transition is ever enabled after it has fired.

The GSPN $\mathcal{T}_\mathcal{F}$ is two-bounded, all places except $\mathsf{Unavail}_v$ are one-bounded. Typically, either the inhibitor arcs prevent adding tokens to places that contain a token, or a token moves throughout the (cycle-free) template. However, two tokens can be placed in $\mathsf{Unavail}_v$: One token is placed in $\mathsf{Unavail}_v$ if v is claimed by a SPARE. Another token is placed in $\mathsf{Unavail}_v$ if v failed. The GSPN templates can be easily extended to ensure 1-boundedness of $\mathsf{Unavail}_v$ as well, cf. [33].

3.5 Tool Support

We realised the GSPN translation of DFTs within the model checker Storm [19], version 1.2.1[3]. Storm can export the obtained GSPNs as, among others, GreatSPN Editor projects [23]. Table 3 gives some indications of the obtained sizes of

Table 3. Experimental evaluation of GSPN translations

Benchmark	DFT				GSPN		
	#BE	#Dyn	#Nodes	$\sigma_{\max}$	#Places	#Timed Trans	#Immed. Trans
HECS 5_5_2_np	61	10	107	16	273	122	181
MCS 3_3_3_dp_x	46	21	80	7	246	92	163
RC 15_15_hc	69	33	103	34	376	138	240

[3] http://www.stormchecker.org/publications/gspn-semantics-for-dfts.html.

the GSPNs for some DFT benchmarks from [13]. All GSPN translations could be computed within a second. As observed before, the GSPN size is linear in the size of the DFT.

4 A Unifying DFT Semantics

The interpretation of DFTs is subject to various subtleties, as surveyed in [4]. Varying interpretations have given rise to various DFT semantics in the literature. The key aspects are summarised in Table 1. In the following, we focus on three key aspects—failure propagation, FDEP forwarding, and non-determinism—and show that these suffice to differentiate *all* five DFT semantics, see Fig. 11. Note that we consider the interleaving semantics of nets.

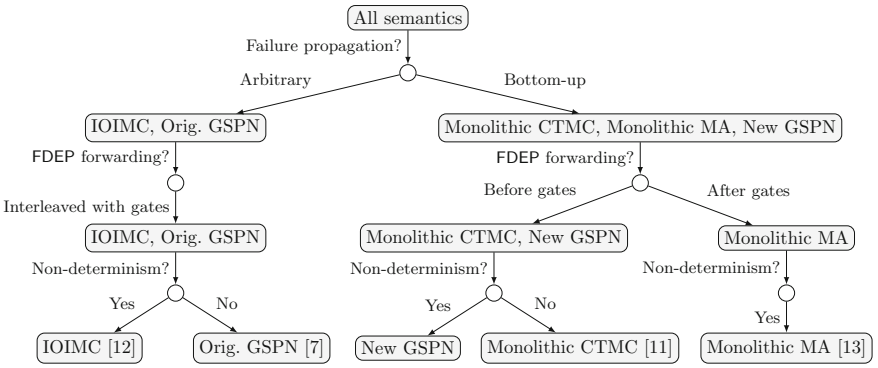
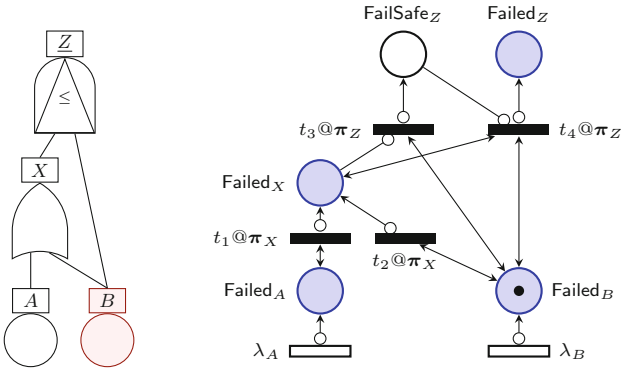


Fig. 11. Decision tree to compare five different DFT semantics

We expose the subtle semantic differences by considering the three aspects using the translated GSPNs of some simple DFTs. The simple DFTs contain structures which occur in industrial case-studies [4]. We vary two ingredients in our net semantics: *instantiations of the priority variables* $\boldsymbol{\pi}$, and the *partitioning* $\mathcal{D}$ *of immediate transitions*. The former constrain the ordering of transitions, while the latter control the treatment of non-determinism. This highlights a key advantage of our net translation: all different DFT semantics from the literature can be captured by small changes in the GSPN. In particular, the net structure itself stays the same for all semantics. Each of the following subsections is devoted to one of the aspects: failure propagation, FDEP forwarding, and non-determinism. Afterwards, we summarise the differences in Table 4.

4.1 Failure Propagation

This aspect is concerned with the order in which failures propagate through the DFT. Consider (a) the DFT $\mathcal{F}_1$ and (b) its GSPN $\mathcal{T}_{\mathcal{F}_1}$ in Fig. 12 and suppose B has failed, as indicated in red and the token in place Failed_B (the same example



(a) DFT $\mathcal{F}_1$ (b) GSPN $\mathcal{T}_{\mathcal{F}_1}$

Fig. 12. Example for failure propagation

was used in the introduction). The question is how B's failure propagates through the DFT. Considering a total ordering on failure propagations, there are two scenarios. *Is B's failure first propagated to gate X, causing* PAND *Z to fail, or is B's failure first propagated to gate Z, turning Z fail-safe?*

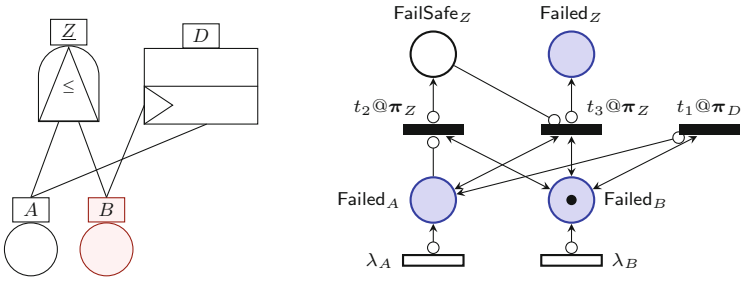
The question reflects in net $\mathcal{T}_{\mathcal{F}_1}$: Consider the enabled transitions t_2 and t_3. Firing t_2 places a token in Failed$_X$ (and in Failed$_B$) and models that B's failure first propagates to X. Next, firing t_4 places a token in Failed$_Z$ and models that the failures of B and X propagate to Z. Now consider first propagating B's failure to Z. This corresponds to firing t_3 and a token in FailSafe$_Z$ modelling that Z is fail-safe. (B's failure can still be propagated to X, but Z remains fail-safe as transition t_4 is disabled due to the token in FailSafe$_Z$.)

The order of failure propagation is thus crucial as it may cause a gate to either fail or to be fail-safe. Existing ways to treat failure propagation are: (1) allow for all possible orders, or (2) propagate failures in a bottom-up manner through the DFT. The former is adopted in the IOIMC and the original GSPN semantics. This amounts in $\mathcal{T}_{\mathcal{F}_1}$ to give all transitions the same priority, e.g., $\boldsymbol{\pi}_v = 1$ for all $v \in \mathcal{F}$. Case (2) forces failures to propagate in a bottom-up manner, i.e., a gate is not evaluated before all its children have been evaluated. This principle is used by the other three semantics. To model this, the priority of a gate v must be lower than the priorities of its children, i.e., $\boldsymbol{\pi}_v < \boldsymbol{\pi}_{v_i}, \forall i \in \{1, \ldots, |\sigma(v)|\}$. In $\mathcal{T}_{\mathcal{F}_1}$, this yields $\boldsymbol{\pi}_Z < \boldsymbol{\pi}_X$, forcing firing t_2 before t_3, see Table 4.

4.2 FDEP Forwarding

The second aspect concerns how FDEPs forward failures in the DFT. Consider (a) the DFT $\mathcal{F}_2$ and (b) its GSPN $\mathcal{T}_{\mathcal{F}_2}$ in Fig. 13. Suppose B fails. The crucial question is—similar to failure propagation—when to propagate B's failure via FDEP D to A. *Is B's failure first propagated via D, causing A and Z to fail, or does B's failure first cause Z to become fail-safe before A fails?* The first scenario is possible as Z is inclusive and A and B are interpreted to fail simultaneously.

In $\mathcal{T}_{\mathcal{F}_2}$, the scenarios are reflected by letting either of the enabled transitions t_1 and t_2 fire first. A similar scenario can be constructed with a PAND$_<$ and an FDEP from A to B.



(a) DFT $\mathcal{F}_2$ [4] (b) GSPN $\mathcal{T}_{\mathcal{F}_2}$

Fig. 13. Example for FDEP forwarding

The order of evaluating FDEPs is thus crucial (as above). We distinguish three options: evaluating FDEPs (1) before, (2) after, or (3) interleaved with failure propagation in gates. The first two options evaluate FDEPs either before or after all other gates, respectively. In $\mathcal{T}_{\mathcal{F}_2}$, these options require that all transitions of an FDEP template get the (1) highest (or (2) lowest, respectively) priority, i.e.,

$$\forall f \in \mathcal{F}_{\text{FDEP}} : \boldsymbol{\pi}_f > \boldsymbol{\pi}_v, \forall v \in \mathcal{F} \setminus \mathcal{F}_{\text{FDEP}} \quad (\text{or, } \boldsymbol{\pi}_f < \boldsymbol{\pi}_v \text{ respectively}).$$

The monolithic CTMC and the new GSPN semantics[4] evaluate FDEPs before gates, whereas the monolithic MA semantics evaluate them after gates. In option (3), FDEPs are evaluated interleaved with the other gates. This option is used by the IOIMC and the original GSPN semantics. In $\mathcal{T}_{\mathcal{F}_2}$, interleaving corresponds to giving all transitions the same priority, e.g. $\boldsymbol{\pi}_v = 1, \forall v \in \mathcal{F}$, see Table 4.

4.3 Non-determinism

The third aspect is how to resolve non-determinism in DFTs. Consider DFT $\mathcal{F}_3$ in Fig. 14 where BE X has failed and FDEP D forwards the failure to BEs A and B. This renders A and B unavailable for SPAREs $S1$ and $S2$. *The question is which one of the failed* SPARE*s ($S1$ or $S2$) claims the spare component C?* This phenomenon is known as a *spare race*. How the spare race is resolved is important: the outcome determines whether PAND Z fails or becomes fail-safe.

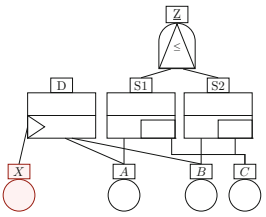


Fig. 14. Example for non-determinism (DFT $\mathcal{F}_3$)

[4] The new GSPN semantics needs further adaptions for downward FDEPs, cf. Sect. 4.4.

The spare race is represented in $\mathcal{T}_{\mathcal{F}_3}$ (depicted in [33]) by a conflict between the claiming transitions of the nets of $S1$ and $S2$. Depending on the previous semantic choices, the race is resolved in different ways. For the monolithic MA semantics, the race is resolved by the order of the FDEP forwarding. For the new GSPN semantics, the race is resolved by the order in which the claim-transitions originating from $\mathsf{templ}_{\mathsf{SPARE}}(S_1)$ and $\mathsf{templ}_{\mathsf{SPARE}}(S_2)$ are handled. In the IOIMC semantics, the winner of the race is determined by the order of interleaving.

For any semantics, the race is represented by a conflict between immediate transitions (with the same priority). We resolve a conflict either by (1) *randomisation*, or (2) *non-determinism*. We realise the randomisation by using weights, i.e., by equipping every immediate transition with the same weight like $W(t) = 1, \forall t \in T$ and letting $\mathcal{D} = T_i$ contain all immediate transitions. A conflict between enabled transitions is then resolved by means of a uniform distribution: each enabled transition is equally probable. This approach reflects the monolithic CTMC and the original GSPN semantics for DFTs.

Case (2) takes non-determinism *as is* and reflects the other three DFT semantics. In this case, in $\mathcal{T}_{\mathcal{F}_3}$ each immediate transition is a separate partition: $\mathcal{D} = \{\{t\} \mid t \in T_i\}$. In many DFTs, the non-determinism is *spurious* and its resolution does not affect standard measures such as reliability and availability. The example $\mathcal{F}_3$ however yields significantly different analysis results depending on how non-determinism is resolved.

Table 4. GSPN differences between supported semantics

DFT semantics	GSPN priority variables		GSPN partitioning
Monolithic CTMC	$\boldsymbol{\pi}_v < \boldsymbol{\pi}_{v_i}$	$\forall v \in \mathcal{F}$, $\forall i \in \{1, \ldots, \lvert\sigma(v)\rvert\}$	$\{T_i\}$
	$\boldsymbol{\pi}_f > \boldsymbol{\pi}_v$	$\forall f \in \mathcal{F}_{\mathsf{FDEP}}, \forall v \notin \mathcal{F}_{\mathsf{FDEP}}$	
IOIMC	$\boldsymbol{\pi}_v = \boldsymbol{\pi}_{v'}$	$\forall v, v' \in \mathcal{F}$	$\{\{t\} \mid t \in T_i\}$
Monolithic MA	$\boldsymbol{\pi}_v < \boldsymbol{\pi}_{v_i}$	$\forall v \in \mathcal{F}$, $\forall i \in \{1, \ldots, \lvert\sigma(v)\rvert\}$	$\{\{t\} \mid t \in T_i\}$
	$\boldsymbol{\pi}_f < \boldsymbol{\pi}_v$	$\forall f \in \mathcal{F}_{\mathsf{FDEP}}, \forall v \notin \mathcal{F}_{\mathsf{FDEP}}$	
Original GSPN	$\boldsymbol{\pi}_v = \boldsymbol{\pi}_{v'}$	$\forall v, v' \in \mathcal{F}$	$\{T_i\}$
New GSPN	$\boldsymbol{\pi}_v \leq \boldsymbol{\pi}_{v_i}$	$\forall v \in \mathcal{F}_{\mathsf{AND}} \cup \mathcal{F}_{\mathsf{OR}}$, $\forall i \in \{1, \ldots, \lvert\sigma(v)\rvert\}$	$\{\{t\} \mid t \in T_i\}$
	$\boldsymbol{\pi}_v < \boldsymbol{\pi}_{v_i}$	$\forall v \notin \mathcal{F}_{\mathsf{AND}} \cup \mathcal{F}_{\mathsf{OR}}$, $\forall i \in \{1, \ldots, \lvert\sigma(v)\rvert\}$	
	$\boldsymbol{\pi}_f \geq \boldsymbol{\pi}_{f_i}$	$\forall f \in \mathcal{F}_{\mathsf{FDEP}}$, $\forall i \in \{2, \ldots, \lvert\sigma(v)\rvert\}$	
	$\boldsymbol{\pi}_f \leq \boldsymbol{\pi}_{f_1}$	$\forall f \in \mathcal{F}_{\mathsf{FDEP}}$	

Remark 1. The semantics of GSPNs [5,6] assigns a weight to every immediate transition. These weights induce a probabilistic choice between conflicting immediate transitions. If several immediate transitions are enabled, the probability of selecting one is determined by its weight relative to the sum of the weights of all enabled transitions, see Sect. 2.1. Under this interpretation, the stochastic process underlying a confusion-free GSPNs is a CTMC. In order to capture the possibility of non-deterministically resolving, e.g., spare races, we use a GSPN semantics [16] where immediate transitions are partitioned. Transitions resolved in a random manner (by using weights) are in a single partition, transitions resolved non-deterministically constitute their own partition—their weights are irrelevant. For confusion-free GSPNs, our interpretation corresponds to [5,6] and yields a CTMC. In general, however, the underlying process is an MA.

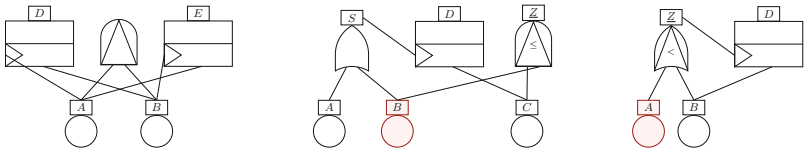
The GSPN adaptations for the different DFT semantics are summarised in Table 4. The last two rows of the table concern FDEPs that are triggered by gates (rather than BEs) and are discussed in detail below.

4.4 Allow FDEPs Triggered by Gates

So far we assumed that FDEP triggers are BEs. We now lift this restriction simplifying the presentation and discuss the options when FDEPs can be triggered by a gate, see Fig. 15(b) and (c). The row "downward" FDEPs in Table 2 reflects this notion. The challenge is to treat *cyclic dependencies*. Cyclic dependencies already occur at the level of BEs, see Fig. 15(a). According to the monolithic CTMC and new GSPN semantics, FDEPs forward failures immediately: All BEs that fail are marked failed before any gate is evaluated, naturally matching bottom-up propagation. The effect is as-if the BEs A and B failed simultaneously. For the new GSPN semantics, we generalise this propagation, and support FDEPs triggered by gates. Consider $\mathcal{F}_5$ in Fig. 15(b): The failure of B indirectly (via S and D) forwards to C. If Z is evaluated after the failure is forwarded to C, the interpretation is that B and C failed simultaneously and the PAND fails, as intended. To guarantee that C is marked failed before Z is evaluated, *S and D require higher priorities than Z* in the net. Consequently, all children of Z are evaluated before Z is evaluated.

Concretely, we generalise bottom-up propagation by refining the priorities: First, we observe that only for dynamic gates, where the order in which children fail matters, the children need to be evaluated strictly before the parents. For other gates, we may weaken the constraints on the priorities. A non-strict ordering suffices: $\forall v \in \mathcal{F}_{\mathsf{AND}} \cup \mathcal{F}_{\mathsf{OR}} : \boldsymbol{\pi}_v \leq \boldsymbol{\pi}_{v_i}, \forall i \in \{1, \ldots, |\sigma(v)|\}$. Second, we mimic bottom-up propagation in FDEP forwarding, meaning that dependent events require a priority not larger than their triggers. Thus, we ensure for each FDEP f, $\boldsymbol{\pi}_f \leq \boldsymbol{\pi}_{f_1}$, and $\boldsymbol{\pi}_f \geq \boldsymbol{\pi}_{f_i}$ for all children $i \neq 1$. Equal priorities are admitted. For FDEPs, like for static gates, the status change is order-independent.

Some DFTs (with FDEPs triggered by gates and cyclic forwarding) do not admit a valid priority-assignment. We argue that the absence of a suitable priority assignment is natural; DFTs without valid priority assignment can model



(a) DFT $\mathcal{F}_4$ (b) DFT $\mathcal{F}_5$ (c) DFT $\mathcal{F}_6$

Fig. 15. Examples for downward FDEP forwarding

a paradox. The DFT $\mathcal{F}_6$ in Fig. 15(c) illustrates this. The new GSPN semantics induce the following constraints:

$$\boldsymbol{\pi}_A < \boldsymbol{\pi}_Z, \quad \boldsymbol{\pi}_B < \boldsymbol{\pi}_Z, \quad \boldsymbol{\pi}_Z \leq \boldsymbol{\pi}_D, \quad \text{and} \quad \boldsymbol{\pi}_D \leq \boldsymbol{\pi}_B.$$

The constraints imply $\boldsymbol{\pi}_B < \boldsymbol{\pi}_B$, which is unsatisfiable. BE A has failed and the exclusive POR Z fails too. (A detailed account of POR-gates is given in [33].) But then B fails because of FDEP D. If we now assume A and B to fail simultaneously, the exclusive POR cannot fail, as its left child A did not fail strictly before B. Then, D's trigger would have never failed. Thus, it is reasonable to exclude such DFTs and consider them ill-formed.

The IOIMC and the monolithic MA semantics support FDEPs triggered by gates, but have different interpretations of simultaneity. The monolithic CTMC semantics is in line with our interpretation, but the algorithm [34] claimed to match this semantics produces deviating results for the DFTs in this sub-section.

5 Conclusions and Future Work

This paper presents a unifying GSPN semantics for Dynamic Fault Trees (DFTs). The semantics is compositional, the GSPN for each gate is rather simple. The most appealing aspect of the semantics is that design choices for DFT interpretations are concisely captured by changing only transition priorities and the partitioning of transitions. Our semantics thus provides a framework for comparing DFT interpretations. Future work consists of extending the framework to DFTs with repairs [8,31] and to study unfoldings [35] of the underlying nets.

References

1. Trivedi, K.S., Bobbio, A.: Reliability and Availability Engineering: Modeling, Analysis, and Applications. Cambridge University Press, Cambridge (2017)
2. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15–16**, 29–62 (2015)
3. Dugan, J.B., Bavuso, S.J., Boyd, M.: Fault trees and sequence dependencies. In: Proceedings of RAMS, pp. 286–293. IEEE (1990)
4. Junges, S., Guck, D., Katoen, J.P., Stoelinga, M.: Uncovering dynamic fault trees. In: Proceedings of DSN, pp. 299–310 (2016)

5. Marsan, M.A., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM TOCS **2**(2), 93–122 (1984)
6. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley, Hoboken (1995)
7. Raiteri, D.C.: The conversion of dynamic fault trees to stochastic Petri nets, as a case of graph transformation. ENTCS **127**(2), 45–60 (2005)
8. Bobbio, A., Raiteri, D.C.: Parametric fault trees with dynamic gates and repair boxes. In: Proceedings of RAMS, pp. 459–465. IEEE (2004)
9. Bobbio, A., Franceschinis, G., Gaeta, R., Portinale, L.: Parametric fault tree for the dependability analysis of redundant systems and its high-level Petri net semantics. IEEE Trans. Softw. Eng. **29**(3), 270–287 (2003)
10. Kabir, S., Walker, M., Papadopoulos, Y.: Quantitative evaluation of Pandora temporal fault trees via Petri nets. IFAC-PapersOnLine **48**(21), 458–463 (2015)
11. Coppit, D., Sullivan, K.J., Dugan, J.B.: Formal semantics of models for computational engineering: a case study on dynamic fault trees. In: Proceedings of ISSRE, pp. 270–282 (2000)
12. Boudali, H., Crouzen, P., Stoelinga, M.: A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE TDSC **7**(2), 128–143 (2010)
13. Volk, M., Junges, S., Katoen, J.P.: Fast dynamic fault tree analysis by model checking techniques. IEEE Trans. Ind. Inform. **14**(1), 370–379 (2018)
14. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: Proceedings of LICS, pp. 342–351. IEEE Computer Society (2010)
15. Hermanns, H.: Interactive Markov Chains: The Quest for Quantified Quality. LNCS, vol. 2428. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45804-2_3
16. Eisentraut, C., Hermanns, H., Katoen, J.-P., Zhang, L.: A semantics for every GSPN. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 90–109. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_6
17. Sullivan, K., Dugan, J.B., Coppit, D.: The Galileo fault tree analysis tool. In: Proceedings of FTCS, pp. 232–235 (1999)
18. Arnold, F., Belinfante, A., Van der Berg, F., Guck, D., Stoelinga, M.: DFTCalc: a tool for efficient fault tree analysis. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) SAFECOMP 2013. LNCS, vol. 8153, pp. 293–301. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40793-2_27
19. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A Storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
20. Trivedi, K.S., Sahner, R.A.: SHARPE at the age of twenty two. SIGMETRICS Perform. Eval. Rev. **36**(4), 52–57 (2009)
21. Ciardo, G., Miner, A.S., Wan, M.: Advanced features in SMART: the stochastic model checking analyzer for reliability and timing. SIGMETRICS Perform. Eval. Rev. **36**(4), 58–63 (2009)
22. Baarir, S., Beccuti, M., Cerotti, D., Pierro, M.D., Donatelli, S., Franceschinis, G.: The GreatSPN tool: recent enhancements. SIGMETRICS Perform. Eval. Rev. **36**(4), 4–9 (2009)
23. Amparore, E.G.: A new GreatSPN GUI for GSPN editing and CSL^{TA} model checking. In: Norman, G., Sanders, W. (eds.) QEST 2014. LNCS, vol. 8657, pp. 170–173. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10696-0_13

24. Montani, S., Portinale, L., Bobbio, A., Raiteri, D.C.: Radyban: a tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks. Reliab. Eng. Syst. Saf. **93**(7), 922–932 (2008)
25. Boudali, H., Dugan, J.B.: A continuous-time Bayesian network reliability modeling, and analysis framework. IEEE Trans. Reliab. **55**(1), 86–97 (2006)
26. Bouissou, M., Bon, J.L.: A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. Reliab. Eng. Syst. Saf. **82**(2), 149–163 (2003)
27. Rauzy, A., Blériot-Fabre, C.: Towards a sound semantics for dynamic fault trees. Reliab. Eng. Syst. Saf. **142**, 184–191 (2015)
28. Merle, G., Roussel, J.M., Lesage, J.J.: Quantitative analysis of dynamic fault trees based on the structure function. Qual. Reliab. Eng. Int. **30**(1), 143–156 (2014)
29. Walker, M., Papadopoulos, Y.: Qualitative temporal analysis: towards a full implementation of the fault tree handbook. Control Eng. Pract. **17**(10), 1115–1125 (2009)
30. Chen, D., Mahmud, N., Walker, M., Feng, L., Lönn, H., Papadopoulos, Y.: Systems modeling with EAST-ADL for fault tree analysis through HiP-HOPS. IFAC Proc. Vol. **46**(22), 91–96 (2013)
31. Guck, D., Spel, J., Stoelinga, M.: DFTCalc: reliability centered maintenance via fault tree analysis (tool paper). In: Butler, M., Conchon, S., Zaïdi, F. (eds.) ICFEM 2015. LNCS, vol. 9407, pp. 304–311. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25423-4_19
32. Raiteri, D.C.: Integrating several formalisms in order to increase fault trees' modeling power. Reliab. Eng. Syst. Saf. **96**(5), 534–544 (2011)
33. Junges, S., Katoen, J.P., Stoelinga, M., Volk, M.: One net fits all: a unifying semantics of dynamic fault trees using GSPNs. CoRR abs/1803.05376 (2018)
34. Manian, R., Coppit, D.W., Sullivan, K.J., Dugan, J.B.: Bridging the gap between systems and dynamic fault tree models. In: Proceedings of RAMS, pp. 105–111 (1999)
35. Engelfriet, J.: Branching processes of Petri nets. Acta Inform. **28**(6), 575–591 (1991)

On the Structure of Cycloids Introduced by Carl Adam Petri

Rüdiger Valk(✉)

Department of Informatics, University of Hamburg, Hamburg, Germany
valk@informatik.uni-hamburg.de

Abstract. Cycloids are particular Petri nets for modelling processes of actions or events. They belong to the fundaments of Petri's general systems theory and have very different interpretations, ranging from Einstein's relativity theory to elementary information processing gates. Despite their simple definitions, their properties are still not completely understood. This contribution provides for the first time a formal definition together with new results concerning their structure. For instance, it is shown that the minimal length of a cycle is the length of a local basic circuit, possibly decreased by an integer multiple of the number of semi-active transitions.

Keywords: Analysis and synthesis · Structure of nets · Cycloids
General net theory

1 Introduction

Cycloids were discovered by C. A. Petri to describe fundamental processes running in time and space. They are based on Minkowski's spacetime model, but use causal dependence instead of numeric distance. They have been introduced in [7] in the section on physical spaces to illustrate the shift from Galilei to Lorentz transformation. Of particular importance to Petri was his discovery that fundamental gates of boolean circuits, such as XOR-transfer, majority-transfer, or Quine-transfer, are topologically equivalent to some of his cycloids (see Fig. 9). In private communication with the author, he explained that he discovered this relationship by pure diligence without any methodical approach. Petri usually introduced the concept using the regimen or organization rule for people carrying buckets to extinguish a fire [7] or by cars driving in line on a road with varying distances as shown in Fig. 1 (from [9]). In the corresponding causal and infinite net, cars are represented by black tokens moving forward in time and space, whereas the gaps are moving also forward in time but in the opposite spacial direction[1]. As the concept considerably differs from Minkowski's space and

[1] In the net, the gaps are also ordinary black tokens, but represented here by a cross to distinguish them from the cars.

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 294–314, 2018.
https://doi.org/10.1007/978-3-319-91268-4_15

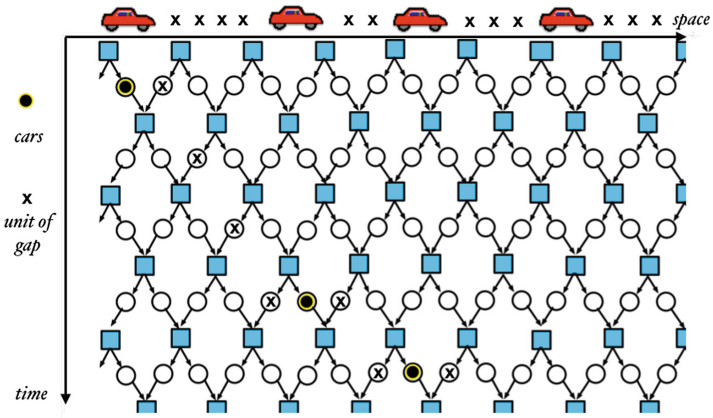


Fig. 1. Cars in Petri space.

is based uniquely on causal dependencies, we call it Petri space[2]. Petri defined the *slowness* of the system by the quotient of the difference of the numbers of gap units and cars (in a suitable section of a case), divided by the sum of the numbers of gaps and cars. In the example this is $w := \frac{gaps-cars}{gaps+cars} = \frac{12-4}{12+4} = \frac{1}{2}$. In [9] he wrote: "The concept of slowness is a key to understanding repetitive group behaviour. It can be applied to organization, to workflow (just-in-time production), and to physical systems".[3] In the report [11], Stehr states that the nets of Fig. 2 have slowness $w = 0$ on the left-hand side and slowness $w = 1/3$ on the right-hand. Intuitively it is visible that the first net is "faster" as more transitions can occur concurrently. But the parameters α and β, which have been used by Petri to define the slowness, are not directly visible from the net. In this paper we show how to compute these parameters.

In this context systems are considered as finite repetitive structures. They are constructed by folding the Petri space. In a first step, a finite space is assumed. Hence after some finite number of steps the initial state is reached again. This is modelled by folding the Petri space in such a way that transitions **a** and **b** (as well as **d** and **c**) of Fig. 3 are identified. The resulting still infinite net is called a *time orthoid* ([7], p. 37), as it extends infinitely in temporal direction.

Next, the analogous step is done with respect to time, i.e. transitions **a** and **d** are identified. The resulting net is finite, and all the four vertices **a, b, c** and **d** are identified. In the middle of the figure, the resulting net is represented by the output of a cycloid tool[4]. On the right hand side, a redesign is shown

[2] In his Hamburg lecture 2004 [8] and in [9] Petri introduced the denomination *natural coordinates* whereas in an earlier publication [7] the term *Minkowski coordinates* has been used. This shows that he also saw the necessity to use a different name.

[3] See more about the notion of slowness in Sect. 4.

[4] The cycloid model generator *cyclogen* written by Fenske, in combination with the RENEW tool.

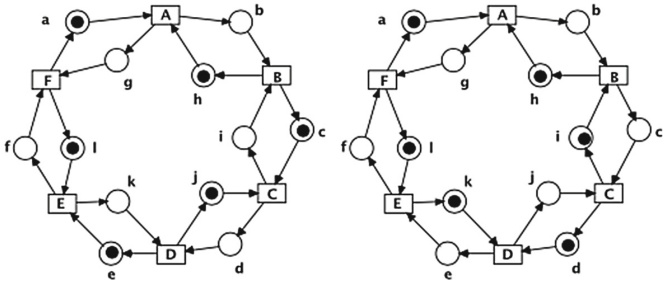


Fig. 2. Cycloids $\mathcal{C}(3,3,1,1)$ and $\mathcal{C}(4,2,1,1)$

emphasizing the cyclic structure[5]. Note that the cycle leading in a straight line from transition **a** via 3 other transitions to transition **c**, the latter identified with **a** in the folding, corresponds to the cycle **t1, t2, t3, t4** in the right-hand side nets. Other examples generated with that tool are given in the Figs. 6 and 8.

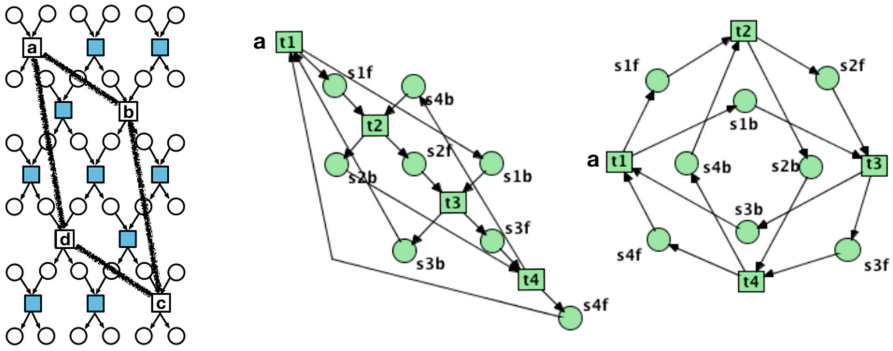


Fig. 3. Folding in Petri space.

The aims of this paper are threefold:

- From the lectures and seminars, Petri gave in the years 1988–2004 at the University of Hamburg, there is a lot of unpublished knowledge in hand-written scripts by Petri and in the notes and the memory of the persons which attended the events [6]. Some of this knowledge is put into writing in this paper.
- The introduction of cycloids in [7] is rather short. This paper provides formal definitions that allow to prove some properties.
- New properties of cycloids are found, like symmetry properties or length-of-shortest-cycles, that allow new analytic procedures.

We acknowledge the work of Uwe Fenske, who collected much of the material mentioned above and contributed formal Definitions 2, 3 and 8 as well as Lemma 4

[5] The net is known as "oscillator net" or "four seasons net". See also Fig. 9.

in a thesis [1]. This thesis also contains a large number of explanations and motivations of Petri's concepts. Later he implemented the cycloid tool mentioned above, which allowed to create numerous examples by the construction of cycloid systems in form of RENEW nets. We also acknowledge Peter Langner who wrote the web tool "Cycloids' Characteristics" [5], which was very useful to investigate cycloids in the form of the fundamental parallelogram (see Figs. 6 and 8). We further acknowledge the help of Lawrence Cabac who provided a plug-in for the computation of cycles from a RENEW net. Finally thanks to Uwe Fenske, Mark-Oliver Stehr, Bernd Neumann and Olaf Kummer for comments on this paper, the latter for suggesting Theorem 5(b) and (c).

2 Nets, Net Systems and Petri Space

Definition 1. *A* net $\mathcal{N}$ *is defined by a triple* (S, T, F) *where* S *is a set, called set of* state elements *or* places*, a set* T *of* transitions *and a* flow relation F*, with the following restrictions:*

(a) $S \cap T = \emptyset$	*(S and T are disjoint sorts)*
(b) $F \subseteq S \times T \cup T \times S$	*(only distinct sort elements are connected by arcs)*
(c) $F \cap F^{-1} = \emptyset$	*(no selfloops)*
(d) $\mathrm{dom}(F) \cup \mathrm{ran}(F) = S \cup T$	*(no isolated elements).*

An element from $X := S \cup T$ *is said to be a* net element *of* $\mathcal{N}$*.* ${}^\bullet x := F^{-1}[x]$[6]*,* $x^\bullet := F[x]$ *denote the input and output elements of an element* x*, respectively.*

A transition $t \in T$ *is* active *in a marking* $M \subseteq S$ *if* $0 \leq |{}^\bullet t \cap M| = |{}^\bullet t|$ *and* $t^\bullet \cap M = \emptyset$*. In this case the* follower marking M' *is defined and given by* $M' = M \backslash {}^\bullet t \cup t^\bullet$*.*

A transition $t \in T$ *with* $|{}^\bullet t| \geq 2$ *is* semi-active *in a marking* $M \subseteq S$ *if* $0 < |{}^\bullet t \cap M| < |{}^\bullet t|$ *and* $t^\bullet \cap M = \emptyset$*.*

A transition $t \in T$ *with* $|{}^\bullet t| \geq 1$ *is* input-marked *or* marked *in a marking* $M \subseteq S$ *if* $0 < |{}^\bullet t \cap M| \leq |{}^\bullet t|$ *and* $t^\bullet \cap M = \emptyset$*.*

A net (S, T, F) *together with an initial marking* M_0 *is called a* net system*.*

Active transitions follow the usual definition: ${}^\bullet t \subseteq M$ and $t^\bullet \cap M = \emptyset$. For a semi-active transition there are some, but not sufficiently many input tokens. A transition, with $|{}^\bullet t| \geq 2$, is marked if it is active or semi-active.

Definition 2. *A Petri space is defined by the net*
$\mathcal{PS}_1 := (S_1, T_1, F_1)$ *where*

$$S_1 = S_1^{\rightarrow} \cup S_1^{\leftarrow}, \quad S_1^{\rightarrow} = \{s_{\xi,\eta}^{\rightarrow} \,|\, \xi, \eta \in \mathbb{Z}\}, \quad S_1^{\leftarrow} = \{s_{\xi,\eta}^{\leftarrow} \,|\, \xi, \eta \in \mathbb{Z}\}, \quad S_1^{\rightarrow} \cap S_1^{\leftarrow} = \emptyset$$
$$T_1 = \{t_{\xi,\eta} \,|\, \xi, \eta \in \mathbb{Z}\},$$
$$F_1 = \{(t_{\xi,\eta}, s_{\xi,\eta}^{\rightarrow}) \,|\, \xi, \eta \in \mathbb{Z}\} \cup \{(s_{\xi,\eta}^{\rightarrow}, t_{\xi+1,\eta}) \,|\, \xi, \eta \in \mathbb{Z}\} \cup$$
$$\{(t_{\xi,\eta}, s_{\xi,\eta}^{\leftarrow}) \,|\, \xi, \eta \in \mathbb{Z}\} \cup \{(s_{\xi,\eta}^{\leftarrow}, t_{\xi,\eta+1}) \,|\, \xi, \eta \in \mathbb{Z}\}.$$

$S_1^{\rightarrow}$ *is called set of* forward places *and* $S_1^{\leftarrow}$ *the set of* backward places.

[6] $F^{-1}[x]$ is the relational image of the element x with respect to the inverse of the relation F.

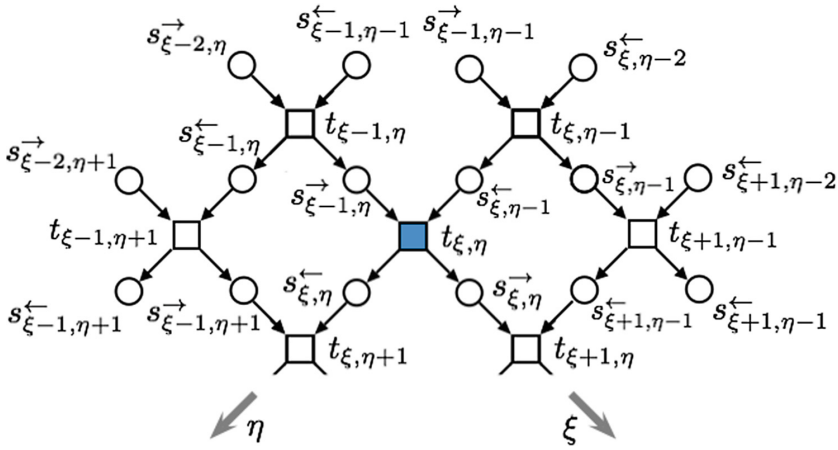


Fig. 4. Denomination of Petri space elements

The introduced denominations of the Petri space elements are shown in Fig. 4 and should be compared with Fig. 1. Contrary to the Minkowski space, the Petri space is independent of an embedding into $\mathbb{Z} \times \mathbb{Z}$. It is therefore suitable for the modelling in transformed coordinates as in non-Euclidian space models. However, the reader will wonder that we will apply linear algebra, for instance using equations of lines. This is done only to determine the relative position of points. It can be understood by first topologically transforming and embedding the space into $\mathbb{R} \times \mathbb{R}$, calculating the position and then transforming back into the Petri space. Distances, however, are not computed with respect to the Euclidean metric, but by counting steps in the grid of the Petri space.

3 Cycloids

Definition 3. *A* cycloid *is a net* $\mathcal{C}(\alpha, \beta, \gamma, \delta) = (S, T, F)$*, defined by parameters* $\alpha, \beta, \gamma, \delta \in \mathbb{N}_+$[7] *as a quotient of the Petri space* $\mathcal{PS}_1 := (S_1, T_1, F_1)$ *(Definition 2) with respect to the equivalence relation* $\equiv \subseteq X_1 \times X_1$ *with*[8]

$\equiv[S_1^{\rightarrow}] \subseteq S_1^{\rightarrow}, \quad \equiv[S_1^{\leftarrow}] \subseteq S_1^{\leftarrow}, \quad \equiv[T_1] \subseteq T_1,$

$x_{\xi,\eta} \equiv x_{\xi+m\alpha+n\gamma,\, \eta-m\beta+n\delta}$ *for all* $\xi, \eta, m, n \in \mathbb{Z}$, $X = X_1/_{\equiv}$

$[\![x]\!]_{\equiv}\ F\ [\![y]\!]_{\equiv} \Leftrightarrow \exists x' \in [\![x]\!]_{\equiv}\ \exists y' \in [\![y]\!]_{\equiv} : x'\ F_1\ y' \qquad$ *for all* $x, y \in X_1$.

Isomorphic nets are denominated as cycloids, as well.

From [1] we cite the following lemma.

[7] $\mathbb{N}_+$ denotes the set of positive integers.

[8] $\equiv[A]$ is the relational image of the set A with respect to the relation $\equiv$.
$[\![x]\!]_{\equiv}$ denotes the equivalence class to which x belongs in the quotient $X_1/_{\equiv}$.

Lemma 4. *The natural application with respect to* $\equiv \subseteq X_1 \times X_1$ *and explicitely specified by* $f_\equiv : X_1 \to X$, $f_\equiv(x_{\xi,\eta}) = [\![x_{\xi,\eta}]\!]_\equiv$ *and the property* $f_\equiv(x_{\xi,\eta}) = f_\equiv(x_{\xi+m\alpha+n\gamma,\,\eta-m\beta+n\delta})$ *for arbitrary* $\xi, \eta, m, n \in \mathbb{Z}$ *is a net morphism, and particularly a folding and a quotient.*

Proof. Following [10] the map $f_\equiv$ is a morphism if and only if $\equiv \cap (S_1 \times T_1) = \emptyset$. This holds in our case as property (b) of Definition 1 is preserved by $f_\equiv$. Also by [10] $f_\equiv$ is a folding and a quotient. □

As we are interested in exploring the structure of cycloids from their parameters, symmetries are of importance. The first symmetry will be used in later sections, namely that the structure is preserved by exchanging α with β, and γ with δ, respectively. This symmetry describes the exchange of forward by backward lines, which are terms used by Petri. If ordinary time and space coordinates are considered (see Fig. 1 and [7]) it describes the reversal of space orientation. The second and third symmetry are a kind of shearing of the cycloid. They have an interesting application in Sect. 6.

Theorem 5. *The following cycloids are isomorphic to* $\mathcal{C}(\alpha, \beta, \gamma, \delta)$*:*

(a) $\mathcal{C}(\beta, \alpha, \delta, \gamma)$,
(b) $\mathcal{C}(\alpha, \beta, \gamma - \alpha, \delta + \beta)$ *if* $\gamma > \alpha$ *and*
(c) $\mathcal{C}(\alpha, \beta, \gamma + \alpha, \delta - \beta)$ *if* $\delta > \beta$.

Proof. (a) We show that the mapping $\varphi : X_1 \to X_1$ defined by $\varphi(x_{\xi,\eta}) := x_{\eta+\beta,\xi-\alpha}$ is an isomorphism that is congruent to $\equiv$. To avoid confusion we denote the second cycloid by $\mathcal{C}'(\alpha', \beta', \gamma', \delta')$, i.e. $\alpha' = \beta$, $\beta' = \alpha$, $\gamma' = \delta$, $\delta' = \gamma$.

φ is an isomorphism on the Petri space since $\varphi(x_{\xi,\eta}) = \varphi(x_{\xi',\eta'}) \Rightarrow \eta + \beta = \eta' + \beta \wedge \xi - \alpha = \xi' - \alpha \Rightarrow (\xi, \eta) = (\xi', \eta')$, i.e. φ is injective and by similar arguments also surjective.

It remains to show, that φ is congruent, i.e. $x_{\xi,\eta} \equiv x_{\xi_1,\eta_1} \Rightarrow \varphi(x_{\xi,\eta}) \equiv \varphi(x_{\xi_1,\eta_1})$. For easier reading, we omit the letter x and calculate with the indices only: with $\varphi(\xi, \eta) = (\xi', \eta')$ and $\varphi(\xi_1, \eta_1) = (\xi_1', \eta_1')$ we now prove $(\xi, \eta) \equiv (\xi_1, \eta_1) \Rightarrow (\xi', \eta') \equiv (\xi_1', \eta_1')$. By the definition of $\equiv$ we have $(\xi_1, \eta_1) = (\xi + m\alpha + n\gamma, \eta - m\beta + n\delta)$ for some $m, n \in \mathbb{Z}$ and

$$\begin{aligned}(\xi_1', \eta_1') &= (\eta - m\beta + n\delta + \beta, \xi + m\alpha + n\gamma - \alpha)\\ &= (\eta + \beta - m\beta + n\delta, \xi - \alpha + m\alpha + n\gamma)\\ &= (\xi' + m'\alpha' + n'\gamma', \eta' - m'\beta' + n'\delta')\end{aligned}$$

with $m' = -m$ and $n' = n$, hence $(\xi_1', \eta_1') \equiv (\xi', \eta')$.

(b) Since $\xi + m\alpha + n(\gamma - \alpha) = \xi + (m - n)\alpha + n\gamma$ and $\eta - m\beta + n(\delta + \beta) = \eta - (m - n)\beta + n\delta$, the equivalence relation folding the Petri space is the same as the original one of Definition 2. The analogous argument proves part (c). □

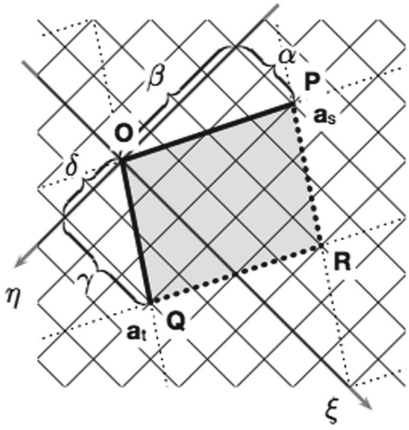


Fig. 5. Fundamental parallelogram

For proving properties of cycloids particular denotations are needed. Petri used the letters O, P, R and Q for the vertices of the fundamental parallelogram ([7], p. 41). Equations, vectors and distances with respect to these vertices will be used (see Fig. 5).

Lemma 6. *For a cycloid $\mathcal{C}(\alpha, \beta, \gamma, \delta)$ the vertices with their coordinates in clockwise order are $O = (0, 0)$ (origin), $P = (\alpha, -\beta)$ (space), $R = (\alpha+\gamma, \delta-\beta)$ (space and time) and $Q = (\gamma, \delta)$ (time). For pairs (A, B) of such nodes the following table gives the equation for the line $\overline{AB}$ through A and B in column 2, the vector $\overrightarrow{AB}$ from A to B in column 3 and the distance $d(A, B)$ between A and B.*

(A, B)	Equation for $\overline{AB}$	Vector $\overrightarrow{AB}$	Distance $d(A, B)$
(O, P)	$\eta = -\frac{\beta}{\alpha}\xi$	$\begin{pmatrix} \alpha \\ -\beta \end{pmatrix} = \mathbf{a}_s$	$\alpha + \beta$
(O, Q)	$\eta = \frac{\delta}{\gamma}\xi$	$\begin{pmatrix} \gamma \\ \delta \end{pmatrix} = \mathbf{a}_t$	$\gamma + \delta$
(P, R)	$\eta = \frac{\delta}{\gamma}(\xi - \alpha) - \beta$	$\begin{pmatrix} \gamma \\ \delta \end{pmatrix}$	$\gamma + \delta$
(Q, R)	$\eta = \frac{-\beta}{\alpha}(\xi - \gamma) + \delta$	$\begin{pmatrix} \alpha \\ -\beta \end{pmatrix}$	$\alpha + \beta$
(P, Q)	$\eta = \frac{\beta+\delta}{\gamma-\alpha}(\xi - \alpha) - \beta$	$\begin{pmatrix} -\alpha+\gamma \\ \beta+\delta \end{pmatrix}$	$\lvert\alpha - \gamma\rvert + \beta + \delta$
(O, R)	$\eta = \frac{\delta-\beta}{\alpha+\gamma}\xi$	$\begin{pmatrix} \alpha+\gamma \\ \delta-\beta \end{pmatrix}$	$\alpha + \gamma + \lvert\beta - \delta\rvert$

Proof. The vectors $\overrightarrow{OP}$ and $\overrightarrow{OQ}$ are obvious (see Fig. 5), while the vector $\overrightarrow{OR}$ is the sum of both. $\overrightarrow{PR}$ equals $\overrightarrow{OQ}$ and $\overrightarrow{PQ} = \overrightarrow{OQ} - \overrightarrow{OP}$. Similarly the equations for the lines $\overline{OP}$ and $\overline{OQ}$ are obvious (see Fig. 5). The equation for $\overline{PR}$ is a shift of value α of $\overline{OQ}$ in ξ-direction and of value $-\beta$ in η-direction. In a similar way $\overline{QR}$ is a shift of $\overline{OP}$. The slope of $\overline{QR}$ and $\overline{OQ}$ follow from $\overline{OP}$ and $\overline{PR}$, respectively. The slope of $\overline{PQ}$ follows from $\overrightarrow{PQ}$ while its η-intercept is computed by ordinary methods. To prove column 4, observe that the distance is different from Euclidean geometrie, as the steps between points of the grid are counted. Therefore $d(\begin{pmatrix} a \\ b \end{pmatrix}, \begin{pmatrix} c \\ d \end{pmatrix}) = |a - c| + |b - d|$. □

The number of transitions of a cycloid is frequently used by Petri and called area A ([7], p. 40). He gave a geometrical proof in his lectures, which was described in [1]. In the proof below we use a property of determinants. This will be useful when considering cycloids in higher dimensions.

Theorem 7

(a) A cycloid $\mathcal{C}(\alpha, \beta, \gamma, \delta)$ has $A = |T| = \alpha\delta + \beta\gamma$ transitions and $|S| = 2\ |T|$ places.

(b) The set T of transitions of such a cycloid is the union of three sets[9], called Upper Area UA, Middle Area MA and Lower Area LA. These sets are:

1. *Upper Area for $0 \leq \xi \leq \min(\alpha, \gamma)$:*

$$UA := \{t_{\xi,\eta} | \xi, \eta \in \mathbb{Z}, -\frac{\beta}{\alpha}\xi \leq \eta \leq \frac{\delta}{\gamma}\xi\}$$

2. *Middle Area 1 for $\gamma \leq \alpha$ and $\gamma \leq \xi \leq \alpha$:*

$$MA_1 := \{t_{\xi,\eta} | \xi, \eta \in \mathbb{Z}, -\frac{\beta}{\alpha}\xi \leq \eta \leq -\frac{\beta}{\alpha}(\xi - \gamma) + \delta\}$$

3. *Middle Area 2 for $\alpha \leq \gamma$ and $\alpha \leq \xi \leq \gamma$:*

$$MA_2 := \{t_{\xi,\eta} | \xi, \eta \in \mathbb{Z}, |\frac{\delta}{\gamma}(\xi - \alpha) - \beta \leq \eta \leq \frac{\delta}{\gamma}\xi\}$$

4. *Lower Area for $\max(\alpha, \gamma) \leq \xi \leq \alpha + \gamma$:*

$$LA := \{t_{\xi,\eta} | \xi, \eta \in \mathbb{Z}, \frac{\delta}{\gamma}(\xi - \alpha) - \beta \leq \eta \leq -\frac{\beta}{\alpha}(\xi - \gamma) + \delta\}$$

Proof

(a) It is well-known that the volume A of a parallelepiped of n vectors is the determinant of the matrix having these vectors as columns. In our case we have two vectors in two dimensions: $A = det(\overrightarrow{OP}\ \overrightarrow{OQ}) = \begin{vmatrix} \alpha & \gamma \\ -\beta & \delta \end{vmatrix} = \alpha\delta + \beta\gamma$. For each transition $t_{\xi,\eta}$ there are two places $s^{\rightarrow}_{\xi,\eta}$ and $s^{\leftarrow}_{\xi,\eta}$, hence $|S| = 2\ |T|$.

[9] These sets are not disjoint.

(b) The Upper Area is a triangle bordered by the lines $\overline{OP}$, $\overline{OQ}$ and $\xi = min(\alpha, \gamma)$ (see Fig. 5 and the table of Lemma 6). The Middle Area depends on the value of $\alpha \leq \gamma$, but the proof also applies. The same holds for the Lower Area. The special case $\alpha = \gamma$ is covered by all four cases. □

4 Cycloid Systems

As in most Petri net models, a net structure together with an initial marking is called a net system as it gives rise to dynamic processes. The first formal definition of an initial marking for cycloids is given by Kummer [3,4]. For the cycloid $\mathcal{C}(4,4,2,2)$ he gives three live initial markings having different reachability sets. This shows that the right choice of an initial marking is important. In the same year, but not known by Kummer, Petri published the following informal definition ([7], p. 38): We provide each cycloid with a *standard marking* by marking the earliest case in the fundamental parallelogram. In his earlier lectures[10] Petri gave an interpretation: shift the space repetition vector $\overrightarrow{OP} = \mathbf{a}_s$ (see Fig. 5) in the direction of the time repetition vector $\overrightarrow{OQ} = \mathbf{a}_t$ until it does not meet any crossing point (i.e. transitions) of the Petri space grid. The edges of the grid which are crossed in this way are the locations of the intended initial marking. To formalize this approach, we select transitions lying between the line $\overline{OP}$ (having equation $\eta = -\frac{\beta}{\alpha}\xi$) and the line $\eta = -\frac{\beta}{\alpha}(\xi + 1)$. The latter results from $\overline{OP}$ by a shift of distance 1 in negative ξ-direction. For each such transition $t_{\xi,\eta}$ a token is located in its forward place $s^{\rightarrow}_{\xi,\eta}$. In a similar way, tokens are introduced in backward places of transitions between lines $\eta = -\frac{\beta}{\alpha}\xi$ and the line $\eta = -\frac{\beta}{\alpha}\xi - 1$, i.e. the line shifted by 1 in negative η-direction. Between any two of such marked places there is no directed path in the Petri grid. Therefore these marked places are causally independent[11], as required for a marking. The definition of Fenske and Kummer are very similar and generate the same reachability set. From the difference it becomes apparent that Petri's informal definition is not unambiguous. In Petri's handwritten script [6] we found the cycloid $\mathcal{C}(4,3,4,3)$ (see Fig. 6), which is used for illustration. Also in Fig. 6 the corresponding cycloid system is shown in two different representations, together with their standard initial marking.

Definition 8. *For a cycloid $\mathcal{C}(\alpha,\beta,\gamma,\delta)$ constructed as a quotient from the Petri space $\mathcal{PS}_1 = (S_1, T_1, F_1)$ by the equivalence relation $\equiv$ we define a cycloid system by adding the following* standard initial marking[12]

$$M_0 = \{s^{\rightarrow}_{\xi,\eta} \in S_1^{\rightarrow} \mid \eta \leq -\tfrac{\beta}{\alpha}\xi \ \wedge \ \eta > -\tfrac{\beta}{\alpha}(\xi+1)\}\,/_{\equiv} \ \cup \tag{1}$$

$$\{s^{\leftarrow}_{\xi,\eta} \in S_1^{\leftarrow} \mid \eta \leq -\tfrac{\beta}{\alpha}\xi \ \wedge \ \eta > -\tfrac{\beta}{\alpha}\xi - 1\}\,/_{\equiv} \tag{2}$$

[10] Reported by Fenske.
[11] They are in the concurrency relation **co**.
[12] We will use the term *initial marking*, for short.

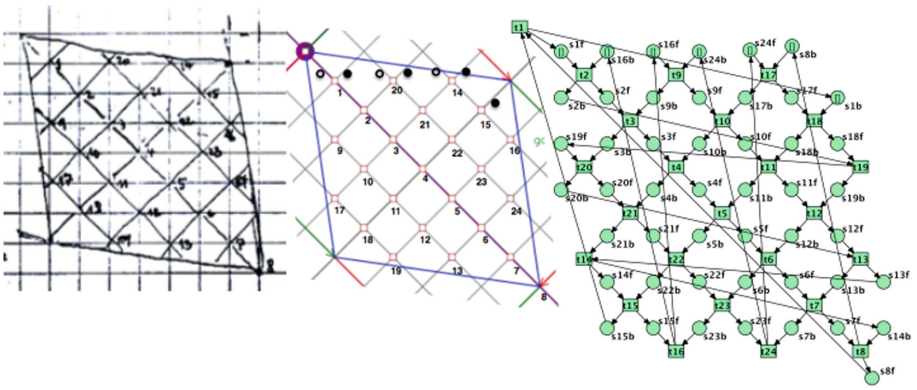


Fig. 6. Cycloid system $\mathcal{C}(4,3,4,3)$ from [6] and redesign.

A proof that a cycloid system is strongly connected, live, safe and secure is given in [1], drafting upon propositions by Stehr [12][13]. The next lemma shows properties that are useful for designing the initial marking and are used in the following proofs. Parts (a) and (b) of the lemma show that the coordinates of the first and the last[14] marked transitions $t_{1,0}$ and $t_{\alpha,1-\beta}$ are the same in all cycloids with $\alpha \geq \beta$. (c) proves a more general regularity, namely that the backward input places of all marked transitions are marked. Part (d) is useful to show that some of these transitions are semi-active, and (e) and (f) will be used to prove that coordinates of a marked transition fulfil a certain condition.

Lemma 9. *Let $\mathcal{C}(\alpha,\beta,\gamma,\delta)$ be a cycloid system with $\alpha \geq \beta$ and initial marking M_0. (See Figs. 4 and 7 for the following properties.)*

(a) $t_{1,0}$ is active.
(b) $t_{\alpha,1-\beta}$ is active if $\alpha = \beta$, but is semi-active if $\alpha > \beta$.
(c) The backward input place is marked for all marked transitions.
(d) If $s^{\leftarrow}_{\xi,\eta-1} \in M_0$ then $s^{\rightarrow}_{\xi,\eta} \notin M_0$.
(e) If $t_{\xi,\eta}$ is a marked transition, then $t_{\xi,\eta-1}$ is not.
(f) If $t_{\xi-1,\eta+1}$ and $t_{\xi,\eta}$ are marked transitions, then $s^{\rightarrow}_{\xi-1,\eta} \in M_0$.

Proof

(a) The coordinates of the input places of $t_{1,0}$ are $(0,0)$ and $(1,-1)$ and therefore satisfy the conditions (1) and (2) of Definition 8 due to $\alpha \geq \beta$.
(b) The coordinates of the backward input place of $t_{\alpha,1-\beta}$ are $(\alpha,-\beta)$ and satisfy the condition (2). Condition (1) for the forward input place $s^{\rightarrow}_{\alpha-1,1-\beta}$ is equivalent to $\beta \geq \beta+1-\frac{\beta}{\alpha} \wedge \beta < \beta+1$ and requires $\alpha = \beta$ (as we assumed $\alpha \geq \beta$).

[13] For the definitions of *safe* and *secure* see [7], whereas *live* was used in the usual form (e.g. see [2], p. 59). For a cycloid to be secure $\alpha,\beta,\gamma,\delta \geq 2$ is required.
[14] First and last with respect to the space dimension.

(c) As for a marked transition $t_{\xi,\eta}$ at least one input place is marked in M_0, it is sufficient to show that the backward input place is marked if the forward input place is marked: if $s^{\rightarrow}_{\xi-1,\eta} \in M_0$ then $s^{\leftarrow}_{\xi,\eta-1} \in M_0$. In fact, from the first part of condition (1) for $s^{\rightarrow}_{\xi-1,\eta}$, namely $\eta \leq -\frac{\beta}{\alpha}(\xi-1)$ and using $\alpha \geq \beta$ it follows $\eta - 1 \leq -\frac{\beta}{\alpha}\xi$ which is the first part of condition (2) for $s^{\leftarrow}_{\xi,\eta-1}$. The same holds for the second part of the conditions.

(d) If $s^{\rightarrow}_{\xi,\eta} \in M_0$ then (by condition (1)) $\eta \leq -\frac{\beta}{\alpha}\xi$ which is in contradiction to the second part of condition (2) for $s^{\leftarrow}_{\xi,\eta-1}$, hence $s^{\rightarrow}_{\xi,\eta} \notin M_0$.

(e) The following property has to be proved: if one of the two (or both) input places $s^{\rightarrow}_{\xi-1,\eta}$ or $s^{\leftarrow}_{\xi,\eta-1}$ of $t_{\xi,\eta}$ are marked then by (c) the backward input place $s^{\leftarrow}_{\xi,\eta-2}$ of $t_{\xi,\eta-1}$ is not marked. Indeed, let be $s^{\rightarrow}_{\xi-1,\eta} \in M_0$, i.e. $\eta \leq -\frac{\beta}{\alpha}(\xi-1) \wedge \eta > -\frac{\beta}{\alpha}\xi$ implying $\eta \leq -\frac{\beta}{\alpha}\xi + \frac{\beta}{\alpha} \leq -\frac{\beta}{\alpha}\xi + 1$ (condition (*)). Assuming $s^{\leftarrow}_{\xi,\eta-2}$ be marked leads to $\eta - 2 > -\frac{\beta}{\alpha}\xi - 1$ and $\eta > -\frac{\beta}{\alpha}\xi + 1$ in contradiction to condition (*). The second case for $s^{\leftarrow}_{\xi,\eta-1}$ is similar.

(f) If $t_{\xi-1,\eta+1}$ is a marked transition, then by (c) $s^{\leftarrow}_{\xi-1,\eta} \in M_0$ (see Fig. 4) and by the first part of condition (2) it follows $\eta \leq -\frac{\beta}{\alpha}(\xi-1)$. This is the same condition (2) for $s^{\rightarrow}_{\xi-1,\eta}$. If $t_{\xi,\eta}$ is a marked transition, then by (c) $s^{\leftarrow}_{\xi,\eta-1} \in M_0$ and by the second part of condition (2) it follows $\eta - 1 > -\frac{\beta}{\alpha}\xi - 1$. This is equivalent to the same condition (2) for $s^{\rightarrow}_{\xi-1,\eta}$. □

As we are interested to find the cycloid parameters from the cycloid system in any representation we introduce new parameters μ, μ_a, μ_0 and τ, τ_a, τ_0 corresponding to the initial marking and the initially active transitions, respectively.

Definition 10. *For a cycloid system $\mathcal{C}(\alpha,\beta,\gamma,\delta)$ with initial marking M_0 the following system parameters are defined:*

(a) $\tau_0 := |\{t| \ |{}^\bullet t \cap M_0| \geq 1\}|$ *is the number of transitions initially marked.*
(b) $\tau_a := |\{t| \ |{}^\bullet t \cap M_0| = 2\}|$ *is the number of initially active transitions.*
(c) $\tau := |\{t| \ |{}^\bullet t \cap M_0| = 1\}|$ *is the number of initially semi-active transitions.*
(d) $\mu_0 := |M_0|$ *is the number of tokens in the initial marking.*
(e) μ_a *is number of tokens activating initially active transitions.*
(f) μ *is number of tokens activating initially semi-active transitions.*

Tokens contributing to μ_a and μ are called active and semi-active tokens, respectively.

Lemma 11. *The number of tokens of the initial marking M_0 is $\mu_0 = \alpha + \beta$.*

Proof. The transitions generating the forward tokens in Definition 8 have the number of β coordinates $(0,0), (\xi_1, 1), \cdots, (\xi_{\beta-1}, \beta-1)$, while those generating the backwards tokens have α coordinates $(1,\eta_1), (2,\eta_2) \cdots, (\alpha,\eta_\alpha)$. The total number is $\alpha + \beta$. □

Petri [6] called the tokens of the former and latter class *forward* and *backwards flow tokens*, respectively. Their total number corresponds to the distance $d(O,P) = \alpha + \beta$ (see Lemma 6).

Lemma 12. *The number of semi-active tokens of the initial marking* M_0 *is* $\mu = |\alpha - \beta|$.

Proof. First, we assume $\alpha \geq \beta$. We define a path, called *m-path*, containing all marked transitions as nodes. The edges connect marked transitions $t_{\xi,\eta}$ and $t_{\xi+1,\eta'}$ with $1 \leq \xi \leq \alpha$.[15] By Lemma 9(a) and (b) the first transition is $t_{1,0}$ and the last is $t_{\alpha,1-\beta}$. From this we have for the η-coordinates $0 \leq \eta \leq 1-\beta$, with the following property.

The edges of the m-path are composed of edges of (vector-)type $(1,-1)$ (diagonal in the ξ-η-grid) and of type $(1,0)$ (following a line $\eta = const$), since the type $(0,-1)$ is excluded by Lemma 9(e): if $t_{\xi,\eta}$ is a marked transition in M_0 then $t_{\xi,\eta-1}$ is not. By (c) and (d) of the same lemma the backward input place of each marked transition is marked, but the forward output place is not. Therefore a (place on a) $(1,0)$-edge is not marked and the transition at the higher-ξ-end of this edge is semi-active. Furthermore the forward input place of a transition at the end of a $(1,-1)$-edge is marked by Lemma 9(f). Therefore this transition is active. As a consequence a token is semi-active if and only if it marks the place on a $(1,0)$-type edge and the number of semi-active tokens equals the number of $(1,0)$-edges on the m-path.

Both types of edges, $(1,0)$ and $(1,-1)$ have the number 1 in their ξ-component. Therefore the number of all edges of the m-path is the difference of the ξ-components of the first transition $t_{1,0}$ and the last transition $t_{\alpha,1-\beta}$, namely $|1-\alpha|$. With respect to the η-component only $(1,-1)$ contribute. Therefore their number is the difference of the η-components of the first and the last transition, namely $|0-(1-\beta)|$. The number of $(1,0)$-edges is their difference $|1-\alpha| - |0-(1-\beta)| = |1-\alpha| - |\beta-1| = |\alpha-\beta|$. This proves the number of semi-active tokens to be $|\alpha-\beta|$. To revoke the assumption $\alpha \geq \beta$ we apply Theorem 5(a) by swapping α and β keeping the structure isomorphic. □

Example 13. To illustrate the proof, consider Fig. 7. It shows the case $\alpha = \beta = 6$. There are 6 active transitions with coordinates $(1,0), (2,-1), \cdots, (6,-5)$ forming a m-path. The case $\alpha = 6$ and $\beta' = 3$ is represented in the same picture. The new origin O' is in the point $(0, -\alpha + \beta') = (0,-3)$. The coordinates with respect to the new origin are distinguished by apostrophes. The nodes of the m-path have the coordinates $(1',0'), (2',0'), (3',-1'), (4',-1'), (5',-2'), (6',-2')$. The path contains two $(1,1)$-edges and three $(1,0)$-edges. It is evident that the number of $(1,0)$-edges equals the distance of the two origins 0 and $0'$, which is $\alpha - \beta' = 3$. The initial marking is shown by small circles on edges, hence 3 transitions are active and 3 are semi-active. To show a case with two consecutive $(1,1)$-edges, consider the case $\alpha = 6$ and $\beta'' = 4$.

[15] It may be helpful for the reader to consider Fig. 7, which illustrates the proof and is explained afterwards.

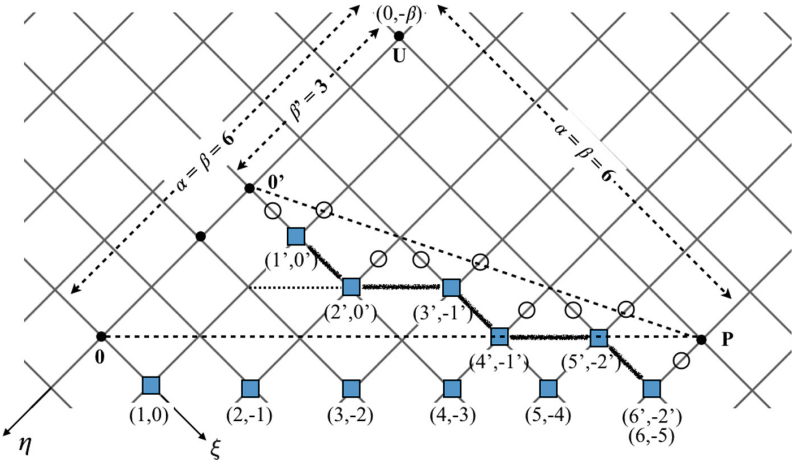


Fig. 7. Marked paths for the cases $\alpha = \beta = 6$ and $\alpha = 6$, $\beta' = 3$

Theorem 14. *Let $\mathcal{C}(\alpha, \beta, \gamma, \delta)$ be a cycloid system with initial marking M_0*

(a) $\tau_0 = max\{\alpha, \beta\}$
(b) $\tau_a = min\{\alpha, \beta\}$
(c) $\tau = |\alpha - \beta|$
(d) $\mu_0 = \alpha + \beta$
(e) $\mu_a = 2 \cdot min\{\alpha, \beta\}$
(f) $\mu = |\alpha - \beta|$.

Proof. The propositions of (d) and (f) are proved in the preceding lemmata.

For (e) we use $\mu_a = \mu_0 - \mu = \alpha + \beta - |\alpha - \beta|$. For $\alpha \geq \beta$ this gives $\alpha+\beta-(\alpha-\beta) = 2\cdot\beta$. In the same way, for $\alpha < \beta$ we obtain $\alpha+\beta-(\beta-\alpha) = 2\cdot\alpha$, hence $\mu_a = 2 \cdot min\{\alpha, \beta\}$. The first three propositions follow immediately: There are as many semi-active transitions as semi-active tokens: $\tau = \mu = |\alpha-\beta|$. Each active transition has two active input tokens: $\tau_a = \frac{1}{2} \cdot \mu_a = min\{\alpha, \beta\}$. Finally, $\tau_0 = \tau_a + \tau = min\{\alpha, \beta\} + |\alpha - \beta|$, which is $\beta + \alpha - \beta = \alpha$ if $\alpha \geq \beta$ and $\alpha + \beta - \alpha = \beta$ if $\alpha < \beta$. □

In his article [7] Petri introduces the notion of *slowness* by $w = \frac{|\alpha-\beta|}{\alpha+\beta}$. Using our notation, the slowness of a cycloid is the ratio μ/μ_0 of semi-active tokens relative to all tokens in the initial marking. The more tokens are semi-active, the higher is the slowness. In the left-hand example of Fig. 2 we have minimal slowness $w = 0$ as all tokens are active, while on the right-hand side $w = \frac{1}{3}$ as a third of all tokens is semi-active. While Petri's definition of slowness is non-probabilistic, in a personal communication he wrote: "Slowness is a mass or group phenomenon. The formula is valid for very large numbers of objects following a uniformly distributed behaviour. By a simple rule-of-thumb, a distribution proceeds as its slowest part".

5 Minimal Cycloid Cycles

As another parameter which is independent of a fundamental parallelogram representation, we now look for the length of minimal cycles. In his lecture notes [6] Petri mentions the "number of segments in local basic circuit" as $\gamma + \delta$ in connection with his investigations on security. As we will see in this section, this is in fact the length of a minimal cycloid circuit in some cases, but no further results of Petri are known on the topic. As we investigate the property of minimal cycles using the fundamental parallelogram, we first state that such a cycle always appears as a normal form containing a vertex O, P, Q or R.

Lemma 15. *For any cycloid $\mathcal{C}(\alpha, \beta, \gamma, \delta)$ there is a minimal cycle containing the origin O in its fundamental parallelogram representation.*

Proof. It is obvious that a cycloid contains cycles (closed paths). Consider a fixed minimal cycle and a transition t_{ξ_0,η_0} contained in this cycle. The mapping $\varphi : X_1 \to X_1$ defined by $\varphi(x_{\xi,\eta}) := x_{\xi-\xi_0,\eta-\eta_0}$ is an automorphism that is congruent to $\equiv$. Therefore there is a also a minimal cycle containing $t_{\xi,\eta}$ which is the origin $O = (0, 0)$ in the Petri space. □

As shown in Fig. 8, the edges of a path can leave the limiting lines of the fundamental parallelogram. We first consider the opposite case.

Definition 16. *Let $\mathcal{C}(\alpha, \beta, \gamma, \delta)$ be a cycloid. A path or a cycle is called* internal *if it does not leave the limiting lines of the fundamental parallelogram. The internal minimal cycle index is defined by*

$$
\begin{aligned}
i_0(\alpha,\beta,\gamma,\delta) \;=\; & \textbf{if } \alpha \le \beta \textbf{ then} \\
& \quad \textbf{if } \beta \le \delta \textbf{ then } 1 \textbf{ else } 0 \textbf{ fi} \\
& \textbf{else} \\
& \quad \textbf{if } \alpha \le \gamma \textbf{ then } -1 \textbf{ else } 0 \textbf{ fi} \\
& \textbf{fi}
\end{aligned}
$$

If the parameters are given by context, the internal index is denoted by i_0.

Theorem 17. *The length of a minimal internal cycle of a cycloid $\mathcal{C}(\alpha, \beta, \gamma, \delta)$ is $cyc_0(\alpha, \beta, \gamma, \delta) = \gamma + \delta + i_0 \cdot (\alpha - \beta)$.*

Proof. As we argued above, with respect to paths and cycles in the fundamental parallelogram and by Lemma 15, it is sufficient to consider the paths between the vertices O, P, Q and R. Therefore we can reduce our investigations to the paths (A, B), as given in the second column of the following table. The minimal paths between (O, Q) and (P, R) are the same and have the length of the distance $d(O, Q) = d(P, R) = \gamma+\delta$ by Lemma 6. Such a path is always possible, as already observed by Petri, in contrast to paths between O and P or Q and R. For any such internal path, due to $\overrightarrow{OP} = \overrightarrow{QR} = (\alpha, -\beta)$ a step in negative η-direction would be included, which is impossible.

Case	(A, B)	Path possible	Path length	i_0
I	$(O, Q), (P, R)$	True	$\gamma + \delta$	0
II	(P, Q)	$\alpha \leq \gamma$	$-\alpha + \beta + \gamma + \delta$	-1
III	(O, R)	$\beta \leq \delta$	$\alpha - \beta + \gamma + \delta$	1
	$(O, P), (Q, R)$	False		

The paths between (P, Q) and (O, R) are only possible under specific conditions, however. (P, Q) is possible if the ξ-coordinate of P is not greater than that of Q, i.e. $\alpha \leq \gamma$. (O, R) is possible if and only if $R = (\alpha + \gamma, \delta - \beta)$ (Lemma 6) has a non-negative η-coordinate, i.e. $\delta \geq \beta$. Due to these conditions the signs for the absolute value in Lemma 6 for $d(P, Q) = |\alpha - \gamma| + \beta + \delta = \gamma - \alpha + \beta + \delta$ and $d(O, R) = \alpha + \gamma + |\beta - \delta| = \alpha + \gamma + \delta - \beta$ can be omitted (column 4). With the values for i_0 in the fifth column of the preceding table, the formula $cyc_0(\alpha, \beta, \gamma, \delta) = \gamma + \delta + i_0 \cdot (\alpha - \beta)$ reproduces the values in the fourth column. It remains to find out which of the three cases I, II and III gives the length of the minimal cycle. Let us compare, for instance, case I with case II. $\gamma + \delta \leq -\alpha + \beta + \gamma + \delta$ is equivalent to $\alpha \leq \beta$, which gives the entry of the line I and column II of the following table. Similarly, comparing case II with case III gives $-\alpha + \beta + \gamma + \delta \leq \alpha - \beta + \gamma + \delta \Leftrightarrow \beta \leq \alpha$. The other entries in the table are computed in the same way.

Case	I	II	III
I		$\alpha \leq \beta$	$\beta \leq \alpha$
II	$\beta \leq \alpha$		$\beta \leq \alpha$
III	$\alpha \leq \beta$	$\alpha \leq \beta$	

Compiling these results, in the case of $\alpha \leq \beta$ we obtain III $\leq$ I $\leq$ II, where the case identifier stands for the cycle length. Case III is the winner if it is possible, i.e. under the condition $\alpha \leq \beta \ \wedge \ \beta \leq \delta$ with cycle length $\alpha - \beta + \gamma + \delta$. This is just the right condition, namely giving $i_0 = 1$ in Definition 16 and the correct cycle length in Theorem 17. If case III is not possible, we obtain case I as the minimal cycle length with condition $\alpha \leq \beta \ \wedge \ \beta > \delta$ and $i_0 = 0$ which gives again the correct cycle length $\gamma + \delta$ in Theorem 17.

The case $\beta \leq \alpha$ is deduced from the case $\alpha \leq \beta$ by using the isomorphism of Theorem 5 applying the substitution $[\alpha \to \beta, \beta \to \alpha, \gamma \to \delta, \delta \to \gamma]$. □

As shown by the cycle $t_1, t_2, t_3, t_4, t_5, t_6$ of length 6 in the cycloid at the left-hand side of Fig. 8, a minimal cycle is not necessarily internal. Looking on the corresponding fundamental parallelogram on the right-hand side, the cycle can be obtained by starting in the origin, proceeding in ξ-direction until meeting the line $\overline{QR_1}$. The example is chosen in such a way that the ξ-axis and the line $\overline{QR}$ intersect in a point in the Petri space, namely the point R_3 in Fig. 8,

which is equivalent to R_1. Geometrically, the point is obtained by unfolding[16] the fundamental parallelogram $i - 1 = 2$ times. As shown in the proof below we will obtain $i = \frac{\delta}{\beta} = 3$. The vector $\overrightarrow{OR_3}$ can be calculated by adding $\overrightarrow{OQ}$ and i times the vector $\overrightarrow{QR_1} = \overrightarrow{OP_1}$. This vector is $\overrightarrow{OR_3} = \begin{pmatrix} \gamma + i \cdot \alpha \\ \delta - i \cdot \beta \end{pmatrix} = \begin{pmatrix} 6 \\ 0 \end{pmatrix}$ with length $|\overrightarrow{OR_3}| = \gamma + \delta + i \cdot (\alpha - \beta) = 6$. The case where the intersect of the lines is not in the Petri space is treated in the proof.

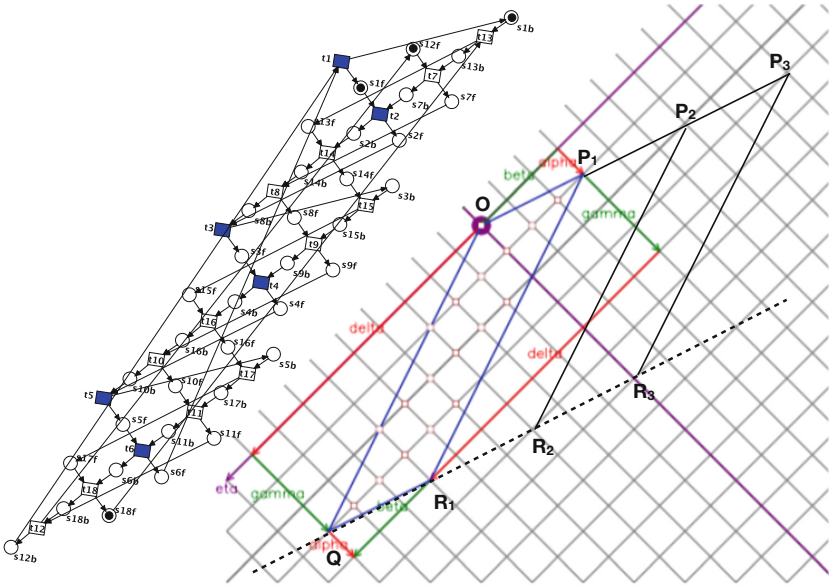


Fig. 8. Cycloid $\mathcal{C}(1, 3, 3, 9)$ as Petri net and in fundamental diagram representation.

Definition 18. *The minimal cycle index of a cycloid* $\mathcal{C}(\alpha, \beta, \gamma, \delta)$ *is defined by*

$$i(\alpha, \beta, \gamma, \delta) = \textbf{if } \alpha \leq \beta \textbf{ then } \lfloor \frac{\delta}{\beta} \rfloor \textbf{ else } - \lfloor \frac{\gamma}{\alpha} \rfloor \textbf{ fi}$$

If the parameters are given by context, the index is denoted by i.

Theorem 19. *The length of a minimal cycle of a cycloid* $\mathcal{C}(\alpha, \beta, \gamma, \delta)$ *is*

$$cyc(\alpha, \beta, \gamma, \delta) = \gamma + \delta + i \cdot (\alpha - \beta).$$

If the parameters are given by context the cycle index is denoted by i.

[16] Such unfoldings of the fundamental parallelogram have been frequently used by Petri, see "Nets, Time and Space" [7], Fig. 11, for instance.

Proof. With respect to paths and cycles in the fundamental parallelogram and by Lemma 15 it is sufficient to consider paths starting in the origin O.

(a) We first consider the case $\alpha \leq \beta$. As will be justified below, the best choice is a line starting in the origin O in direction ξ, i.e. having the equation $\eta = 0$. It meets the line $\overline{QR}$ with equation $\eta = \frac{-\beta}{\alpha}(\xi - \gamma) + \delta$ (see Lemma 6) in a point X. If this point is equivalent to Q or R (with respect to $\equiv$), then $d(O, X) = cyc$ is the length of the cycle in question. Hence, setting $\xi = cyc$ and $\eta = 0$ in the equation above, we obtain: $0 = \frac{-\beta}{\alpha}(cyc - \gamma) + \delta \Leftrightarrow \frac{\beta}{\alpha}(cyc - \gamma) = \delta \Leftrightarrow cyc = \delta\frac{\alpha}{\beta} + \gamma \Leftrightarrow cyc = \delta\frac{\alpha}{\beta} + \gamma + \delta - \delta \Leftrightarrow cyc = \gamma + \delta + \frac{\delta}{\beta}(\alpha - \beta)$. If X is equivalent to Q or R with respect to $\equiv$ then $\frac{\delta}{\beta}$ is an integer. If this is not the case, we compute X in a different way. In fact, X can be obtained by the following vector equation $\overrightarrow{OX} = \overrightarrow{OQ} + j \cdot \overrightarrow{OP} = \begin{pmatrix} \gamma \\ \delta \end{pmatrix} + j \cdot \begin{pmatrix} \alpha \\ -\beta \end{pmatrix} = \begin{pmatrix} \gamma + j \cdot \alpha \\ \delta - j \cdot \beta \end{pmatrix}$ for some integer j. The length of this vector is $|\overrightarrow{OX}| = \gamma + j \cdot \alpha + \delta - j \cdot \beta = \gamma + \delta + j \cdot (\alpha - \beta)$. Comparing $|\overrightarrow{OX}| = \gamma + \delta + j \cdot (\alpha - \beta)$ with $cyc = \gamma + \delta + \frac{\delta}{\beta}(\alpha - \beta)$ from case (a) we conclude $j \leq \frac{\delta}{\beta} \leq j + 1$. Due to the causality structure of the cycloid the transition with respect to $j + 1 = \lceil \frac{\delta}{\beta} \rceil$ is not directly reachable, hence $j = \lfloor \frac{\delta}{\beta} \rfloor$. This gives $cyc = \gamma + \delta + \lfloor \frac{\delta}{\beta} \rfloor \cdot (\alpha - \beta)$ in the case $\alpha \leq \beta$. The optimality of the choice of a line in case a) becomes evident here, since a smaller value of i results in a longer cycle, whereas a greater value of i is impossible due to the causality structure of the cycloid.

(b) For the alternative case we look at the isomorphic cycloid $\mathcal{C}(\beta, \alpha, \delta, \gamma)$ (by interchanging α and β, as well as γ and δ, see Theorem 5 which has a minimal cycle of the same length, hence $cyc = \gamma + \delta + \lfloor \frac{\gamma}{\alpha} \rfloor \cdot (\beta - \alpha)$ also in the case $\alpha > \beta$. Both cases together verify the theorem. □

Remark: Using Petri's terminology and the notion of semi-active transition, the minimal length of a cycle is the length $\gamma + \delta$ of a local basic circuit, possibly decreased by an integer multiple i of the number $\tau = |\alpha - \beta|$ of semi-active transitions.

Note that Theorems 17 and 19 are consistent in the following sense. The internal minimal cycle length is obtained without unfolding the fundamental parallelogram. This means that the index i is 0 or 1. If we replace $\lfloor \frac{\delta}{\beta} \rfloor$ by $min\{1, \lfloor \frac{\delta}{\beta} \rfloor\}$ and $-\lfloor \frac{\gamma}{\alpha} \rfloor$ by $max\{-1, -\lfloor \frac{\gamma}{\alpha} \rfloor\}$ in Definition 18, we obtain Theorem 17 from Theorem 19.

6 Computing Cycloid Parameters from System Parameters

Next we exploit our results to find the fundamental parallelogram representation of a cycloid from its net presentation using the system parameters τ_0, τ_a, A and cyc. The corresponding equivalence is denoted as σ-*equivalence*. The letter σ

emphasizes the use of the shortest cycle length *cyc*, whereas the other parameters are more natural. It also gives room for other equivalences like λ-*equivalence*, where the longest cycle length is used, instead. Similar to the theory of regions, the following procedures do not necessarily give a unique result. But for $\alpha \neq \beta$ the resulting cyloids are isomorphic.

Definition 20. *Cycloids with identical system parameters τ_0, τ_a, A and cyc are called σ-equivalent.*

Theorem 21. *Given a cycloid $\mathcal{C}(\alpha, \beta, \gamma, \delta)$ in its net representation where the parameters τ_0, τ_a, A and cyc are known (but the parameters $\alpha, \beta, \gamma, \delta$ are not). Then a σ-equivalent cycloid $\mathcal{C}(\alpha', \beta', \gamma', \delta')$ can be computed by the formulas $\alpha' = \tau_0$, $\beta' = \tau_a$ and, if $\alpha' \neq \beta'$ the positive solutions of γ' **mod** $\alpha' = \frac{\alpha' \cdot cyc - A}{\alpha' - \beta'}$ and $\delta' = \frac{1}{\alpha'}(A - \beta' \cdot \gamma')$. These equations may result in different cycloids which are isomorphic, however. If $\alpha' = \beta'$ then $\gamma' = \lceil \frac{cyc}{2} \rceil$ and $\delta' = \lfloor \frac{cyc}{2} \rfloor$ can be used.*

Proof. As by Theorem 5 for the case $\alpha \leq \beta$, there is an isomorphic solution for $\alpha \geq \beta$ we can restrict to the latter case. Hence by Theorem 14 we can choose $\alpha' = \tau_0$ and $\beta' = \tau_a$, giving $\alpha' = \alpha$ and $\beta' = \beta$.

For computing γ' and δ' we use the following equations if $\alpha \neq \beta$: from $A = \alpha\delta' + \beta\gamma'$ we obtain $\delta' = \frac{A}{\alpha} - \frac{\beta}{\alpha}\gamma'$ and insert this value into the formula for *cyc* in the case $\alpha \geq \beta$: $cyc = \gamma' + \delta' + \lfloor \frac{\gamma'}{\alpha} \rfloor \cdot (\beta - \alpha) = \gamma' + \frac{A}{\alpha} - \frac{\beta}{\alpha}\gamma' + \lfloor \frac{\gamma'}{\alpha} \rfloor \cdot (\beta - \alpha)$. This is equivalent to $\gamma' - \alpha \cdot \lfloor \frac{\gamma'}{\alpha} \rfloor = \frac{\alpha \cdot cyc - A}{\alpha - \beta}$. Using $\gamma' - \alpha \cdot \lfloor \frac{\gamma'}{\alpha} \rfloor = \gamma'$ **mod** α we obtain one or more (positive) solutions γ', which also give the same number of solutions for $\delta' = \frac{A}{\alpha} - \frac{\beta}{\alpha}\gamma'$. If γ_1 and γ_2 are two different such solutions, we have $\gamma_2 = \gamma_1 + k \cdot \alpha$ for some $k \in \mathbb{Z}$. W.l.o.g. assume $k \geq 1$. Then for δ_2 we obtain $\delta_2 = \frac{A}{\alpha} - \frac{\beta}{\alpha}\gamma_2 = \delta_1 - k \cdot \beta$. Hence applying Theorem 5(c) (k-times) the resulting cycloids $\mathcal{C}(\alpha, \beta, \gamma_1, \delta_1)$ and $\mathcal{C}(\alpha, \beta, \gamma_1 + k \cdot \alpha, \delta_1 - k \cdot \beta)$ are isomorphic.

If $\alpha' = \beta'$ then the minimal cycle length is always $\gamma' + \delta'$. To verify again the same values $cyc' = cyc$ and $A' = A$ in $\mathcal{C}(\alpha', \beta', \gamma', \delta')$, we observe: $cyc' = \gamma' + \delta' = \lceil \frac{cyc}{2} \rceil + \lfloor \frac{cyc}{2} \rfloor = cyc$ and $A' = \alpha\delta' + \beta\gamma' = \alpha(\delta' + \gamma') = \alpha \cdot cyc' = \alpha \cdot cyc = A$. □

Example 22. For the cycloid system on the left-hand side of Fig. 2 we obtain $\alpha' = \tau_0 = 3$, $\beta' = \tau_a = 3$ and $\alpha' = \tau_0 = 4$, $\beta' = \tau_a = 2$ for the right-hand net of the same figure. To compute γ' and δ' we start with the net on the right-hand side with $A = 6$ transitions and $cyc = 2$. As $\alpha' \neq \beta'$ we obtain γ' **mod** $\alpha' = \frac{\alpha' \cdot cyc - A}{\alpha' - \beta'} = \frac{4 \cdot 2 - 6}{4 - 2} = 1$. The set of positive solutions of γ' **mod** $4 = 1$ is $\{4n + 1 | n \in \mathbb{Z}\} = \{1, 5, 9, \cdots\}$. Using these values we obtain $\delta' = \frac{1}{\alpha'}(A - \beta' \cdot \gamma') = \frac{1}{4}(6 - 2 \cdot \gamma') = \{1, -1, -3, \cdots\}$. Therefore $\gamma' = 1, \delta' = 1$ is a unique positive solution. To compute γ' and δ' for the net on the left-hand side ($A = 6$, $cyc = 2$) we obtain $\gamma' = \lceil \frac{2}{2} \rceil = 1$ $\delta' = \lfloor \frac{2}{2} \rfloor = 1$, since $\alpha' = \beta'$.

Example 23. For the cycloid system of Fig. 6 we obtain $\alpha' = \tau_0 = 4$, $\beta' = \tau_a = 3, A = 24, cyc = 6$. As $\alpha' > \beta'$ we consider γ' **mod** $\alpha' = \frac{\alpha' \cdot cyc - A}{\alpha' - \beta'} = \frac{4 \cdot 6 - 24}{4 - 3} = 0$. The set of positive solutions of γ' **mod** $4 = 0$ is $\{4n | n \in \mathbb{Z}\} = \{0, 4, 8, \cdots\}$. Using these values we obtain $\delta' = \frac{1}{\alpha'}(A - \beta' \cdot \gamma') = \frac{1}{4}(24 - 3 \cdot \gamma')$. Hence the set

of values for (γ', δ') reduces to $\{(0, \frac{3}{2}), (4, 3), (0, 8), \cdots\}$. Therefore $\gamma' = 4, \delta' = 3$ is a unique positive integer solution.

Example 24. Consider the cycloid $\mathcal{C}(3, 2, 8, 2)$ with $A = 22$ and $cyc = 8$. Then $\alpha' = 3$, $\beta' = 2$ and γ' **mod** $3 = \frac{\alpha' \cdot cyc - A}{\alpha' - \beta'} = \frac{3 \cdot 8 - 22}{3-2} = 2$ has a set of solutions $\gamma' \in \{\cdots -1, 2, 5, 8, 11, \cdots\}$ and with $\delta' = \frac{1}{\alpha'}(A - \beta' \cdot \gamma') = \frac{1}{3}(22 - 2 \cdot \gamma')$ also $(\gamma', \delta') \in \{\cdots (-1, 8), (2, 6), (5, 4), (8, 2), (11, 0), \cdots\}$. As only positive integer values for γ' and δ' are possible, we obtain two cycloids $\mathcal{C}(3, 2, 2, 6)$ and $\mathcal{C}(3, 2, 5, 4)$ that are σ-equivalent to the initial cycloid $\mathcal{C}(3, 2, 8, 2)$. All three cycloids are isomorphic due to Theorem 5.

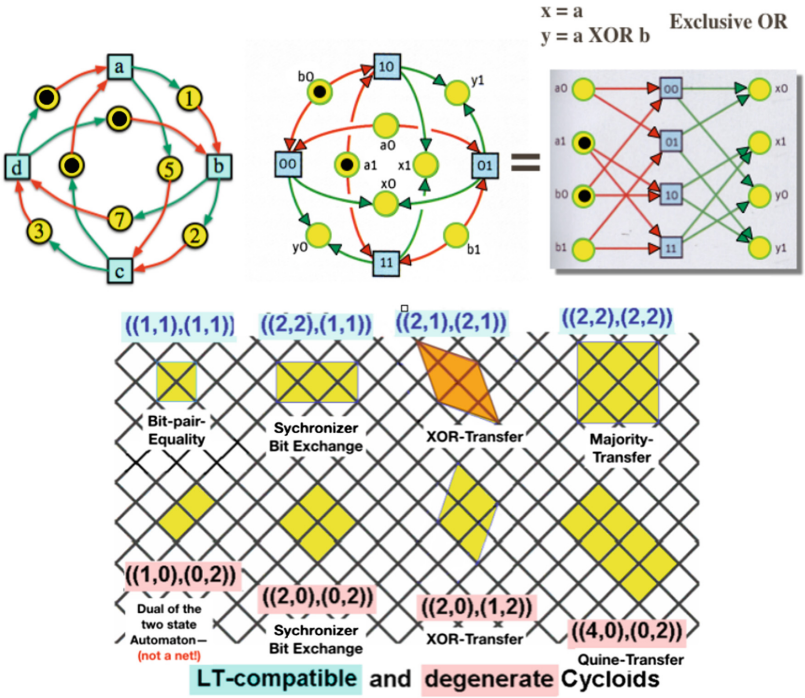


Fig. 9. Cycloid $\mathcal{C}(2, 1, 2, 1)$, XOR-gate and Petri's patterns of behavior

Example 25. As mentioned in the introduction, the topological equivalence of elementary gates of information processing on the bit level to particular cycloids is an important result of Petri's work. In a mail from August 28, 2007 Petri wrote that the "oscillator" is the easiest example for the "surprising" construction of a cycloid from the XOR-gate, which he called a "Pattern of Group Behaviour" based on the same topology. Petri wrote that he first found such an equivalence in the year 1964 by the example of the Quine-Transfer. Let us come back to the XOR-gate shown on the right-hand side of Fig. 9 and its redesign in the middle.

By reorientation of the arcs, Petri built the net on the left hand side. It becomes a cycloid system by adding a token to the place a_0 in order to obtain a live net, as in a marked graph each cycle has to contain exactly one token. The net is known as "oscillator net" or "four seasons net" (equivalent to Fig. 3).

Here our result helps Petri's construction: initially marked transitions: $\alpha' = \tau_0 = |\{a, b\}| = 2$, initially active transitions $\beta' = \tau_a = |\{a\}| = 1$, minimal cycle length $cyc = 2$ and $A = 4$, hence $\gamma' \mathbf{mod}\ 2 = \frac{\alpha' \cdot cyc - A}{\alpha' - \beta'} = \frac{2 \cdot 2 - 4}{2 - 1} = 0$ and $\delta' = \frac{1}{\alpha'}(A - \beta' \cdot \gamma') = 2 - \frac{\gamma'}{2}$. The set of solutions for (γ', δ') is $\{\cdots (0, 2), (2, 1), (4, 0), \cdots\}$. Therefore $\gamma' = 2$ and $\delta' = 1$ is a unique positive integer solution, giving the cycloid $\mathcal{C}(2, 1, 2, 1)$, as shown in Petri's slide of Fig. 9.

7 Conclusion

In this paper a formal definition for the model of cycloids is given, enabling proofs of known and new properties. This has been done on the basis of C. A. Petri's formal and informal papers, some of which are internal for the Hamburg research group on General Net Theory. To prepare new results and also as a mathematical tool box for further research, some technical properties are derived. The main theorem gives a compact formula for the length of minimal cycloid cycles. Together with three other system parameters, this allowed to compute the cycloid parameters α, β, γ and δ solely from the cycloid net. At the end, some examples are given, including some suggested by C. A. Petri himself.

References

1. Fenske, U.: Petris Zykloide und Überlegungen zur Verallgemeinerung. Diploma thesis (2008)
2. Girault, C., Valk, R. (eds.): Petri Nets for System Engineering - A Guide to Modelling, Verification and Applications. Springer, Berlin (2003). https://doi.org/10.1007/978-3-662-05324-9. 585 p.
3. Kummer, O.: Axiomatic Systems in Concurrency Theory. Logos Verlag Berlin, Berlin (2001)
4. Kummer, O., Stehr, M.-O.: Petri's axioms of concurrency a selection of recent results. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 195–214. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63139-9_37
5. Langner, P.: Cycloids' Characteristics (2013). http://cycloids.adventas.de/
6. Petri, C.A.: Collection of Handwritten Scripts on Cycloids. Lecture University of Hamburg (1990/1991)
7. Petri, C.A.: Nets, time and space. Theor. Comput. Sci. **153**, 3–48 (1996)
8. Petri, C.A.: Slides of the Lecture "Systematik der Netzmodellierung", Hamburg (2004). http://www.informatik.uni-hamburg.de/TGI/mitarbeiter/profs/petri_eng.html
9. Petri, C.A., Valk, R.: On the Physical Basics of Information Flow - Results obtained in cooperation Konrad Zuse. http://www.informatik.uni-hamburg.de/TGI/mitarbeiter/profs/petri_eng.html (2008)

10. Smith, E., Reisig, W.: The semantics of a net is a net - an exercise in general net theory. In: Voss, K., Genrich, J., Rozenberg, G. (eds.) Concurrency and Nets, pp. 461–479. Springer, Berlin (1987). https://doi.org/10.1007/978-3-642-72822-8_29
11. Stehr, M.O.: System specification by cyclic causality constraints. Technical report 210, Department of Informatics, University Hamburg (1998)
12. Stehr, M.O.: Characterizing security in synchronization graphs. Petri Net Newsl. **56**, 17–26 (1999)

Markings in Perpetual Free-Choice Nets Are Fully Characterized by Their Enabled Transitions

Wil M. P. van der Aalst(✉)

Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany
wvdaalst@pads.rwth-aachen.de

Abstract. A marked Petri net is *lucent* if there are no two different reachable markings enabling the same set of transitions, i.e., states are fully characterized by the transitions they enable. This paper explores the class of marked Petri nets that are lucent and proves that *perpetual marked free-choice nets* are lucent. Perpetual free-choice nets are free-choice Petri nets that are live and bounded and have a home cluster, i.e., there is a cluster such that from any reachable state there is a reachable state marking the places of this cluster. A home cluster in a perpetual net serves as a "regeneration point" of the process, e.g., to start a new process instance (case, job, cycle, etc.). Many "well-behaved" process models fall into this class. For example, the class of short-circuited sound workflow nets is perpetual. Also, the class of processes satisfying the conditions of the α algorithm for process discovery falls into this category. This paper shows that the states in a perpetual marked free-choice net are fully characterized by the transitions they enable, i.e., these process models are lucent. Having a one-to-one correspondence between the actions that can happen and the state of the process, is valuable in a variety of application domains. The full characterization of markings in terms of enabled transitions makes perpetual free-choice nets interesting for workflow analysis and process mining. In fact, we anticipate new verification, process discovery, and conformance checking techniques for the subclasses identified.

1 Introduction

Structure theory is a branch in Petri nets [8,20–23] that asks what behavioral properties can be derived from it structural properties [10,12,13]. Many different subclasses have been studied. Examples include state machines, marked graphs, free-choice nets, asymmetric choice nets, and nets without TP and PT handles. Structure theory also studies local structures such as traps and siphons that may reveal information about the behavior of the Petri net and includes linear algebraic characterizations of behavior involving the matrix equation or invariants [12,13,20].

In this paper, we focus on the following fairly general question: *What is the class of Petri nets for which each marking is uniquely identified by the set of*

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 315–336, 2018.
https://doi.org/10.1007/978-3-319-91268-4_16

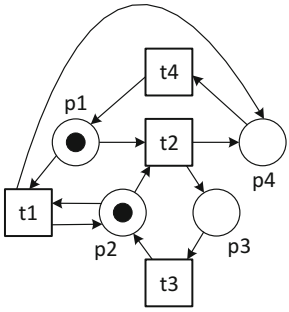


Fig. 1. A perpetual marked free-choice net (i.e., live, bounded, and having a home cluster) that is lucent (each reachable marking is unique in terms of the transitions it enables).

enabled transitions? We call such nets *lucent*. A lucent marked Petri net cannot have two different reachable markings that enable the same set of transitions.

Consider, for example, the Petri net shown in Fig. 1. There are four reachable markings. Marking $[p1, p2]$ enables $\{t1, t2\}$. Marking $[p1, p3]$ enables $\{t3\}$. Marking $[p2, p4]$ enables $\{t4\}$. Marking $[p3, p4]$ enables $\{t3, t4\}$. Hence, the marked net is lucent, because each of the four markings is uniquely identified by a particular set of enabled transitions. The Petri net shown in Fig. 2 is not lucent. After firing either transition $t1$ or $t2$ only $t3$ is enabled, i.e., the two corresponding $[p2, p5]$ and $[p2, p6]$ markings enable the same set of transitions. The choice between $t4$ and $t5$ is controlled by a token in $p5$ or $p6$, and this state information is not "visible" when only $t3$ is enabled. As illustrated by Fig. 2, it is easy to construct non-free-choice nets that are not lucent. Moreover, unbounded Petri nets cannot be lucent. These examples trigger the question: Which classes of marked Petri nets are guaranteed to be lucent?

In this paper, we will show that *perpetual marked free-choice nets* are always lucent. These nets are live and bounded and also have a so-called *home cluster*. A home cluster serves as a "regeneration point", i.e., a state where all tokens mark a single cluster. The property does not hold in general. Liveness, boundedness, the existence of a home cluster, and the free-choice requirement are all needed. We will provide various counterexamples illustrating that dropping one of the requirements is not possible.

Free-choice nets are well studied [11,12,15,25]. The definite book on the structure theory of free-choice nets is [13]. Also, see [12] for pointers to literature. Therefore, it is surprising that the question whether markings are uniquely identified by the set of enabled transitions (i.e., lucency) has not been explored in literature. Most related to the results presented in this paper is the work on the so-called *blocking theorem* [17,26]. Blocking markings are reachable markings which enable transitions from only a single cluster. Removing the cluster yields a dead marking. Figure 1 has three blocking markings ($[p1, p2]$, $[p1, p3]$, and $[p2, p4]$). The blocking theorem states that in a bounded and live free-choice net each cluster has a unique blocking marking. We will use this result, but prove a much more general property. *Note that we do not look at a single cluster and*

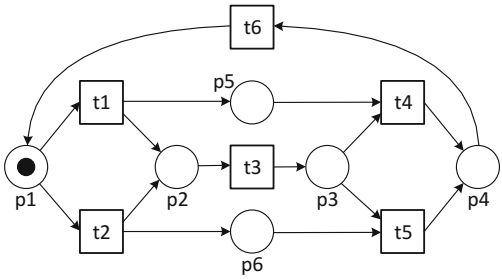


Fig. 2. A perpetual marked non-free-choice net that is not lucent because there are two reachable markings ($[p2, p5]$ and $[p2, p6]$) enabling the same set of transitions ($\{t3\}$).

do not limit ourselves to blocking markings. We consider all markings including states (partially) marking multiple clusters.

We expect that the theoretical results presented in this paper will enable new analysis techniques in related fields such as *business process management* [14], *workflow management* [24], and *process mining* [4]. Lucency is a natural assumption in many application domains and should be exploited. For example, the worklists of a workflow management system show the set of enabled actions. Hence, lucency allows us to reason about the internal state of the system in terms of the actions it allows. We also anticipate that lucency can be exploited in workflow verification, process discovery, and conformance checking [5]. Event logs used in process mining only reveal the actions performed and not the internal state [3,4]. Moreover, the class of perpetual marked free-choice nets considered in this paper is quite large and highly relevant in many application domains. The existence of a "regeneration point" (home cluster) is quite general, and liveness and boundedness (or soundness) are often desirable. For example, the class of short-circuited sound workflow nets is perpetual. Processes that are cyclic, often have a home cluster. Non-cyclic process often have a clear start and end state and can be short-circuited thus introducing a home cluster. For example, the representational bias of the α algorithm (i.e., the class of process models for which rediscovery is guaranteed) corresponds to a subclass of perpetual marked free-choice nets [9].

The remainder of this paper is organized as follows. Section 2 introduces preliminaries and known results (e.g., the blocking theorem). Section 3 defines lucency as a (desirable) behavioral property of marked Petri nets. Section 4 introduces perpetual nets and important notions like partial P-covers and local safeness. These are used to prove the main theorem of this paper showing that markings are unique in terms of the transitions they enable. Section 5 concludes the paper.

2 Preliminaries

This section introduces basic concepts related to Petri nets, subclasses of nets (e.g., free-choice nets and workflow nets), and blocking markings.

2.1 Petri Nets

Multisets are used to represent the state of a Petri net. $\mathcal{B}(A)$ is the set of all multisets over some set A. For some multiset $b \in \mathcal{B}(A)$, $b(a)$ denotes the number of times element $a \in A$ appears in b. Some examples: $b_1 = [\,]$, $b_2 = [x, x, y]$, $b_3 = [x, y, z]$, $b_4 = [x, x, y, x, y, z]$, and $b_5 = [x^3, y^2, z]$ are multisets over $A = \{x, y, z\}$. b_1 is the empty multiset, b_2 and b_3 both consist of three elements, and $b_4 = b_5$, i.e., the ordering of elements is irrelevant and a more compact notation may be used for repeating elements.

The standard set operators can be extended to multisets, e.g., $x \in b_2$, $b_2 \uplus b_3 = b_4$, $b_5 \backslash b_2 = b_3$, $|b_5| = 6$, etc. $\{a \in b\}$ denotes the set with all elements a for which $b(a) \geq 1$. $[f(a) \mid a \in b]$ denotes the multiset where element $f(a)$ appears $\sum_{x \in b | f(x) = f(a)} b(x)$ times.

$\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in X^*$ denotes a sequence over X of length n. $\langle\,\rangle$ is the empty sequence. Sequences can be concatenated using "$\cdot$", e.g., $\langle a, b \rangle \cdot \langle b, a \rangle = \langle a, b, b, a \rangle$. It is also possible to project sequences: $\langle a, b, b, a, c, d \rangle \restriction_{\{a,c\}} = \langle a, a, c \rangle$.

Definition 1 (Petri Net). *A Petri net is a tuple $N = (P, T, F)$ with P the non-empty set of places, T the non-empty set of transitions such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ the flow relation such that the graph $(P \cup T, F)$ is connected.*

Definition 2 (Marking). *Let $N = (P, T, F)$ be a Petri net. A marking M is a multiset of places, i.e., $M \in \mathcal{B}(P)$. (N, M) is a marked net.*

For a subset of places $X \subseteq P$: $M \restriction_X = [p \in M \mid p \in X]$ is the marking projected on this subset.

A Petri net $N = (P, T, F)$ defines a directed graph with nodes $P \cup T$ and edges F. For any $x \in P \cup T$, $\overset{N}{\bullet} x = \{y \mid (y, x) \in F\}$ denotes the set of input nodes and $x \overset{N}{\bullet} = \{y \mid (x, y) \in F\}$ denotes the set of output nodes. The notation can be generalized to sets: $\overset{N}{\bullet} X = \{y \mid \exists_{x \in X}\ (y, x) \in F\}$ and $X \overset{N}{\bullet} = \{y \mid \exists_{x \in X}\ (x, y) \in F\}$ for any $X \subseteq P \cup T$. We drop the superscript N if it is clear from the context.

A transition $t \in T$ is *enabled* in marking M of net N, denoted as $(N, M)[t\rangle$, if each of its input places $\bullet t$ contains at least one token. $en(N, M) = \{t \in T \mid (N, M)[t\rangle\}$ is the set of enabled transitions.

An enabled transition t may *fire*, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the output places $t\bullet$. Formally: $M' = (M \backslash \bullet t) \uplus t\bullet$ is the marking resulting from firing enabled transition t in marking M of Petri net N. $(N, M)[t\rangle(N, M')$ denotes that t is enabled in M and firing t results in marking M'.

Let $\sigma = \langle t_1, t_2, \ldots, t_n \rangle \in T^*$ be a sequence of transitions. $(N, M)[\sigma\rangle(N, M')$ denotes that there is a set of markings $M_0, M_1, \ldots, M_n$ $(n \geq 0)$ such that $M_0 = M$, $M_n = M'$, and $(N, M_i)[t_{i+1}\rangle(N, M_{i+1})$ for $0 \leq i < n$. A marking M' is *reachable* from M if there exists a *firing sequence* σ such that $(N, M)[\sigma\rangle(N, M')$. $R(N, M) = \{M' \in \mathcal{B}(P) \mid \exists_{\sigma \in T^*}\ (N, M)[\sigma\rangle(N, M')\}$ is the set of all reachable markings.

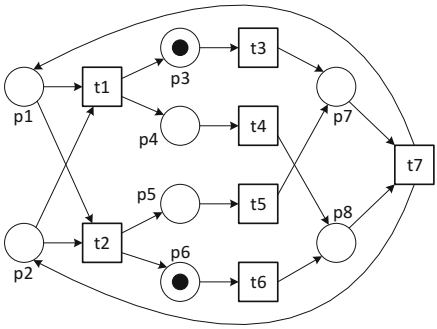


Fig. 3. A perpetual marked free-choice net [13]. The net is live, bounded, and has so-called "home clusters" (e.g., $\{p7, p8, t7\}$). The net is also lucent.

Figure 3 shows a marked Petri net having 8 places and 7 transitions. Transitions $t3$ and $t6$ are enabled in the initial marking $M = [p3, p6]$. $R(N, M) = \{[p3, p6], [p6, p7], [p3, p8], [p7, p8], [p1, p2], [p3, p4], [p5, p6], [p4, p7], [p5, p8]\}$. For example, the firing sequence $\langle t3, t6, t7\rangle$ leads to marking $[p1, p2]$, i.e., $(N, [p3, p6])[\langle t3, t6, t7\rangle\rangle(N, [p1, p2])$.

A *path* in a Petri net $N = (P, T, F)$ is a sequence of nodes $\rho = \langle x_1, x_2, \ldots, x_n\rangle$ such that $(x_i, x_{i+1}) \in F$ for $1 \leq i < n$. ρ is an elementary path if $x_i \neq x_j$ for $1 \leq i < j \leq n$.

Next, we define a few, often desirable, behavioral properties: liveness, boundedness, and the presence of (particular) home markings.

Definition 3 (Liveness and Boundedness). *A marked net (N, M) is* live *if for every reachable marking $M' \in R(N, M)$ and every transition $t \in T$ there exists a marking $M'' \in R(N, M')$ that enables t. A marked net (N, M) is k-bounded if for every reachable marking $M' \in R(N, M)$ and every $p \in P$: $M'(p) \leq k$. A marked net (N, M) is* bounded *if there exists a k such that (N, M) is k-bounded. A 1-bounded marked net is called* safe. *A Petri net N is* structurally bounded *if (N, M) is bounded for any marking M. A Petri net N is* structurally live *if there exists a marking M such that (N, M) is live. A Petri net N is* well-formed *if there exists a marking M such that (N, M) is live and bounded.*

The marked Petri net shown in Fig. 3 is live and safe. Hence, it is also well-formed.

Definition 4 (Home Marking). *Let (N, M) be a marked net. A marking M_H is a home marking if for every reachable marking $M' \in R(N, M)$: $M_H \in R(N, M')$. (N, M) is cyclic if M is a home marking.*

The marked Petri net shown in Fig. 3 has 8 home markings: $\{[p6, p7], [p3, p8], [p7, p8], [p1, p2], [p3, p4], [p5, p6], [p4, p7], [p5, p8]\}$. However, the net is not cyclic because $[p3, p6]$ is not a home marking.

2.2 Subclasses of Petri Nets

For particular subclasses of Petri net there is a relationship between structural properties and behavioral properties like liveness and boundedness [12]. In this paper, we focus on free-choice nets [13].

Definition 5 (P-net, T-net, and Free-choice Net). *Let $N = (P, T, F)$ be a Petri net. N is an P-net (also called a state machine) if $|{\bullet}t| = |t{\bullet}| = 1$ for any $t \in T$. N is a T-net (also called a marked graph) if $|{\bullet}p| = |p{\bullet}| = 1$ for any $p \in P$. N is free-choice net if for any for any $t_1, t_2 \in T$: ${\bullet}t_1 = {\bullet}t_2$ or ${\bullet}t_1 \cap {\bullet}t_2 = \emptyset$. N is strongly connected if the graph $(P \cup T, F)$ is strongly connected, i.e., for any two nodes x and y there is a path leading from x to y.*

An alternative way to state that a net is free-choice is the requirement that for any $p_1, p_2 \in P$: $p_1{\bullet} = p_2{\bullet}$ or $p_1{\bullet} \cap p_2{\bullet} = \emptyset$. The Petri net shown in Fig. 3 is free-choice. The Petri net shown in Fig. 2 is not free-choice because $t4$ and $t5$ shared an input place ($p3$) but have different sets of input places.

Definition 6 (Siphon and Trap). *Let $N = (P, T, F)$ be a Petri net and $R \subseteq P$ a subset of places. R is a siphon if ${\bullet}R \subseteq R{\bullet}$. R is a trap if $R{\bullet} \subseteq {\bullet}R$. A siphon (trap) is called proper if it is not the empty set.*

Any transition that adds tokens to a siphon also takes tokens from the siphon. Therefore, an unmarked siphon remains unmarked. Any transition that takes tokens from a trap also adds tokens to the trap. Therefore, a marked trap remains marked.

Definition 7 (Cluster). *Let $N = (P, T, F)$ be a Petri net and $x \in P \cup T$. The cluster of node x, denoted $[x]_c$ is the smallest set such that (1) $x \in [x]_c$, (2) if $p \in [x]_c \cap P$, then $p{\bullet} \subseteq [x]_c$, and (3) if $t \in [x]_c \cap T$, then ${\bullet}t \subseteq [x]_c$. $[N]_c = \{[x]_c \mid x \in P \cup T\}$ is the set of clusters of N.*

Note that $[N]_c$ partitions the nodes in N. The Petri net shown in Fig. 3 has 6 clusters: $[N]_c = \{\{p1, p2, t1, t2\}, \{p3, t3\}, \{p4, t4\}, \{p5, t5\}, \{p6, t6\}, \{p7, p8, t7\}\}$.

Definition 8 (Cluster Notations). *Let $N = (P, T, F)$ be a Petri net and $C \in [N]_c$ a cluster. $P(C) = P \cap C$ are the places in C, $T(C) = T \cap C$ are the transitions in C, and $M(C) = [p \in P(C)]$ is the smallest marking fully enabling the cluster.*

Definition 9 (Subnet, P-component, T-Component). *Let $N = (P, T, F)$ be a Petri net and $X \subseteq P \cup T$ such that $X \neq \emptyset$. $N{\restriction}_X = (P \cap X, T \cap X, F \cap (X \times X))$ is the subnet generated by X. $N{\restriction}_X$ is a* P-component *of N if $\overset{N}{\bullet}p \cup p\overset{N}{\bullet} \subseteq X$ for $p \in X \cap P$ and $N{\restriction}_X$ is a strongly connected P-net. $N{\restriction}_X$ is a* T-component *of N if $\overset{N}{\bullet}t \cup t\overset{N}{\bullet} \subseteq X$ for $t \in X \cap T$ and $N{\restriction}_X$ is a strongly connected T-net. $PComp(N) = \{X \subseteq P \cup T \mid N{\restriction}_X$ is a P-component$\}$. $TComp(N) = \{X \subseteq P \cup T \mid N{\restriction}_X$ is a T-component$\}$.*

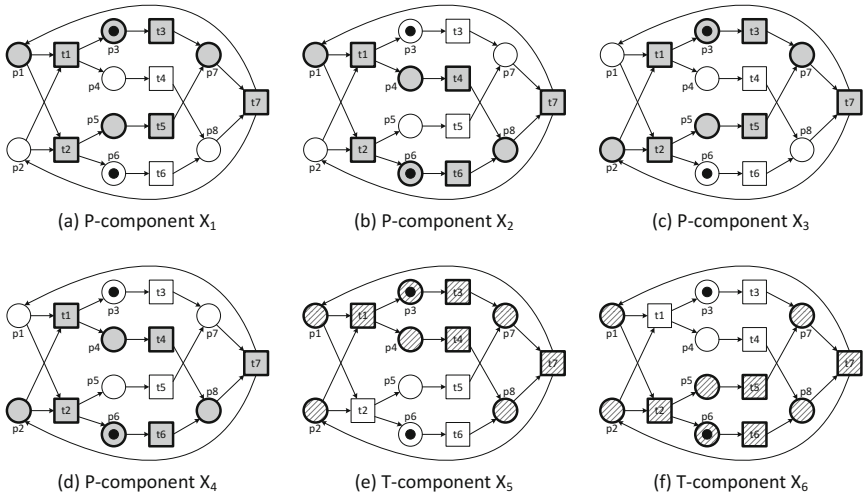


Fig. 4. The Petri net shown in Fig. 3 has four P-components and two T-components.

The Petri net shown in Fig. 3 has four P-components and two T-components (see Fig. 4). These components cover all nodes of the net.

Definition 10 (P-cover, T-cover). *Let $N = (P, T, F)$ be a Petri net. N has a* P-cover *if and only if* $\bigcup PComp(N) = P \cup T$.[1] *$N$ has a* T-cover *if and only if* $\bigcup TComp(N) = P \cup T$.

Since the early seventies, it is known that well-formed free-choice nets have a P-cover and a T-cover (first shown by Michel Hack).

Theorem 1 (Coverability Theorem [13]). *Let $N = (P, T, F)$ be a well-formed free-choice net.* $\bigcup PComp(N) = \bigcup TComp(N) = P \cup T$.

Moreover, for any well-formed free-choice net N and marking M: (N, M) is live if and only if every P-component is marked in M (Theorem 5.8 in [13]).

2.3 Workflow Nets

In the context of business process management, workflow automation, and process mining, often a subclass of Petri nets is considered where each net has a unique source place i and a unique sink place o [1].

Definition 11 (Workflow net). *Let $N = (P, T, F)$ be a Petri net. N is a workflow net if there are places $i, o \in P$ such that $\bullet i = \emptyset$, $o\bullet = \emptyset$, and all nodes $P \cup T$ are on a path from i to o. Given a workflow net N, the short-circuited net is $\overline{N} = (P, T \cup \{t^*\}, F \cup \{(t^*, i), (o, t^*)\})$.*

[1] $\bigcup Q = \bigcup_{X \in Q} X$ for some set of sets Q.

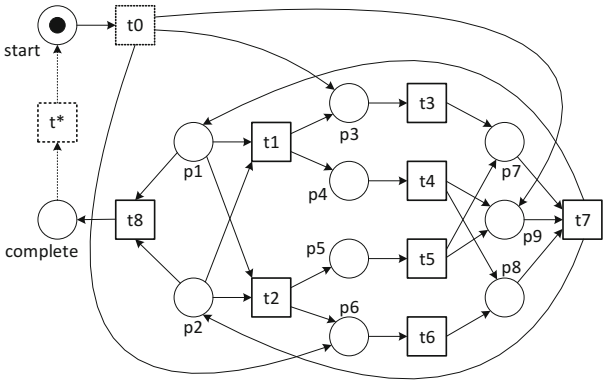


Fig. 5. The free-choice net without transition t^* is a workflow net. The short-circuited net is perpetual, i.e., live, bounded, and having a home cluster (e.g., $\{start, t0\}$). The short-circuited net is also lucent.

The short-circuited net is strongly connected. Different notions of soundness have been defined [6]. Here we only consider classical soundness [1].

Definition 12 (Sound). *Let $N = (P, T, F)$ be a workflow net with source place i. N is sound if and only if $(\overline{N}, [i])$ is live and bounded.*

Note that soundness implies that starting from the initial state (just a token in place i), it is always possible to reach the state with one token in place o (marking $[o]$). Moreover, after a token is put in place o, all the other places need to be empty. Finally, there are no dead transitions (each transition can become enabled).

Figure 5 shows a sound workflow net. By adding transition t^* the net is short-circuited. The short-circuited net is live, safe, and cyclic.

2.4 Uniqueness of Blocking Markings in Free-Choice Nets

A *blocking marking* is a marking where all transitions in a particular cluster are enabled while all others are disabled. For example, in Fig. 3, $[p3, p6]$ is not a blocking marking, but $[p3, p8]$, $[p6, p7]$, and $[p7, p8]$ are examples of blocking markings.

Definition 13 (Blocking Marking). *Let (N, M) be a marked net and $C \in [N]_c$ a cluster. A* blocking marking *for C is a marking $M_B \in R(N, M)$ such that $en(N, M_B) = T(C)$, i.e., all transitions in the cluster are enabled, but no other transitions.*

In [18] Genrich and Thiagarajan showed that unique blocking markings exist for all clusters in live and safe marked graphs. This was generalized by Gaujal, Haar, and Mairesse in [17] where they showed that blocking markings exist and

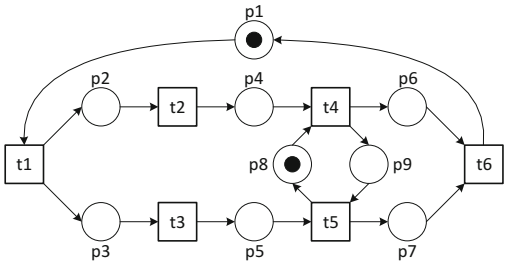


Fig. 6. A live and safe marked free-choice net that is not locally safe and not perpetual. Nevertheless, the net is lucent.

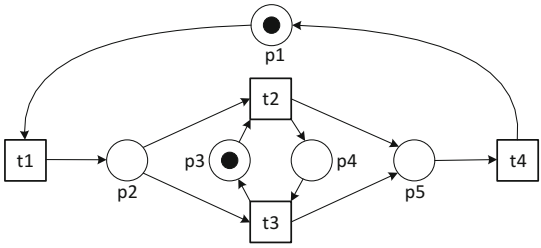


Fig. 7. A live and locally safe non-free-choice net. Cluster $\{p1, t1\}$ has two reachable blocking markings $M_1 = [p1, p3]$ and $M_2 = [p1, p4]$. Also cluster $\{p5, t4\}$ has two reachable blocking markings $M_3 = [p3, p5]$ and $M_4 = [p4, p5]$.

are unique in live and bounded free-choice nets. Note that in a free-choice net all transitions in the cluster are enabled simultaneously (or all are disabled). There is one unique marking in which precisely one cluster is enabled. Moreover, one can reach this marking without firing transitions from the cluster that needs to become enabled. A simplified proof was given in [26] and another proof sketch was provided in [12].

Theorem 2 (Existence and Uniqueness of Blocking Markings [17]**).** *Let* (N, M) *live and bounded free-choice net and* $C \in [N]_c$ *a cluster. There exists a unique blocking marking for* C *reachable from* (N, M)*, denoted by* $B^C_{(N,M)}$*. Moreover, there exists a firing sequence* $\sigma \in (T \setminus C)^*$ *such that* $(N, M)[\sigma\rangle(N, B^C_{(N,M)})$.

The free-choice net in Fig. 6 is live and bounded. Hence, each cluster has a unique blocking marking. The unique blocking marking of the cluster $\{p2, t2\}$ is $[p2, p5]$. The unique blocking marking of the cluster $\{p6, p7, t6\}$ is $[p6, p7, p8]$.

The free-choice net in Fig. 1 has three clusters. The three blocking markings are $[p1, p2]$, $[p1, p3]$, and $[p2, p4]$. Marking $[p3, p4]$ is not a blocking marking because transitions from different clusters are enabled.

Figure 7 illustrates that the free-choice property is essential in Theorem 2. Cluster $C_1 = \{p1, t1\}$ has two reachable blocking markings $M_1 = [p1, p3]$ and $M_2 = [p1, p4]$. Cluster $C_2 = \{p5, t4\}$ also has two reachable blocking markings $M_3 = [p3, p5]$ and $M_4 = [p4, p5]$.

3 Lucency

This paper focuses on the question whether markings can be uniquely identified based on the transitions they enable. Given a marked Petri net we would like to know whether each reachable marking has a unique "footprint" in terms of the transitions it enables. If this is the case, then the net is *lucent*.

Definition 14 (Lucent). *Let (N, M) be a marked Petri net. (N, M) is lucent if and only if for any $M_1, M_2 \in R(N, M)$: $en(N, M_1) = en(N, M_2)$ implies $M_1 = M_2$.*

The marked Petri net in Fig. 1 is lucent because each of the four reachable markings has a unique footprint in terms of the set of enabled transitions. The marked Petri net shown in Fig. 2 is not lucent because there are two markings $M_1 = [p2, p5]$ and $M_2 = [p2, p6]$ with $en(N, M_1) = en(N, M_2) = \{t3\}$ and $M_1 \neq M_2$. The marked Petri nets in Figs. 3, 5, and 6 are lucent. The non-free-choice net in Fig. 7 is not lucent (markings $[p3, p5]$ and $[p4, p5]$ enable $t4$, and $[p1, p3]$ and $[p1, p4]$ enable $t1$). Figure 8 shows a free-choice net that is also not lucent (markings $[p3, p7, p8]$ and $[p3, p5, p7]$ both enable $\{t1, t4\}$).

Lemma 1. *Let (N, M) be a lucent marked Petri net. (N, M) is bounded and each cluster has at most one blocking marking.*

Proof. Assume that (N, M) is both lucent and unbounded. We will show that this leads to a contradiction. Since (N, M) is unbounded, we can find markings M_1 and M_2 and sequences σ_0 and σ such that $(N, M)[\sigma_0\rangle(N, M_1)[\sigma\rangle(N, M_2)$ and M_2 is strictly larger than M_1. This implies that we can repeatedly execute σ getting increasingly larger markings: $(N, M_2)[\sigma\rangle(N, M_3)[\sigma\rangle(N, M_4)[\sigma\rangle(N, M_5)\ldots$. At some stage, say at M_k, the set of places that are marked stabilizes. However, the number of tokens in some places continues to increase in M_{k+1}, M_{k+2}, etc. Hence, we find markings that enable the same set of transitions but that are not the same. For example, $M_{k+1} \neq M_{k+2}$ and $en(N, M_{k+1}) = en(N, M_{k+2})$. Hence, the net cannot be lucent.

Take any cluster C and assume that (N, M) has two different reachable blocking markings M_1 and M_2. This means that $en(N, M_1) = en(N, M_2) = C \cap T$. Hence, (N, M) could not be lucent, yielding a contradiction again. □

We would like to find subclasses of nets that are guaranteed to be lucent based on their structure. Theorem 2 and the fact that lucency implies the existence unique blocking markings, suggest considering live and bounded free-choice nets. However, as the example in Fig. 8 shows, this is not sufficient.

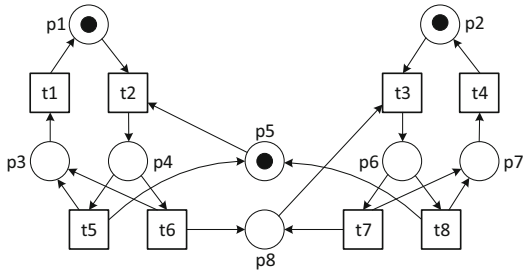


Fig. 8. A live and locally safe free-choice net that is not lucent because reachable markings $[p3, p7, p8]$ and $[p3, p5, p7]$ both enable $t1$ and $t4$.

4 Characterizing Markings of Perpetual Free-Choice Nets

Theorem 2 only considers blocking markings, but illustrates that the free-choice property is important for lucency. Consider for example Fig. 7. $M_1 = [p1, p3]$ and $M_2 = [p1, p4]$ both enable $t1$. $M_3 = [p3, p5]$ and $M_4 = [p4, p5]$ both enable $t4$. Obviously, the property does not hold for non-free-choice nets even when they are live, safe, cyclic, etc. However, as Fig. 8 shows, the property also does not need to hold for free-choice nets even when they are live, safe, and cyclic. Yet, we are interested in the class of nets for which all reachable markings have a unique "footprint" in terms of the transitions they enable. Therefore, we introduce the class of *perpetual nets*. These nets have a "regeneration point" involving a so-called "home cluster".

4.1 Perpetual Marked Nets

A *home cluster* is a cluster corresponding to a home marking, i.e., the places of the cluster can be marked over and over again while all places outside the cluster are empty.

Definition 15 (Home Cluster). *Let (N, M) be a marked net with $N = (P, T, F)$ and $C \in [N]_c$ a cluster of N. C is a* home cluster *of (N, M) if $M(C)$ is a home marking, i.e., for every reachable marking $M' \in R(N, M)$: $M(C) \in R(N, M')$.*

Consider the marked net in Fig. 1. There are three clusters: $C_1 = \{p1, p2, t1, t2\}$, $C_2 = \{p3, t3\}$, and $C_3 = \{p4, t4\}$. C_1 is a home cluster because $M(C_1) = [p1, p2]$ is a home marking. C_2 is not a home cluster because $M(C_2) = [p3]$ is not a home marking. C_3 is also not a home cluster because $M(C_3) = [p4]$ is not a home marking.

The marked net in Fig. 7 also has three clusters: $C_1 = \{p1, t1\}$, $C_2 = \{p2, p3, p4, t2, t3\}$, and $C_3 = \{p5, t4\}$. Since $[p1]$, $[p2, p3, p4]$, and $[p5]$ are not home markings, the net has no home clusters.

Nets that are live, bounded, and have at least one home cluster are called *perpetual*.

Definition 16 (Perpetual Marked Net). *A marked net* (N, M) *is perpetual if it is live, bounded, and has a home cluster.*

The marked Petri nets in Figs. 1, 2, 3, and 5 are perpetual. The nets in Figs. 6, 7, and 8 are not perpetual. Home clusters can be viewed as "regeneration points" because the net is always able to revisit a state marking a single cluster.

Lemma 2 (Sound Workflow Nets are Perpetual). *Let* N *be a sound workflow net with source place* i*. The short-circuited marked net* $(\overline{N}, [i])$ *is perpetual.*

Proof. Soundness implies that $(\overline{N}, [i])$ is live and bounded. Moreover, $[i]$ is a home cluster. It is always possible to enable and fire t^* due to liveness. After firing t^*, place i is marked and there can be no other tokens because otherwise $(\overline{N}, [i])$ would be unbounded. Hence, $[i]$ is a home marking. $\{i\} \cup i\bullet$ is a cluster because the transitions in $i\bullet$ cannot have additional input places (otherwise they would be dead). □

Next to workflow nets, there are many classes of nets that have a "regeneration point" (i.e., home cluster). For example, process models discovered by discovery algorithms often have a well-defined start and end point. By short-circuiting such nets, one gets home clusters.

4.2 Local Safeness

It is easy to see that non-safe Petri nets are likely to have different markings enabling the same set of transitions. In fact, we need a stronger property that holds for perpetual marked free-choice nets: *local safeness*. Local safeness is the property that each P-component is safe (i.e., the sum of all tokens in the component cannot exceed 1).

Definition 17 (Locally Safe). *Let* (N, M) *be a marked P-coverable net.* (N, M) *is* locally safe *if all P-components are safe, i.e., for any P-component* $X \in PComp(N)$ *and reachable marking* $M' \in R(N, M)$*:* $\sum_{p \in X \cap P} M'(p) \leq 1$.

Note that a safe marked P-coverable net does not need to be locally safe. Consider for example the marked net in Fig. 6. The net is safe, but the P-component $\{p1, p3, p5, p8, p6, t1, t3, t4, t5, t6\}$ has two tokens. However, all perpetual marked free-choice nets are locally safe.

Lemma 3 (Perpetual Marked Free-Choice Nets Are Locally Safe). *Let* (N, M) *be a perpetual marked free-choice net.* (N, M) *is locally safe.*

Proof. Since (N, M) is perpetual, therefore it is live, bounded, and has a home cluster C. N is well-formed and therefore has a P-cover. A bounded well-formed free-choice net is only live if every P-component is initially marked (see Theorem 5.8 in [13]). Hence, also in home marking $M(C)$ the P-components are marked (the number of tokens is invariant). Therefore, in any P-component one of the places in $P(C)$ appears. There cannot be two places from cluster C in the

same P-component (this would violate the requirement that transitions in a P-component have precisely one input place). Hence, each P-component is marked with precisely one token and this number is invariant for all reachable markings. Hence, (N, M) is locally safe. □

The nets in Figs. 1, 3, and 5 are free-choice and perpetual and therefore also locally safe. The net in Fig. 2 is locally safe and perpetual, but not free-choice. The marked Petri net in Fig. 6 is not perpetual and also not locally safe. Figure 8 shows that there are free-choice nets that are live and locally safe, but not perpetual.

4.3 Realizable Paths

Free-choice nets have many interesting properties showing that the structure reveals information about the behavior of the net [13]. Tokens can basically "decide where to go", therefore such nets are called free-choice.

The following lemma from [19] shows that tokens can follow an elementary path in a live and bounded free-choice net where the initial marking marks a single place and that is a home marking.

Lemma 4 (Realizable Paths in Cyclic Free-Choice Nets [19]). *Let (N, M) be a live, bounded, and cyclic marked free-choice net with $M = [p_H]$. Let M' be a reachable marking which marks place q and let $\langle p_0, t_1, p_1, t_2, \ldots, t_n, p_n\rangle$ with $p_0 = q$ and $p_n = p_H$ be an elementary path in N. There exists a firing sequence σ such that $(N, M')[\sigma\rangle(N, M)$ and each of the transitions $\{t_1, \ldots t_n\}$ is executed in the given order and none of the intermediate markings marks p_H.*

Proof. See [19]. □

Note that Lemma 4 refers to a subclass of perpetual marked free-choice nets. A similar result can be obtained for P-components in a perpetual marked free-choice net.

Lemma 5 (Realizable Paths Within P-components). *Let (N, M) be a perpetual mark-ed free-choice net with home cluster C. Let $X \in PComp(N)$ be the nodes of some P-component and $M' \in R(N, M)$ an arbitrary reachable marking. For any elementary path $\langle p_0, t_1, p_1, t_2, \ldots, t_n, p_n\rangle \in X^*$ in N with $p_0 \in M'$ and $p_n \in P(C)$: there exists a firing sequence σ such that $(N, M')[\sigma\rangle(N, M(C))$ and $\sigma{\restriction}_X = \langle t_1, t_2, \ldots, t_n\rangle$.*

Proof. Let (P_X, T_X, F_X) be the P-component corresponding to X. Note that p_0 is the only place of P_X that is marked in M'. Moreover, elements in $\langle p_0, t_1, p_1, t_2, \ldots, t_n, p_n\rangle \in X^*$ are unique because the path is elementary. In fact, the places in $\{p_0, p_1, \ldots, p_n\} \subseteq P_X$ belong to different clusters because a P-component cannot have multiple places of the same cluster. As a result also $\{t_1, t_2, \ldots t_n\} \subseteq T_X$ belong to different clusters.

If $p_0 = p_n$, then the lemma holds because $p_n \in P(C)$ is marked and we can also mark the other places in $P(C)$. Theorem 2 can be applied such that all

places in $P(C)$ can be marked without firing any transitions in $T(C)$. In fact, there exists a sequence σ_B such that $(N, M')[\sigma_B\rangle(N, M(C))$ and $\sigma_B{\restriction_X} = \langle\,\rangle$. σ_B does not involve any transitions in T_X, because $T(C)$ transitions are not needed and all other transitions in T_X are blocked because p_n is the only place in P_X that is marked. When all places in $P(C)$ are marked, then all other places need to be empty, otherwise (N, M) is not bounded (see Lemma 2.22 in [13]). Hence, σ_B leads indeed to $M(C)$.

If $p_0 \neq p_n$, then there is a firing sequence removing the token from p_0 (because $M(C)$ is a home marking and $p_0 \notin M(C)$). Let $(N, M')[\sigma_1\rangle(N, M_1)$ be the sequence enabling a transition that removes the token from p_0 (for the first time). In M_1, transition $t_1 \in p_0\bullet$ is enabled (because N is free-choice all transitions in $p_0\bullet$ are enabled). σ_1 cannot fire any transitions in T_X, because p_0 is the only place of P_X that is marked. Therefore, transitions in $[p_0]_c$ need to be enabled first. Let M_1' be the marking after firing t_1 ($(N, M')[\sigma_1\rangle(N, M_1)[t_1\rangle(N, M_1')$). Note that p_1 is the only place of P_X marked in M_1'. Let $(N, M')[\sigma_2\rangle(N, M_2)$ be the sequence enabling a transition that removes the token from p_1. Transition t_2 is enabled in the marking reached after σ_2: $(N, M_2)[t_2\rangle(N, M_2')$. Again σ_2 cannot involve any transitions in T_X and enables all transitions in $p_1\bullet$. M_2' marks place p_2 as the only place in P_X. By recursively applying the argument it is possible to construct the firing sequence $\sigma' = \sigma_1 \cdot t_1 \cdot \sigma_2 \cdot t_2 \cdot \ldots \cdot \sigma_n \cdot t_n$ which marks p_n. From the resulting marking one can fire σ_B leading to marking $M(C)$. For the case $p_0 = p_n$ we explained that such a σ_B exists. This shows that we can construct $\sigma = \sigma' \cdot \sigma_B$ such that $(N, M')[\sigma\rangle(N, M(C))$ and $\sigma{\restriction_X} = \langle t_1, t_2, \ldots, t_n\rangle$. □

4.4 Partial P-Covers

Hack's Coverability Theorem (Theorem 1) states that well-formed free-choice nets have a P-cover. Our proof that markings are distinguishable based on their enabled transitions exploits this. In fact, we will construct nets using subsets of P-components. Therefore, we define a notion of a Q-projection.

Definition 18 (Partial P-cover and Projection). *Let (N, M) be a marked P-coverable Petri net. Any $Q \subseteq PComp(N)$ with $Q \neq \emptyset$ is a partial P-cover of N. $N{\restriction_{\bigcup Q}}$ is the Q-projection of N. $(N{\restriction_{\bigcup Q}}, M{\restriction_{\bigcup Q}})$ is the marked Q-projection of (N, M).*

A Q-projection inherits properties from the original net (free-choice and well-formed) and the Q-projection is again P-coverable.

Lemma 6. *Let $N = (P, T, F)$ be a P-coverable Petri net, Q a partial P-cover of N, and $N{\restriction_{\bigcup Q}} = (P_Q, T_Q, F_Q)$ the Q-projection of N. $\bigcup Q = P_Q \cup T_Q$, $Q \subseteq PComp(N{\restriction_{\bigcup Q}}) \subseteq PComp(N)$, and $N{\restriction_{\bigcup Q}}$ has a P-cover.*

Proof. Let $Q = \{X_1, X_2, \ldots X_n\}$ be P-components of N, $P_i = X_i \cap P$, $T_i = X_i \cap T$, for $1 \leq i \leq n$. $N{\restriction_{\bigcup Q}} = (P_Q, T_Q, F_Q)$ such that $P_Q = \bigcup_i P_i$ and $T_Q = \bigcup_i T_i$. Hence, by definition $\bigcup Q = P_Q \cup T_Q$.

Each P-component X_i is fully described by P_i, because in any P-component, place p is always connected to the transitions in ${}^{N}\bullet p$ and $p{}^{N}\bullet$. All the original components in Q used to form the partial P-cover of N are also components of $N\restriction_{\bigcup Q}$, because the subsets of places are in P_Q and all surrounding transitions are included and no new transitions have been added. However, new combinations may be possible (covering subsets of the places in P_Q). Hence, $Q \subseteq PComp(N\restriction_{\bigcup Q})$. $PComp(N\restriction_{\bigcup Q}) \subseteq PComp(N)$ because a partial P-cover cannot introduce new P-components. $N\restriction_{\bigcup Q}$ has a P-cover, because $\bigcup PComp(N\restriction_{\bigcup Q}) = P_Q \cup T_Q$. □

Lemma 7. *Let $N = (P, T, F)$ be a well-formed free-choice net and Q a partial P-cover of N. The Q-projection of N (i.e., $N\restriction_{\bigcup Q}$) is a well-formed free-choice net.*

Proof. Let $N\restriction_{\bigcup Q} = N_Q = (P_Q, T_Q, F_Q)$. N_Q is free-choice because N is free-choice and for any added place $p \in P_Q$ all surrounding transitions $\bullet p \cup p\bullet$ are also added. Hence, for any $p_1, p_2 \in P_Q$: $p_1\bullet = p_2\bullet$ or $p_1\bullet \cap p_2\bullet = \emptyset$.

N_Q is structurally bounded because it is covered by P-components (Lemma 6). The number of tokens in a P-component is constant and serves as an upper bound for the places in it.

To show that N_Q is structurally live we use Commoner's Theorem [13]: "A free-choice marked net is live if and only if every proper siphon includes an initially marked trap". Places in N and N_Q have identical pre and post-sets, hence, for any $R \subseteq P_Q$: ${}^{N}\bullet R = {}^{N_Q}\bullet R$ and $R{}^{N}\bullet = R{}^{N_Q}\bullet$. Hence, R cannot be a siphon in N and not in N_Q (or vice versa). ${}^{N}\bullet R \subseteq R{}^{N}\bullet$ if and only if ${}^{N_Q}\bullet R \subseteq R{}^{N_Q}\bullet$. Also, R cannot be a trap in N and not in N_Q (of vice versa). $R{}^{N}\bullet \subseteq {}^{N}\bullet R$ if and only if $R{}^{N_Q}\bullet \subseteq {}^{N_Q}\bullet R$. Take any proper siphon R in N_Q. This is also a proper siphon in N. R contains a proper trap R' in N. Clearly, $R' \subseteq P_Q$ and is also a proper trap in N_Q. By initially marking all places, R' is also marked and the net is (structurally) live. Therefore, N_Q is well-formed. □

A partial P-cover of N may remove places. Removing places can only enable more behavior. Transitions are only removed if none of the input and output places are included. Therefore, any firing sequence in the original net that is projected on the set of remaining transitions is enabled in the net based on the partial P-cover.

Lemma 8. *Let (N, M) be a live and locally safe marked free-choice net (with $N = (P, T, F)$), Q a partial P-cover of N, and (N_Q, M_Q) the marked Q-projection of (N, M) (with $N_Q = (P_Q, T_Q, F_Q)$). For any sequence $\sigma \in T^*$ that is executable in (N, M) (i.e., $(N, M)[\sigma\rangle(N, M')$), the projected sequence $\sigma_Q = \sigma\restriction_{T_Q}$ is also executable in the marked Q-projection and ends in marking $M'\restriction_{\bigcup Q}$ (i.e., $(N_Q, M_Q)[\sigma_Q\rangle(N_Q, M'\restriction_{\bigcup Q})$).*

Proof. Let (N_Q, M_Q) be the marked Q-projection of (N, M). N_Q has fewer places. Removing places can only enable more behavior and never block behavior. Therefore, σ is still possible after removing the places not part of any of the included P-components. After removing these places, transitions not in any included P-component become disconnected and can occur without any constraints. Hence, σ can be replayed and results in the projected marking $(M'{\restriction}_{\bigcup Q})$. Removing these transitions from the sequence $(\sigma_Q = \sigma {\restriction}_{T_Q})$ corresponds to removing them from the net. Therefore, $(N_Q, M_Q)[\sigma_Q\rangle(N_Q, M'{\restriction}_{\bigcup Q})$. □

By combining the above insights, we can show that the Q-projection of a perpetual marked free-choice net is again a perpetual marked free-choice net.

Lemma 9. *Let (N, M) be a perpetual marked free-choice net and Q a partial P-cover of N. The marked Q-projection of (N, M) (i.e., $(N{\restriction}_{\bigcup Q}, M{\restriction}_{\bigcup Q})$) is a perpetual marked free-choice net.*

Proof. Let $N{\restriction}_{\bigcup Q} = N_Q = (P_Q, T_Q, F_Q)$ and $M_Q = M{\restriction}_{\bigcup Q}$. N_Q is a well-formed free-choice net (see Lemma 7). To prove that (N_Q, M_Q) is perpetual, we need to show that it is live, bounded, and has a home cluster.

Let C be a home cluster of (N, M). Every P-component of N includes precisely one place of $P(C)$ and holds precisely one token in any reachable state. Any P-component in N_Q is also a P-component in N (Lemma 6) and therefore also has one token in any reachable state. Hence, (N_Q, M_Q) is locally safe.

Every P-component of N_Q is marked in M and also in M_Q. (Lemma 6 shows that $PComp(N{\restriction}_{\bigcup Q}) \subseteq PComp(N)$. This implies that all components of N_Q are also components of N and thus initially marked.) Hence, we can apply Theorem 5.8 in [13] to show that the net is live.

$C_Q = C \cap (P_Q \cup T_Q)$ is a home cluster of (N_Q, M_Q) because the transitions in $C_Q \cap T_Q$ are live and when they are enabled only the places in $P(C_Q)$ are marked. Hence, $M(C_Q)$ is a home marking. □

4.5 Characterization of Markings in Perpetual Free-Choice Nets

We have introduced perpetual free-choice nets because it represents a large and relevant class of models for which the enabling of transitions uniquely identifies a marking, i.e., these nets are lucent. In such process models, there can never be two different markings enabling the same set of transitions. Note that this result is much more general than the blocking marking theorem which only refers to blocking markings and a single cluster.

Theorem 3 (Characterization of Markings in Perpetual Free-Choice Nets). *Let (N, M) be a perpetual marked free-choice net. (N, M) is lucent.*

Proof. Let $N = (P, T, F)$ and $M_1, M_2 \in R(N, M)$ such that $E = en(N, M_1) = en(N, M_2)$. We need to prove that $M_1 = M_2$.

N has a P-cover (Theorem 1). $Q_E = \{X \in PComp(N) \mid X \cap E \neq \emptyset\}$ are all P-components covering at least one transition in E. $N_E = N\restriction_{\bigcup Q_E} = (P_E, T_E, F_E)$ is the Q_E-projection of N. $M_E = M\restriction_{\bigcup Q_E}$ is the corresponding marking. (N_E, M_E) is a perpetual marked free-choice net (see Lemma 9).

M_1 and M_2 "agree" on the places in P_E, i.e., $M_1(p) = M_2(p)$ for $p \in P_E$. The places in $\bullet E$ are marked in M_1 and M_2, because the transitions in $E = en(N, M_1) = en(N, M_2)$ are enabled in both. The places in $P_E \backslash \bullet E$ are empty in M_1 and M_2, because in any reachable marking there is precisely one token in each P-component in Q_E (Lemma 3). Each place in $P_E \backslash \bullet E$ is part of at least one P-component already marking a place in $\bullet E$. Hence, the places in $P_E \backslash \bullet E$ need to be empty in markings M_1 and M_2. $M_0 = [p \in \bullet E] = M_1\restriction_{\bigcup Q_E} = M_2\restriction_{\bigcup Q_E}$ is a shorthand for the common part of M_1 and M_2 in the Q_E-projection of N.

Let's assume $M_1 \neq M_2$ and show that this leads to a contradiction. There is a place p_d for which both markings disagree: $M_1(p_d) \neq M_2(p_d)$. Let $X_d \in PComp(N)$ be an arbitrary P-component such that $p_d \in X_d$. $X_d \notin Q_E$ because M_1 and M_2 "agree" on the places in P_E. Let $P_d = X_d \cap P$ and $T_d = X_d \cap T$ be the places and transitions of P-component X_d for which M_1 and M_2 disagree.

Obviously M_1 and M_2 mark different places in P_d, but both mark precisely one of these places (P-component). Let $p_1^d \in P_d$ and $p_2^d \in P_d$ be the two places marked by respectively M_1 and M_2. $M_1\restriction_{X_d} = [p_1^d]$ and $M_2\restriction_{X_d} = [p_2^d]$. Clearly, $\{p_1^d, p_2^d\} \cap P_E = \emptyset$ since M_1 and M_2 "agree" on the places in P_E. Moreover, the transitions in T_d are not enabled in both M_1 and M_2. $T_1 = p_1^d\bullet$ and $T_2 = p_2^d\bullet$ are the transitions having respectively p_1^d and p_2^d as input place.

Let $Q_F = Q_E \cap \{X_d\}$. $N_F = N\restriction_{\bigcup Q_F} = (P_F, T_F, F_F)$ is the Q_F-projection of N. $M_F = M\restriction_{\bigcup Q_F}$ is the corresponding marking. Also, (N_F, M_F) is a perpetual marked free-choice net (apply again Lemma 9).

Let $M_1^F = M_1\restriction_{\bigcup Q_F}$ and $M_2^F = M_2\restriction_{\bigcup Q_F}$. Note that $M_1^F = M_0 \uplus [p_1^d]$ and $M_2^F = M_0 \uplus [p_2^d]$. Using Lemma 8, we can conclude that both M_1^F and M_2^F are reachable from M_F, i.e., $(N_F, M_F)[\sigma\rangle(N_F, M_1^F)$ and $(N_F, M_F)[\sigma\rangle(N_F, M_2^F)$.

Because (N_F, M_F) and also (N_F, M_1^F) and (N_F, M_2^F) are live and bounded free-choice nets, we can apply the Blocking Theorem (Theorem 2). There exists a unique blocking marking B_1 for the cluster involving p_1^d and T_1 and a firing sequence σ_1 that leads to the blocking marking without firing transitions in T_1: $(N_F, M_1^F)[\sigma_1\rangle(N_F, B_1)$. Note that σ_1 does not contain any transitions in the set T_d (T_1 is blocked and the rest is disabled because X_d is a P-component), i.e., $\sigma_1 \in (T_E \backslash T_d)^*$. Similarly, there is a unique blocking marking B_2 for the cluster involving p_2^d and T_2 and a firing sequence σ_2 that leads to the blocking marking without firing transitions in T_d: $(N_F, M_2^F)[\sigma_2\rangle(N_F, B_2)$ and $\sigma_2 \in (T_E \backslash T_d)^*$.

The two selected places are still marked in the corresponding blocking markings: $p_1^d \in B_1$ and $p_2^d \in B_2$. Therefore, one can write $B_1 = B_1' \uplus [p_1^d]$ and $B_2 = B_2' \uplus [p_2^d]$. Using this notation we can write: $(N_F, M_0 \uplus [p_1^d])[\sigma_1\rangle(N_F, B_1' \uplus [p_1^d])$ and $(N_F, M_0 \uplus [p_2^d])[\sigma_2\rangle(N_F, B_2' \uplus [p_2^d])$. Clearly, σ_1 and σ_2 do not depend on tokens in p_1^d or p_2^d (no transition in T_d appears in σ_1 or σ_2). Hence, also $(N_F, M_0 \uplus [p_2^d])[\sigma_1\rangle(N_F, B_1' \uplus [p_2^d])$ and $(N_F, M_0 \uplus [p_1^d])[\sigma_2\rangle(N_F, B_2' \uplus [p_1^d])$. Assume that the transitions in T_2 are not enabled in $B_1' \uplus [p_2^d]$, then $B_1' \uplus [p_2^d]$ is a reachable dead marking (because in $B_1' \uplus [p_1^d]$ only transitions in T_1 are

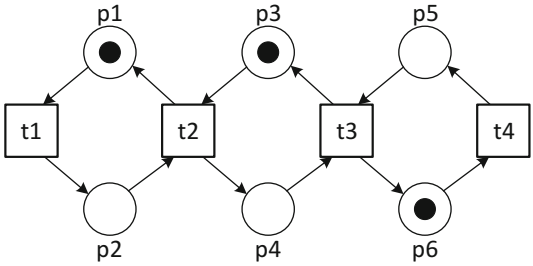


Fig. 9. A live and locally safe marked free-choice net that is not perpetual. The model is also not lucent since there are two reachable markings $M_1 = [p1, p3, p6]$ and $M_2 = [p1, p4, p6]$ that both enable $t1$ and $t4$.

enabled, the only transitions that may be enabled in $B'_1 \uplus [p_2^d]$ are the transitions in T_2). This would yield a contradiction, so the transitions T_2 need to be enabled in $B'_1 \uplus [p_2^d]$. By symmetry, we can also conclude that the transitions T_1 need to be enabled in $B'_2 \uplus [p_1^d]$ (otherwise we also find a contradiction). Hence, $B'_1 \uplus [p_1^d]$ and $B'_2 \uplus [p_1^d]$ are blocking markings for any transition in T_1 and $B'_2 \uplus [p_2^d]$ and $B'_1 \uplus [p_2^d]$ are blocking markings for any transition in T_2. Because blocking markings are unique, we conclude $B'_1 = B'_2$. Let $B' = B'_1 = B'_2$.

$B_1 = B' \uplus [p_1^d]$ is the unique blocking marking for any transition in T_1. This marking is marking all input places of T_1 and T_2 except p_2^d. $B_2 = B' \uplus [p_2^d]$ is the unique blocking marking for any transition in T_2. This marking is marking all input places of T_1 and T_2 except p_1^d. (N_F, M_F) is a perpetual marked free-choice net with some home cluster C. Hence, we can apply Lemma 5. Let $C_d = C \cap X_d$. There is one place $p_c^d \in C_d \cap P$. There exist an elementary path of the form $\langle p_0, t_1, p_1, t_2, \ldots, t_n, p_n \rangle \in {X_d}^*$ such that $p_0 = p_1^d$ and $p_n = p_c^d$ (because the added P-component is strongly connected). Suppose that the elementary path does not contain p_2^d (i.e., $p_i \neq p_2^d$ for any $0 \leq i \leq n$). (Note that $p_1^d \neq p_2^d$, $p_1^d \neq p_c^d$, and $p_2^d \neq p_c^d$.) Then there exists a firing sequence σ such that $(N_F, B' \uplus [p_1^d])[\sigma\rangle(N_F, M(C))$, $\sigma \restriction_{X_d} = \langle t_1, t_2, \ldots, t_n \rangle$, and $T_2 \cap \{t_1, t_2, \ldots, t_n\} = \emptyset$. This leads to a contradiction because B' marks input places of T_2 that cannot be removed by σ, but disappeared in home marking $M(C)$. Hence, we need to assume that $p_i = p_2^d$ for some $1 \leq i < n$. This implies that there is an elementary path of the form $\langle p_i, t_{i+1}, p_{i+1}, t_{i+2}, \ldots, t_n, p_n \rangle \in {X_d}^*$ such that $p_i = p_2^d$, $p_n = p_c^d$, and $p_j \neq p_1^d$ for all $i \leq j \leq n$. Hence, there exists a firing sequence σ such that $(N_F, B' \uplus [p_2^d])[\sigma\rangle(N_F, M(C))$, $\sigma \restriction_{X_d} = \langle t_{i+1}, t_{i+2}, \ldots, t_n \rangle$, and $T_1 \cap \{t_{i+1}, t_{i+2}, \ldots, t_n\} = \emptyset$. This again leads to a contradiction because B' marks the input places of T_1 that cannot be removed by σ, but disappeared in home marking $M(C)$.

Hence, the assumption $M_1 \neq M_2$ leads to a contradiction, showing that $M_1 = M_2$. □

Table 1. Overview of the examples used: *Marked PN* = figure showing a marked Petri net, *FreC* = free-choice, *Live* = live, *Boun* = bounded, *Safe* = safe, *LocS* = locally safe, *PC* = number of P-components, *HClu* = net has at least one home cluster, *Perp* = perpetual, *UnBM* = net has unique blocking marking for each cluster, *Lucent* = lucent, *Pls* = number of places, *Trs* = number of transitions, and *RM* = number of reachable markings.

Marked PN	*FreC*	*Live*	*Boun*	*Safe*	*LocS*	*PC*	*HClu*	*Perp*	*UnBM*	*Lucent*	*Pls*	*Trs*	*RM*
Figure 1	Yes	Yes	Yes	Yes	Yes	2	Yes	Yes	Yes	Yes	4	4	4
Figure 2	No	Yes	Yes	Yes	Yes	2	Yes	Yes	No	No	6	6	6
Figure 3	Yes	Yes	Yes	Yes	Yes	4	Yes	Yes	Yes	Yes	8	7	9
Figure 5	Yes	Yes	Yes	Yes	Yes	6	Yes	Yes	Yes	Yes	11	10	11
Figure 6	Yes	Yes	Yes	Yes	No	5	No	No	Yes	Yes	9	6	8
Figure 7	No	Yes	Yes	Yes	Yes	2	No	No	No	No	5	4	6
Figure 8	Yes	Yes	Yes	Yes	Yes	3	No	No	Yes	No	8	8	12
Figure 9	Yes	Yes	Yes	Yes	Yes	3	No	No	Yes	No	6	4	8

For the class of perpetual marked free-choice nets, markings are uniquely identified by the set of enabled transitions. As shown before, the free-choice property is needed and liveness and boundedness are not sufficient. The above theorem also does not hold for live and locally safe marked free-choice nets (see for example Fig. 8). The requirement that the net has a home cluster seems essential for characterizing marking in terms of enabled transitions.

Consider for example the live and locally safe marked free-choice net in Fig. 9. There are three P-components: $\{p1, p2, t1, t2\}$, $\{p3, p4, t2, t3\}$, and $\{p5, p6, t3, t4\}$. These always contain precisely one token. However, there are two reachable markings $M_1 = [p1, p3, p6]$ and $M_2 = [p1, p4, p6]$ that both enable $t1$ and $t4$. This can be explained by the fact that the net is not perpetual. There are four clusters, but none of these clusters is a home cluster. Note that the counter-example in Fig. 9 is actually a marked graph. This illustrates that the home cluster requirement is also essential for subclasses of free-choice nets.

5 Conclusion and Implications

We started this paper by posing the question: "What is the class of Petri nets for which the marking is uniquely identified by the set of enabled transitions?". This led to the definition of *lucency*. The main theorem proves that markings from perpetual marked free-choice nets are guaranteed to be lucent. Moreover, we showed that all requirements are needed (in the sense that dropping any of the requirements yields a counterexample). Table 1 provides an overview of the examples used in this paper. For example, even live and safe free-choice nets may have multiple markings having the same set of enabled transitions.

Other characterizations may be possible. An obvious candidate is the class of Petri nets without PT and TP handles [16]. As shown in [2] there are many similarities between free-choice workflow nets and well-structured (no PT and TP handles) workflow nets when considering notions like soundness and

P-coverability. Moreover, it seems possible to relax the notion of a regeneration point by considering simultaneously marked clusters.

Structure theory aims to link structural properties of the Petri net to its behavior. The connection between lucency and home clusters in free-choice nets could be relevant for verification and synthesis problems. The ability to link the enabling of transitions to states (i.e., lucency) is particularly relevant when observing running systems or processes, e.g., in the field of process mining [4] where people study the relationship between modeled behavior and observed behavior. If we assume lucency, two interesting scenarios can be considered:

- *Scenario 1: The system's interface or the event log reveals the set of enabled actions.* At any point in time or for any event in the log, we know the internal state of the system or process. This makes it trivial to create an accurate process model (provided that all states have been visited).
- *Scenario 2: The system's interface or the event log only reveals the executed actions.* The internal state of the system is unknown, but we know that it is fully determined by the set of enabled actions (some of which may have been observed).

It is easy to create a discovery algorithm for the first scenario. The second scenario is more challenging. However, the search space can be reduced considerably by assuming lucency (e.g., learning perpetual marked free-choice nets). Hence, Theorem 3 may lead to new process mining algorithms or help to prove the correctness and/or guarantees of existing algorithms.

Assume that each event in the event log is characterized by $e = (\sigma_{pref}, a, \sigma_{post})$ where σ_{pref} is the prefix (activities that happened before e), a is the activity executed, and σ_{post} is the postfix (activities that happened after e). The result of applying a process discovery algorithm can be seen as a function $state()$ which maps any event e onto a state $state(e)$, i.e., the state in which e occurred (see [5,7] for explanations). Hence, events e_1 and e_2 satisfying $state(e_1) = state(e_2)$ occurred in the same state and can be viewed as "equivalent". This way discovery is reduced to finding an *equivalence relation* on the set of events in the log. Given such an equivalence relation, we can apply the approach described under Scenario 1. Viewing process discovery as "finding an equivalence relation on events" provides an original angle on this challenging and highly relevant learning task.

References

1. van der Aalst, W.M.P.: The application of Petri nets to workflow management. J. Circ. Syst. Comput. **8**(1), 21–66 (1998)
2. van der Aalst, W.M.P.: Workflow verification: finding control-flow errors using Petri-net-based techniques. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_11

3. van der Aalst, W.M.P.: Mediating between modeled and observed behavior: the quest for the "Right" process. In: IEEE International Conference on Research Challenges in Information Science (RCIS 2013), pp. 31–43. IEEE Computing Society (2013)
4. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Berlin (2016)
5. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining Knowl. Discov. **2**(2), 182–192 (2012)
6. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects Comput. **23**(3), 333–363 (2011)
7. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Softw. Syst. Model. **9**(1), 87–111 (2010)
8. van der Aalst, W.M.P., Stahl, C.: Modeling Business Processes: A Petri Net Oriented Approach. MIT Press, Cambridge (2011)
9. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
10. Best, E.: Structure theory of Petri nets: the free choice hiatus. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) ACPN 1986. LNCS, vol. 254, pp. 168–205. Springer, Heidelberg (1987). https://doi.org/10.1007/978-3-540-47919-2_8
11. Best, E., Desel, J., Esparza, J.: Traps characterize home states in free-choice systems. Theor. Comput. Sci. **101**, 161–176 (1992)
12. Best, E., Wimmel, H.: Structure theory of Petri nets. In: Jensen, K., van der Aalst, W.M.P., Balbo, G., Koutny, M., Wolf, K. (eds.) Transactions on Petri Nets and Other Models of Concurrency VII. LNCS, vol. 7480, pp. 162–224. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38143-0_5
13. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, Cambridge (1995)
14. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.: Fundamentals of Business Process Management. Springer, Berlin (2013)
15. Esparza, J.: Reachability in live and safe free-choice Petri nets is NP-Complete. Theor. Comput. Sci. **198**(1–2), 211–224 (1998)
16. Esparza, J., Silva, M.: Circuits, handles, bridges and nets. In: Rozenberg, G. (ed.) ICATPN 1989. LNCS, vol. 483, pp. 210–242. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-53863-1_27
17. Gaujala, B., Haar, S., Mairesse, J.: Blocking a transition in a free choice net and what it tells about its throughput. J. Comput. Syst. Sci. **66**(3), 515–548 (2003)
18. Genrich, H.J., Thiagarajan, P.S.: A theory of bipolar synchronization schemes. Theor. Comput. Sci. **30**(3), 241–318 (1984)
19. Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Fundamentals of control flow in workflows. Acta Informatica **39**(3), 143–209 (2003)
20. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
21. Reisig, W.: Petri Nets: Modeling Techniques, Analysis, Methods, Case Studies. Springer, Berlin (2013)
22. Reisig, W., Rozenberg, G. (eds.): Lectures on Petri Nets I: Basic Models. LNCS, vol. 1491. Springer, Berlin (1998). https://doi.org/10.1007/3-540-65306-6

23. Reisig, W., Rozenberg, G. (eds.): Lectures on Petri Nets II: Applications. LNCS, vol. 1492. Springer, Berlin (1998). https://doi.org/10.1007/3-540-65307-4
24. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.: Workflow Patterns: The Definitive Guide. MIT Press, Cambridge (2016)
25. Thiagarajan, P.S., Voss, K.: A fresh look at free choice nets. Inf. Control **61**(2), 85–113 (1984)
26. Wehler, J.: Simplified proof of the blocking theorem for free-choice Petri nets. J. Comput. Syst. Sci. **76**(7), 532–537 (2010)

Tools

ePNK Applications and Annotations: A Simulator for YAWL Nets

Ekkart Kindler(✉)

Software Engineering Section, DTU Compute,
Technical University of Denmark, Kgs. Lyngby, Denmark
ekki@dtu.dk

Abstract. The *ePNK* is an Eclipse based platform and framework for developing and integrating Petri net tools and applications. New types of Petri nets can be realized and plugged into the ePNK without any programming by simply providing a model of the concepts of the new *Petri net type*. Moreover, the ePNK allows developers to customize the graphical appearance of the features of a new Petri net type.

In this paper, we discuss how to implement *applications* for the ePNK, and how they can interact with the end user by so-called *annotations*. This is discussed by the example of a simulator for YAWL nets.

Keywords: Petri net tools · Framework
Petri Net Markup Language (PNML) · Applications · Annotations

1 Introduction

The *ePNK* is an Eclipse based platform and framework for developing and integrating Petri net tools and applications. One of its core features is that the ePNK can be easily equipped with new types of Petri nets: they can be realized and plugged into the ePNK without any programming by providing a model of the concepts of the new type, the so-called *Petri net type definition* (*PNTD*) as introduced for the *Petri Net Markup Language* (*PNML*) [1,2]. Moreover, the ePNK allows developers to customize how the features of a new Petri net type are represented in the graphical editor of the ePNK.

The main idea and features of the ePNK as a tool that fully supports PNML have been presented before [3,4]. One important aspect of the ePNK, however, has not been discussed yet: ePNK *applications* and how to realize them. This, in particular, includes visualizing the result of an application on top of the Petri net in the graphical editor of the ePNK by using *annotations*, and ePNK's mechanism for an application to interact with the end user via annotations.

In this paper, we give an overview of the concepts of ePNK applications by discussing the implementation of a simulator for YAWL nets [5]. Moreover, we briefly discuss how the ePNK's annotations could be used as a basis for interchanging analysis results between different Petri net tools; this could be a good starting point for a future extension of PNML and ISO/IEC 15909 [1,2] for standardizing the exchange of analysis results of Petri nets.

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 339–350, 2018.
https://doi.org/10.1007/978-3-319-91268-4_17

2 The Result

Before we discuss how to realize YAWL nets and the YAWL simulator for the ePNK, we give a brief overview of the final result. Moreover, we use the example for briefly explaining the main concepts of YAWL nets.

Figure 1 shows a YAWL net, which is open in the graphical editor of the ePNK and with two YAWL simulator applications running on it, which are shown in the *ePNK application view* at the bottom. In this view, one simulator, called YAWL Simulator 1, is selected as the *active* one. The current marking of that simulator is shown in the graphical editor by additional annotations in the YAWL net, and some additional information is high-lighted: A blue number at the top-right of a place indicates the number of tokens in the current marking of the net in the active simulator; if there is no such number at a place, the place has currently no tokens. A blue overlay for a transition indicates that the transition is currently enabled. The red overlays for places, transitions and arcs indicate a path on which a token might arrive on place c4 at the enabled OR-join transition a6. This serves as a warning for the user that transition a6 should not be fired yet, since an OR-join transition should wait for tokens that might still arrive along the red path[1].

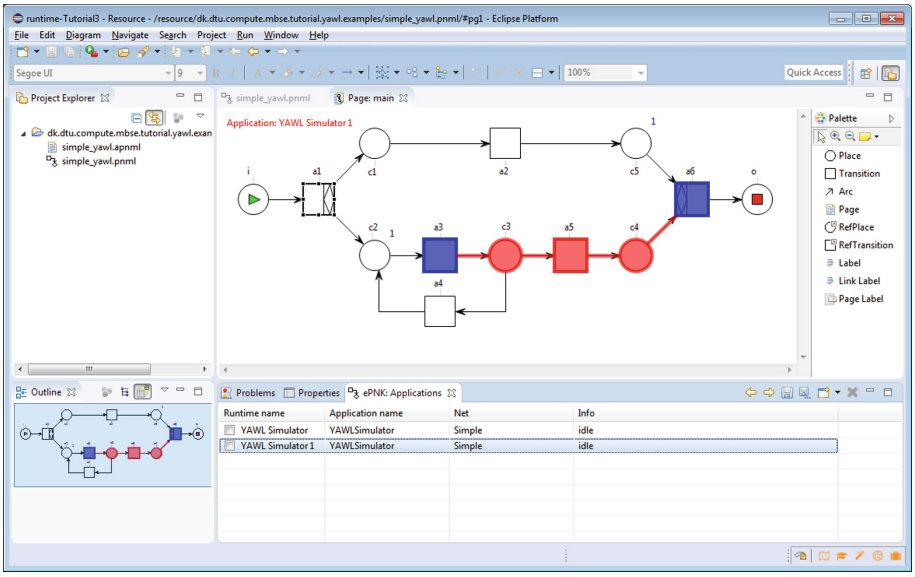


Fig. 1. ePNK with two YAWL simulators running (Color figure online)

In Fig. 1, you can also see some specific features of YAWL nets. We use this example for briefly explaining the main concepts of YAWL. Like normal Petri

[1] This is a subtlety of YAWL OR-joins, which we do not discuss in detail in this paper.

nets, YAWL nets consist of *places*, *transitions* and *arcs*, where arcs can connect places and transitions. In YAWL, places are called *conditions*, and transitions are called *actions*. YAWL has distinguished *start conditions* and *finish conditions*, which is a special attribute associated with conditions; and these conditions are graphically distinguished by special symbols as shown for start condition `i` and finish condition `o`. YAWL nets, actually, require that there is exactly one start condition and one finish condition. Moreover, YAWL actions can define different ways, how an action with multiple incoming arcs synchronizes incoming tokens, called *AND-*, *OR-* and *XOR-joins*; we call this the *join type* of the action. Likewise, a YAWL action with multiple outgoing arcs can define different *split types*, called *AND-*, *OR-* and *XOR-splits*, which define how many outgoing arcs of the action should produce a token when the transition fires. The join and split types are two attributes of an action, which also is indicated in the graphical representation of the action. For example, `a1` is an OR-split action, and `a6` is an OR-join action indicated by the diamond symbol on the right or left side of the respective action. YAWL nets have some additional features like *reset arcs*. But, for simplicity, our example from Fig. 1 does not use reset arcs.

Once a YAWL simulator application is running on a YAWL net and selected as the active one in the ePNK *application view*, the user can interact with the simulator by double clicking on the enabled transitions. This will fire the respective transition and then show the successor marking. Moreover, the user can go back and forth to the previous or next marking by pressing the backwards and forwards button in the toolbar of the *application view*. From the *application view*, it is also possible to save the current state of the simulator with the save button and to start new applications via the drop down menu, which will show all the ePNK applications that are applicable for the net in the currently active editor. Moreover, the user can shut down applications or load an application from a state that was saved earlier.

3 ePNK Applications: Overview

The ePNK comes already with different Petri net types and simulators for the different net types. But, the main purpose of the ePNK are not the Petri net types and the applications it comes with, but the possibility to create and plug in new Petri net types and new applications.

Basically, an ePNK application is some additional software on top of the ePNK, which is started on a Petri net, which can show some results or intermediate states with some graphical overlays on top of the graphical editor of the underlying net and interacts with the end user via these overlays and the buttons in the ePNK application view as discussed for the YAWL simulator in Fig. 1. The only limitation is that each application is associated with one Petri net only—unless you are willing to do major programming.

Typically, your own extension of the ePNK would be done in two steps. First, defining the concepts of your Petri net type by a *Petri net type definition* (*PNTD*), which is discussed in more detail in Sect. 4; you can also customize

the graphical appearance of the elements of your Petri net type in the graphical editor. Note that the definition of a new Petri net type is purely syntactic: it defines which objects may or may not be in this Petri net and how the elements may be related to each other. The Petri net type definition does not define the semantics or the firing rule of the new type. This would be defined along with the applications, in particular for simulators.

If you use an existing Petri net type, either because it comes with the ePNK or is provided by some third party, you can, of course, skip the first step.

The second step, would be defining a new *ePNK application*, which consists of two sub-steps. The first one would be defining the runtime information of your application, which makes up the state of the application. This is done by defining *annotations*, which is discussed in Sect. 5.1. The second sub-step is defining handlers, which define how the annotations of the runtime information are presented to the end user, and which actions should be taken, when the end user interacts with an annotation. This is discussed in Sect. 5.2. The semantics of a Petri net type is actually defined in these actions.

4 The YAWL PNTD

In this section, we briefly discuss how to define a new Petri net type. The concepts of a Petri net type are defined by providing a class diagram[2]. This is called a *Petri net type definition* (*PNTD*). Figure 2 shows the PNTD for YAWL nets. Note that the new concepts, which are shown in light colour, extend the concepts from the *PNML core model*, which are shown in magenta on the left-hand side and the class Attribute. YAWL's *Conditions* are derived from *Places* of the *PNML core model*. Conditions have an additional *attribute* type, saying whether it is a *normal*, a *start* or a *finish* place. *Actions* are derived from *Transitions* of the *PNML core model*; they have two additional attributes, defining the *join-* and the *split-type* of the transition: these attributes can have values *AND*, *XOR* and *OR*. Likewise YAWL's *Arcs* extend the *Arcs* from the *PNML core model*: YAWL arcs have one additional attribute defining the type of the arc, which can be *NORMAL* or *RESET*.

In addition to the concepts defined in the model, there are some *constraints*. Generally, constraints are used to express subtle restrictions on the combination of objects or the legal values of attributes that would be difficult or inconvenient to express in class diagrams. In UML, these are often expressed in OCL. For our example, there is a constraint saying that a YAWL net must have exactly one start and exactly one finish place; an other constraint is that a start place should not have incoming arcs; and a finish place should not have outgoing arcs—and some more. In the EMF technology underlying the ePNK [6], these additional constraints can be formulated either in OCL or programmed in Java. But, we do not discuss the technical details here. If you are interested, you can look them up in the plugin project for YAWL nets (see Sect. 8).

[2] Technically, it is an Ecore model, which is kind of a light-weight version of UML class diagrams used by EMF [6].

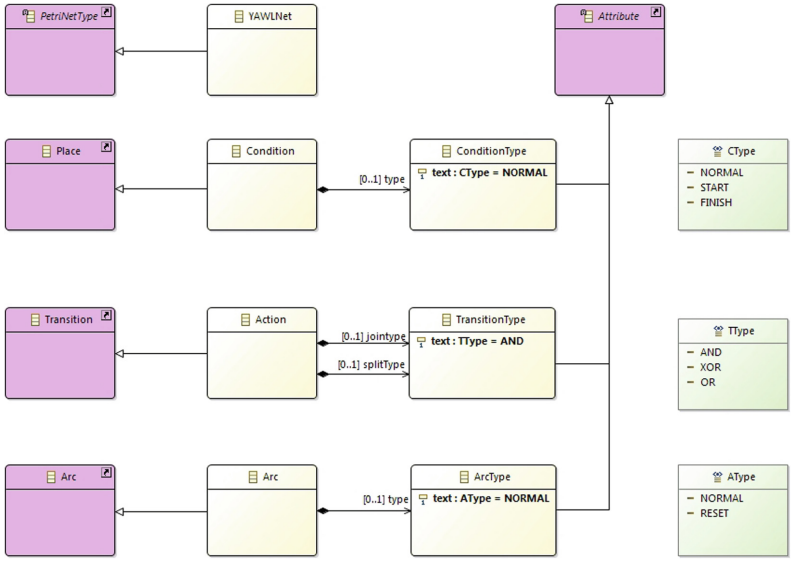


Fig. 2. PNTD for YAWL

In contrast to earlier versions of the ePNK, which required some minor programming, version 1.2 of the ePNK does not need any programming for plugging in a new PNTD; the model from Fig. 2 and the code generated from it by EMF [6] can be plugged directly into the ePNK.

In addition to the PNTD, we would also like to customize the graphical appearance of *start* and *end places*, *reset arcs* and the *split-* and *join-types* of transitions so that they look like YAWL nets. For each object type, we, basically, need to program a class with a method that draws the respective object dependent on its context and its different attributes; and then, plug these classes into the ePNK. Then, the graphical editor of the ePNK is able to show YAWL nets as shown in Fig. 1. You will find the details in the implementation of the YAWL plugin projects (see Sect. 8).

5 The YAWL Simulator

Next, we discuss how to realize a simulator for YAWL nets as an example of how to realize an ePNK application.

5.1 The Annotations

The first step is to define the *runtime information* of the application. Of course, it depends on the specific application what constitutes this *runtime information*. For our YAWL simulator, this runtime information is the current marking of the net along with the sequence of all markings up to the current one. Technically,

the runtime information of an ePNK application is defined by *annotations*, which are associated with the net itself and with the net's objects.

Figure 3 shows the class diagram defining the annotations for the YAWL simulator. This diagram consists of three parts: The two classes PetriNet and Object on the left come from the *PNML core model* [1] and represent the Petri net and its different kinds of objects. These are the objects which are supposed to be annotated. The classes at the top in dark orange, represent the general annotations that are built into the ePNK. These are extended by the annotations of a specific application, which, in our case, are the three classes at the bottom.

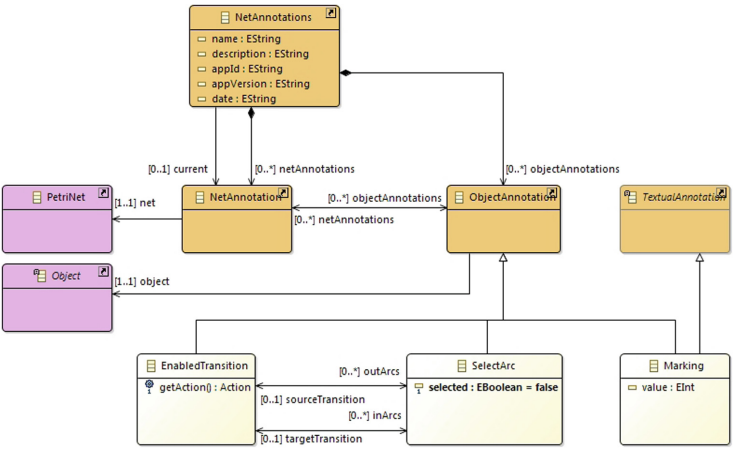


Fig. 3. YAWL annotations (Color figure online)

Let us have a closer look at these concepts: The two concepts PetriNet and Object represent *Petri nets* along with their *objects*: places, transitions, and arcs; but also pages, reference transitions and reference places. As said already, these are the objects annotations refer to. The orange classes at the top are ePNK's built-in concepts of annotations. An ObjectAnnotation annotates exactly one object, which is represented by the reference object. Note that a Petri net and its objects do not know anything about their annotations at all, since applications should be detached from the net they are running on. A NetAnnotation refers to one Petri net and consist of many object annotations. These constitute a specific situation in the application at runtime; in the YAWL simulator, this would be a marking. The class NetAnnotations comprises all the annotations of a running application, a sequence of net annotations; one NetAnnotation is pointed out as the application's *current* one, which is the one shown with labels and overlays on top of the graphical editor of the net, when the application is active. In our YAWL simulator, this would be the current marking. An ePNK application is actually associated with exactly one NetAnnotations object, which in turn contains all the application's relevant runtime information in its associated sequence of NetAnnotation objects. The interpretation of this sequence is

up to the concrete application, but the default interpretation is a sequence of net annotations. In our YAWL simulator, it represents a sequence of markings, with one distinguished as the current one. The class NetAnnotations has some additional attributes, which are relevant when an application—actually its state—is saved. In particular, the appId is used for identifying the application from which the annotations have been saved; this is used for starting the same application again when the end user loads the application from its saved state.

The three classes at the bottom of Fig. 3 define the annotations for our YAWL simulator: EnabledTransition, SelectArc and Marking, which we have seen representations of in Fig. 1 already. Note that we do not define specific annotations for high-lighting the path of possible tokens arriving at an OR-join here, since we can reuse ObjectAnnotation for this purpose, which is defined by the ePNK already. The ePNK annotation model has an abstract class TextualAnnotation. An ePNK application, by default, presents annotations inheriting from TextualAnnotation as textual labels at the top-right of the object in the graphical editor, showing the value of its text or value attribute. In our YAWL simulator, Marking is an example of such a textual annotation. All other annotations are, by default, shown as red overlays of the respective object. But, we will see later that an application can change this. The annotation SelectArc is used for indicating which arcs of an enabled transition can be selected by the user, and which of the arcs are currently selected, represented by the attribute selected. This is relevant for the arcs of XOR-join and -split transitions and for OR-split transitions in order for the end user to be able to select from which places tokens are to be consumed and on which places tokens should be produced when the transition fires. To this end, the user will be able to select a combination of arcs for the respective transition (the selected ones shown in blue, the not selected ones shown in grey). In order to program the logic of the arc selection, the SelectArc annotation is associated with the respective EnabledTransition annotation.

Altogether, the classes from Fig. 3 allow a running simulator to store its state with a NetAnnotations object as its root. We call this the *runtime information* of the simulator. Note, that this can be much more than what is currently shown to the end user; only the current NetAnnotation is shown to the end user.

Associating exactly one NetAnnotations object with each application makes it easy for the ePNK to save the runtime information of an application to a file. And the ePNK can load this file again, and start the corresponding application, since the application's id is stored as an attribute of the top-level NetAnnotations object. This way, a developer of an ePNK application does not need to program anything for realizing the load and save feature of the application. By overriding the isSavable() method, the application can decide whether it should be possible to save its runtime information or not.

This mechanism makes it possible to exchange the runtime information of applications among different tools, as long as they agree on the annotation models, the underlying PNML core model, and on the way the instances of the annotation model are serialized. The ePNK currently uses XMI [7] for this purpose. But, it would be possible to define a dedicated XML-format that would closely

resemble the mapping of the PNML core model to XML. This way, we have a format for interchanging analysis results of Petri nets among different tools. Of course, this helps exchanging the information on a technical or syntactic level only. The conceptual work of defining the meaning of the different annotations and devising standard annotations for the most relevant ones would be up for discussion in the Petri net community, which then could lead to an extension of the ISO/IEC 15909-2 standard [2].

5.2 The Application

In addition to defining its runtime information, an application must implement three different things: the *initialization*, some *presentation handlers*, and some *action handlers*. The *initialization* computes the initial net annotations that represent the initial state (the initial marking in our case); the *presentation handlers* define how the different object annotations should be presented to the end user as labels or overlays in the graphical editor, if the ePNK's default representations are not sufficient; the *action handlers* define what should happen when the end user interacts with an annotation (or actually its presentation) in the graphical editor.

Separating the definition of the runtime information, the presentation handlers and the action handlers in applications follows the architectural pattern of *model-view-controller* (*MVC*).

Here, we cannot discuss all the details of implementing the action handlers and the presentation handlers, for which you can have a look into the YAWL plugins (see Sect. 8). But, we give a brief overview here. The YAWL simulator has two action handlers: one for firing the transition when the end user double clicks on an enabled transition annotation, and one for selecting or unselecting an arc when the user clicks on a select arc annotation. The *enabled transition* handler, basically, adds a new net annotation to the state of the simulator with the annotations representing the new marking, and it makes this new net annotation the current one. This is where the actual semantics of YAWL is implemented. The *select arc* handler, basically, toggles the selected attribute taking the semantics of the respective split or join into account.

A presentation handler, basically, returns an overlay figure for the graphical figure that represents the annotated object in the graphical editor. In our case, it returns a blue overlay for an enabled transition and, dependent on the selected attribute of the SelectArc annotation, a blue or grey overlay for the annotated arc. For all other annotations, ePNK's default presentation handler kicks in, returning a red overlay.

6 Discussion

In the previous sections, we have discussed the main steps of realizing YAWL nets and a simulator for them. In both parts, models played a major role: the concepts of YAWL nets are represented in the model shown in Fig. 2; likewise,

the model of the runtime information is shown in Fig. 3. Based on the model of the runtime information, an ePNK application is realized following the MVC-pattern: the runtime information is represented in the model as annotations, the presentation handlers define how the annotations should be shown on top of the net in a graphical editor, and the action handlers define the controllers.

Defining the runtime information of an application by a model has an additional advantage: It allows the ePNK to generically save and load the state of the application without any additional programming by the developer. This could be the basis for an interchange format of analysis results, as discussed in the end of Sect. 5.1.

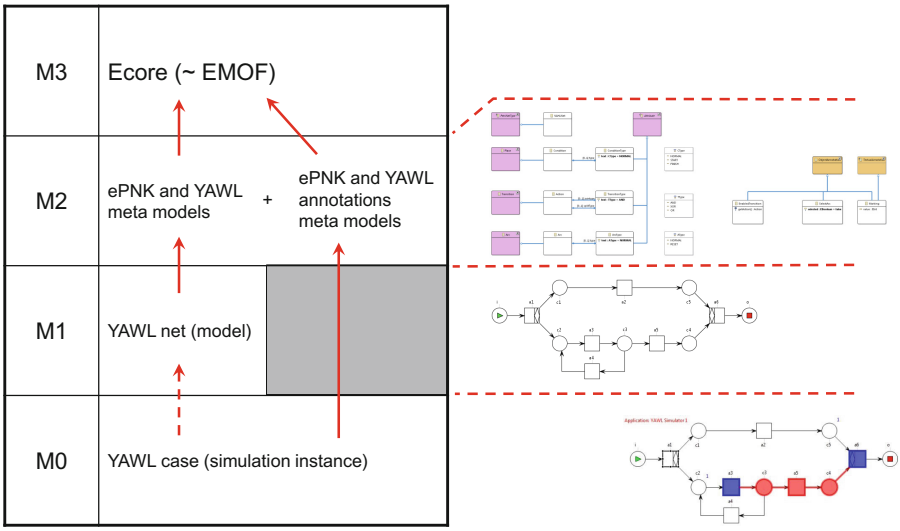


Fig. 4. MOF levels: YAWL nets and simulator (Color figure online)

Figure 4 gives an overview of the different models and meta models involved in defining Petri nets and the annotations used for applications, as well as the different instances of these models. It shows the models, meta models and instances on the levels of the so-called MOF hierarchy [8]. On level M2, there are the PNML core model, which is also the meta model of the ePNK, and the meta model for the PNTD of YAWL—which together define what YAWL nets are. On level M1, there are the YAWL nets which are instances of the meta models on M2, which is indicated by the red arrow. On level M2, there are also the meta models of the runtime information (annotations) of the applications. An instance of that meta model would be a running simulation, a *case* of a YAWL net, with its current marking; this is shown on level M0. Note that, conceptually, a case is an instance of a YAWL net on level M1, indicated by the dashed red arrow; but technically, a YAWL case is an instance of the annotation meta models on level M2. Therefore, the *technical instance* jumps one level. Note, in particular, the difference

between the solid red arrows that represent a *technical is instance* relation and the dashed red arrow that represents a *conceptual is instance* relation. On level M3, is the meta model defining the concepts of Ecore diagrams, which are used to formulate the PNTD of YAWL and the annotations for the YAWL simulator.

7 Related Work and Limitations

The ePNK was developed for fully exploiting the concepts of PNML [1] and, in particular, the concept of *Petri net type definitions*. There are many tools that support PNML in some form or the other (a few of them are listed on the PNML Home page [9]). But, most tools support only a fixed set of Petri net types, and it is not easily possible to define a new one. The PNML Framework [10] is a notable exception, in that it is made for defining new Petri net types by using PNTDs. The limitation of the PNML Framework, however, is that the complete code of the PNML Framework needs to be recompiled for a new PNTD. The ePNK allows plugging in new PNTDs without recompiling the ePNK itself. Moreover, the PNML Framework does not have a graphical editor for Petri nets.

There are several Petri net and Petri net related tools, which allow to plug in new functionality, which we could call applications in our context. Some examples are CPN Tools [11], ProM [12], and Renew [13]. ProM is explicitly made for plugging in new process mining techniques and also allows implementing new notations, but not in the sense of PNTDs. CPN Tools and Renew mostly use a fixed Petri net type (coloured nets and reference nets, respectively). But they allow to plug in new applications. And it is possible to give feedback in the respective graphical editor.

So, the ePNK with its plugin mechanism for easily defining new Petri net types as well as applications fills a very special niche: it is easy to define an application on a new or slightly extended version of Petri nets. Concerning the definition of new notations or new Petri net types, the limitation of the ePNK is that the new notation needs to be a "kind of a Petri net" since it needs to extend the PNML core model by design: so the new notation should have two cardinal types of nodes (place-like and transition-like nodes); of course, the ePNK could be abused, since it is possible to squeeze in many different kinds of nodes and arcs by additional attributes for places and transitions and arcs. But, we would consider that to be artificial. The example of YAWL nets, however, shows that notations that have nodes of many different kinds and with attributes can easily represented as a PNTD, if they fall into two major categories.

What concerns applications, the limitations are that the main information of an application is supposed to be shown on top of the graphical nets. Of course, an application can implement additional views—an example would be the ePNKs simulator for high-level nets, which has a view for the complete history of the simulation. But, that requires much more programming and defies to some extent the original idea of the ePNK.

We believe that within these limits, the ePNK will be of use for people who would want to quickly experiment with an idea for a new Petri net type

or a simple application. They would get a graphical editor and the graphical presentation of results of their application and the interaction with the end user, basically, for free. For us, the ePNK came in handy when we needed a notation for defining the life-cycle of objects in the context of the *Event Coordination Notation* (*ECNO*) [14]: We could very quickly define ECNO nets by a Petri net type definition for the ePNK. From these models plus ECNO's coordination model, running software could be generated fully automatically.

8 Technical Details

The ePNK is realized as an extension of Eclipse. In order to install it on your computer, you need to install Java and Eclipse[3]. Then, you can install the ePNK from inside Eclipse (Help→Install New Software ...). Version 1.2 of the ePNK can be installed by creating a new update site http://www2.compute.dtu.dk/~ekki/projects/ePNK/1.2/update/ and then selecting all ePNK features. The update site includes a "YAWL net type, graphical extension and simulator" feature, which includes all YAWL extensions discussed in this paper as well as some YAWL example nets.

These projects come with the complete source code, so that you can look up the technical details and implementation issues, which we could not discuss in this paper. To this end, open the "Plug-ins" view of Eclipse (Window→Show View→Other...) and, in this view, select all YAWL projects, which have the prefix dk.dtu.compute.mbse.tutorial.yawl; right-click on them and select Import As→Source Project. After that, you will have five new projects in your Eclipse workspace.

The project dk.dtu.compute.mbse.tutorial.yawl.examples contains three PNML files with YAWL nets—among others, the file simple_yawl.pnml, which is the example shown in Fig. 1. You can open it by double-clicking on it. Note that the file opens in the ePNK's tree editor; you can fold out the document's objects until the pages appear, and then open the graphical editor by double-clicking on the page. Then, you should open the "ePNK: Applications" view as discussed above for the "Plug-ins" view. From there, you can start a new YAWL simulator via the drop down menu; from there, you can also load the example simulation (simple_yawl.apnml) discussed in this paper by "Load Application" and then navigate back and forth in this simulation.

For looking up technical details of the implementation, you can look into the other projects: dk.dtu.compute.mbse.tutorial.yawl contains the definition of YAWL nets. dk.dtu.compute.mbse.tutorial.yawl.graphics implements the custom graphics for YAWL nets, and dk.dtu.compute.mbse.tutorial.yawl.simulator implements the YAWL simulator. The last project, dk.dtu.compute.mbse.tutorial.yawl.edit contains automatically generated code only.

[3] If you intend to use the ePNK for developing own Petri net types or applications, it is recommended to install the "Eclipse Modeling Tools" package of Eclipse.

9 Conclusion

By the example of YAWL nets and the YAWL simulator, we have discussed the main idea of ePNK applications and how they are realized based on the ePNK's annotation model. Like the definition of YAWL nets themselves, the runtime information of the simulator is defined by a model as an extension of ePNK's annotations model. Using models not only for defining new types, but also for representing the runtime information, makes it easier to load and save the state of applications in a uniform and generic way. This could actually be used as a basis for a generic but yet extensible standard interchange format for analysis results of Petri nets as an extension of PNML [1,2].

References

1. Hillah, L., Kindler, E., Kordon, F., Petrucci, L., Treves, N.: A primer on the Petri net markup language and ISO/IEC 15909–2. In: Jensen, K. (ed.) 10^{th} Workshop on Coloured Petri Nets, CPN 2009, pp. 101–120 (2009)
2. ISO/IEC: systems and software engineering - high-level Petri nets - part 2: transfer format, International Standard ISO/IEC 15909–2:2011 (2011)
3. Kindler, E.: The ePNK: an extensible Petri net tool for PNML. In: Kristensen, L.M., Petrucci, L. (eds.) PETRI NETS 2011. LNCS, vol. 6709, pp. 318–327. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21834-7_18
4. Kindler, E.: The ePNK: a generic PNML tool - users' and developers' guide for version 1.0.0. Technical report IMM-Technical report-2012-14, DTU Informatics, Kgs. Lyngby, Denmark (2012)
5. van der Aalst, W., ter Hofstede, A.: YAWL: yet another workflow language. Inf. Syst. **30**(4), 245–275 (2005)
6. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework. The Eclipse Series, 2nd edn. Addison-Wesley, Boston (2006)
7. OMG: XML metadata interchange (XMI) specification, version 2.0. Technical report formal/03-05-02, The Object Management Group, Inc. (2003)
8. OMG: Meta Object Facility (MOF) specification, version 1.4.1. Technical report formal/05-05-05, The Object Management Group, Inc. (2005)
9. PNML Team: PNML.org: the Petri net markup language home page. http://www.pnml.org/
10. Hillah, L.M., Kordon, F., Petrucci, L., Trèves, N.: PNML framework: an extendable reference implementation of the Petri net markup language. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 318–327. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13675-7_20
11. CPN Tools: home page. http://cpntools.org/
12. PRoM Tools: home page. http://www.promtools.org/doku.php
13. Kummer, O., Wienberg, F., Duvigneau, M., Schumacher, J., Köhler, M., Moldt, D., Rölke, H., Valk, R.: An extensible editor and simulation engine for Petri nets: Renew. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 484–493. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27793-4_29
14. Kindler, E.: Coordinating interactions: the event coordination notation. Technical report DTU Compute Technical report 2014–05, DTU Compute, Kongens Lyngby, Denmark (2014)

Petri Net Model Checking with LoLA 2

Karsten Wolf(✉)

Institut für Informatik, Universität Rostock, Rostock, Germany
karsten.wolf@uni-rostock.de

Abstract. LoLA 2 offers a suite of algorithms for verifying place/transition Petri nets. It combines structural with state space methods and general purpose with Petri net-specific techniques. The methods are easily accessible to people with little knowledge of Petri nets since there is a uniform query language based on temporal logic, and the tool takes care of sound application of its methods. Unlike its predecessor LoLA 1, LoLA 2 is based on a strict modularisation and integration of various standard tools. A careful software engineering approach has been used for coding. Through its code quality and its frequent comparison to other tools in the yearly model checking contests, LoLA 2 has become one of the most reliable verification tools for distributed systems.

1 Introduction

Work on LoLA started in 1997. Originally, the intention was to have just enough code for experimental validation of new state space reduction methods [20,32–34]. This code collection was presented as LoLA 1.0 in [35].

LoLA has been applied in various case studies stemming from different domains. It proved to be useful for finding hazards in asynchronous circuits [37], for validating and comparing Petri net semantics of web service modeling languages [25], for analysing interacting web services [23,24], for ontology-based service composition [28], in the automotive area [27], and for self-adaptive systems [10].

Several researchers used LoLA as a reference representing the state of the art in Petri net verification. They compared the performance of LoLA with their own, domain specific tool to justify their algorithms. Examples for this approach include the verification of parameterised Boolean programs [16], verification of multiprocessor code [2], or new ideas for CTL model checking [6].

Several tools integrated LoLA or offer an export to the LoLA input format. This way, the technology implemented in LoLA is available in their particular frameworks. An incomplete list includes tools for design, analysis and simulation such as Snoopy [13] from Cottbus and Renew [22] from Hamburg, packages that try to integrate analysis and synthesis of nets such as APT [1] from Oldenburg and Travis [26] from Hagen, tools in the area of business process management such as ProM [42] or Oryx [3], or for the exploration of biochemical reaction chains such as the Pathalyzer tool [7].

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 351–362, 2018.
https://doi.org/10.1007/978-3-319-91268-4_18

As far as these activities did not directly include members of our group, they typically did not require any consulting from our side. This underpins the ease of use and integration of LoLA. However, as the code grew further, several initial design decisions turned out to be less than optimal. In particular, tool control via a configuration file (with the necessity to re-compile LoLA for every use) was good enough for the initial purpose of LoLA but not very comfortable for being integrated into other tools. Additionally, the poor code structure inhibited the further development via student projects.

Consequently, we decided to completely re-implement LoLA. LoLA 2.x was designed from the very beginning as a tool targeting two use cases. First, we understand LoLA as a community service making state-of-the-art verification technology publicly available thus taken the burden off other scientists to implement Petri net verification methods themselves. Thus, LoLA is designed such that integration into other tools is as simple as possible. Second, LoLA remains our experimental platform for new verification algorithms. This way, we hope that we can keep LoLA competitive in the future.

Code quality was a core issue for LoLA 2. Consequently, we completely re-implemented the tool and did not re-use any piece of code of LoLA 1. First, LoLA 2 got a clear modular structure, benefitting from the experience with LoLA 1. Second, we used a systematic software engineering approach that includes continuous integration management, frequent code reviews and broad discussions on major design decisions. Test case coverage is close to 100% in the core parts of the tool (some code, e.g. reaction to exceptions in the interaction with the operating system, cannot be covered by test cases). In addition, participation in the model checking contest [8] (LoLA is the only tool that participated every year so far) is an excellent platform for adding trust into the produced result. In 2017, results of LoLA were overwhelmingly consistent with the results of other tools, in some categories even 100%. Remaining issues typically concern fresh algorithms and the (in some cases quite involved) translation of PNML and XML input into the input format of LoLA.

In the sequel we shall give an overview on the offered functionality and the concepts for integration. Then we survey the architecture of LoLA 2, briefly surveying the core modules. Finally, we report on use cases and success stories.

2 Installation and Usage

LoLA 2 can be downloaded from www.service-technology.org and is installed using the `automake` procedures. Petri net input can be generated using an ASCII editor, or by modeling tools that offer an export to LoLA. Then LoLA is called on the command line of any UNIX (LINUX or MACOS) terminal where command line options control the property to be verified as well as the technology to be applied. Results appear on the screen or in a file. The package includes a user manual that describes in detail the installation procedure, the file formats, the output, and the options of LoLA 2.

3 Supported Properties

Generally, LoLA 2 can operate in *full*, *none*, or *model checking* mode. In *full* mode, it just computes a (complete or reduced) state space, without investigation of a property. This might be useful for analysing the impact of a verification technique. In *none* mode, it just pre-processes the net, without any state space generation. This way, the user might gather structural information on the net that is calculated in LoLA 2 prior to state space exploration. The most important mode for using LoLA 2, however, is the *model checking* mode. The user formulates a query (on the command line or in a file). The query language permits the specification of a bound expression or a formula in the temporal logic CTL* [5].

A bound expression is a formal sum using places as variables. LoLA 2 computes the maximum value that this expression can get in any reachable marking. A CTL* formula is first pre-processed based on an integrated term rewriting system. Processing aims at

- Removing syntactic sugar (e.g. replacing implication by disjunction);
- Detecting logical tautologies and contradictions in sub-formulas;
- Separating the formula into as many as possible subproblems that are connected via Boolean operations only;
- Pushing the subproblems into any of the fragments LTL or CTL of CTL*.

Looking for tautologies may appear to be odd, but actually we believe that both place/transition nets and their properties are typically the result of a systematic and partly automatic translation process from other kinds of specification. The translation procedures are not necessarily optimised for getting rid of tautologies.

For the resulting Boolean combination, each subproblem is then categorised into one of the following classes:

- Initial satisfaction (can be decided just by inspecting the initial marking);
- Reachability of deadlocks or deadlock freedom;
- Reachability or invariance of a state predicate;
- TSCC based property ($AG\ EF\ \phi$, $EF\ AG\ \phi$, $AG\ EF\ AG\ \phi$, $EF\ AG\ EF\ \phi$)
- LTL property;
- CTL property;
- CTL* property.

If a property falls into the CTL* category, LoLA cannot verify it and terminates with result *unknown*. For each category, LoLA offers specific verification techniques and specific variants of general techniques. For all classes, state space exploration is available with a category-specific version of the stubborn set method (deadlock: [40], reachability: [32], LTL: [41], CTL: [12]) and a property preserving version of the symmetry method [33,34] being available. For the two reachability categories, we further offer the sweep-line method [19] (with automatic calculation of the progress measure [21]), a random search [31], and

specific structural techniques based on Commoner's theorem [29] and the state equation [44]. For TSCC based properties, a specific search algorithm to find the terminal strongly connected components (TSCC) of the net is used. This category contains Petri net standard properties such as liveness and reversibility.

The atomic propositions in LoLA 2 comprise place based properties such as $p_1 + 2 \cdot p_2 \geq 13$, transition based properties such as FIREABLE(t_3), and the global properties INITIAL and DEADLOCK. For being able to capture boundedness, we introduced a constant oo representing ∞. So, unboundedness of place p_7 can be specified as $EF\ p_7 \geq oo$. For such properties, LoLA can construct the coverability graph instead of the reachability graph.

The set of properties that can be specified and analysed in LoLA 2 is significantly larger than in LoLA 1. LoLA 1 did not support formal sums of places, nor DEADLOCK, nor INITIAL, nor the boundedness properties. Categorisation was left to the user thus requiring more knowledge on Petri nets.

If more than one verification technique is available, LoLA 2 runs a portfolio approach. That is, the user can choose to run several methods sequentially or in parallel. The first algorithm that returns a value different from *unknown* determines result and run-time.

4 Integrating LoLA 2

As its predecessor, LoLA 2 is purely command-line oriented. It can process inputs from the UNIX standard input stream and produce results on the standard output. Alternatively, appropriate files can be used. Petri nets are specified in a language that is machine readable and permits, (to our own taste better than PNML [9]) the manual generation of LoLA input for small models. Translation from PNML to the LoLA input is available as a helper tool that is shipped with the LoLA 2 distribution. Results are presented in human readable form on the terminal (together with data collected during computation and status information). Additionally or alternatively, output can be organised according to the Javascript Object Notation (JSON) format which is very convenient for further computer aided post-processing. For supporting the execution of LoLA 2 on remote servers, the output can be broadcasted via the UDP protocol. A listener tool that is part of the distribution can then catch the broadcasted messages and display them. It can also send messages to LoLA 2 forcing it to terminate. With all these features, LoLA 2 supports being run in complex scripts, or being remotely called from other tools.

5 Architecture of LoLA 2

LoLA 2 is strictly modularised thus making it easy to locate the right place for adding code. In the sequel, we shall briefly survey the most important modules.

Parsing. Using the `bison` and `flex` standard tools, input files are transformed into a syntax tree. We use the term processor `kimwitu++` for post-processing

the syntax tree. Kimwitu is able apply rewriting rules and to systematically traverse the tree, for instance for final generation of the internal data structures. We experienced that managing the rewriting and traversing rules of Kimwitu is much more convenient, less error-prone, and easier extensible than the manual implementations we used in LoLA 1.

Net. Places, transitions, and arcs used to be objects in LoLA 1. In LoLA 2, places and transitions are just indices. Information on a place or transition is found under that index in big arrays. This way, retrieving information on the actual Petri net boils down to the traversal of an array rather than traversal of a linked list. This leads to a more "cache friendly" access to the net. Additionally, on a 64 bit architecture, we may still work with 32 bit numbers for representing the net while pointers and object references would consume 64 bits each.

Preprocessing. The data computed here help us to speed up subsequent state space exploration. We explicitly store, for each transition, arrays containing the pre-set and post-set, respectively. This way, we can implement enabledness check as well as transition occurrence very efficiently. We further store, for every transition t, the set of conflicting transitions $(\bullet t)\bullet$ which is frequently needed in stubborn set calculations. Also for stubborn set calculations, we compute the set of conflict clusters using Tarjan's union/find algorithm [39]. For a net $[P, T, F]$ with set of places P, set of transitions T, and set of arcs $F \subseteq (P \times T) \cup (T \times P)$, a conflict cluster is a class of the partition generated by the symmetric, reflexive, and transitive closure of $F \cap (P \times T)$. Last but not least, we gather information on place and transition invariants to be used for saving memory.

Formula. The formula is internally stored as a tree. The module contains procedures for evaluating and updating state predicates (the temporal parts of the formula are evaluated during the actual state space exploration). Evaluation determines the value in the initial state. Updating computes the impact of a fired transition t occurrence to the formula value. Here, we exploit locality (only subformulas related to t, $\bullet t$, and $t\bullet$ are re-evaluated) and linearity (for instance, a predicate "$p > 1$" is only re-considered if it is false and $p \in t\bullet$, or it is true and $p \in \bullet t$). Necessary dependencies between formula and net structure are calculated during pre-processing. This way, we again save a lot of run-time (given that we need to update a formula millions of times during state space exploration). The module further contains procedures for transforming a state predicate into disjunctive normal form. This is a prerequisite for applying the state equation to reachability problems. Since the formula may explode during this construction (something that frequently occurred in the model checking contests), we have taken this transformation out of the Kimwitu term processor. Additionally, we have implemented an abstract interpretation approach to the sub-formulas. Using that approach, we are able to detect duplicate formulas and can thus alleviate the formula explosion during normalisation.

Planning. This component gathers the information from the categorisation of the property and the command line options. It determines the verification workflow and configures several modules (exploration, firelist, encoding, store) such

that the property under investigation is preserved. For a reachability property, the workflow may consist of a sequential or parallel execution of state space exploration, random walk exploration, and structural verification (siphon/trap property or state equation). For parallel execution of several methods, we use threads. The siphon/trap property is coded as a Boolean formula and shipped to the Minisat SAT checker [4]. The state equation is explored through a call to the tool Sara that extends the evaluation of the state equation with an abstraction refinement approach [44]. The planning component also schedules the verification of more than one property, This may happen if the user specifies more than one property in the command line, or if the property is a Boolean combination of sub-problems which we evaluate separately. As every sub-problem may require state space exploration, we do not support parallel execution in this case as we want to avoid competition for available memory. For supporting more than one state space exploration, we had to solve a severe problem. Since a state space exploration may generate millions of states, we would need to release millions of data objects which may consume a measurable amount of run-time. To avoid this, we clone the process that runs LoLA using the UNIX `fork` command which generates a copy of the process and its whole memory image. Thus, the cloned child process already has all the parsed and pre-processed data and can proceed as if it would execute the first state space exploration. In the end, it communicates its result to the parent process and terminates. Using this mechanism, switching from one subproblem to another takes virtually no time.

Exploration. This is actually a collection of different state space exploration methods. These include

- Simple depth first search for deadlock and reachability properties;
- Coverability graph generation [17] for boundedness properties;
- The sweep-line method [19], including the automated calculation of a progress measure [21];
- A random walk algorithm for deadlock and reachability properties;
- An LTL model checker based on [11] that computes the product system of the reachability graph and a Büchi automaton that is generated from the formula using the `ltl2ba` tool [30];
- A CTL model checker based on [43];
- A depth first search algorithm for TSCC properties that uses a simplification of Tarjan's algorithm for finding the strongly connected components [38];
- A depth first search algorithm for computing bounds.

The actual algorithm is selected by the planning component (see above). Explorations are parameterised concerning their firelist generation, the particular property to be verified, and the way states are encoded and stored (see below). All algorithms implement the on-the-fly principle. That is, they terminate as soon as the result of the verification is determined. For positive queries (target state turns out to be reachable), the on-the-fly approach is crucial for the excellent performance of explicit state space methods [45].

Fire List Generation. This task has been organised as a separate module as it implements the essence of the stubborn set method. From a base class that implements brute force exploration (all enabled transitions form the fire list), we derive classes for stubborn set methods preserving deadlock, reachability, TSCC, boundedness, bounds, LTL, and CTL. Encapsulation in an own module helps us to keep the code for the actual search algorithms reasonably small.

Encoding. The exploration modules basically work on an integer vector representing the current marking of the search. Before actually storing such a vector, we transform it into a bit vector, aiming at less memory consumption. Our compression techniques include

- No compression at all: 32 bits are used per marking and place (useful for unbounded nets);
- Bit compression: Based on place bounds given in the model, as many bits as necessary for representing the numbers between 0 and the place bound are used (useful for bounded nets with bounds that are known by construction);
- Huffman encoding [14] (new in LoLA 2): this is suitable for nets with no knowledge of token bounds;
- Place invariant compression (combinable with the other methods): we exempt places from being stored if their value can be computed from the remaining places using a place invariant [36].

As a specific way of encoding, the calculation of canonical representatives of the symmetry method [15,34] is placed here. It is implemented as an encoder that can be parameterised with any other encoder. It canonises a marking and then encodes it using the other encoder. This way, the symmetry method is completely encapsulated for the remaining state space exploration.

Storing. The store maintains information about the already seen states. It is crucial for termination but also for memory consumption and run-time. According to our own profiling, about 90% of the run time of a state exploration is consumed for searching and inserting states. LoLA supports prefix trees and Bloom filters (new in LoLA 2). A Bloom filter just records hash values of visited markings, so we may miss states thus producing an under approximation of the state space. This is taken care of in the result presentation. Instead of *not reachable*, we would return *unknown* while the result *reachable* is preserved by Bloom filtering. We can reduce the probability of hash collision by adding more hash tables with independent hash functions. The user may choose the number of hash tables. We can further calculate the likelihood of remaining hash collisions.

Symmetry. While the *application* of symmetries is integrated into encoding, their calculation remains a separate task. It is executed prior to state space exploration but after starting alternative parallel verification technique (so they do not need to wait for termination of the sometimes lengthy symmetry calculation). Symmetries are calculated based on graph automorphisms. This graph is composed of the Petri net under investigation and the property (the formula tree). The two graphs are linked at the atomic propositions of the property.

This way, symmetries are computed such that the given property is preserved. This general approach is new in LoLA 2. In LoLA 1, the symmetry method could only be applied to global properties that were known not to break symmetry at all. Although LoLA 2 just computes a polynomial size generating set of the graph automorphism group [18], number of generators and run-time may be prohibitive. We counter this problem by the opportunity to compute symmetries in parallel (exploiting the multicore nature of today's computers) and by user definable bounds for run-time and number of generators. We then continue with a subgroup of the symmetry group of the net.

Siphon/Trap Property. If every siphon contains a marked trap, a Petri net (under mild restrictions concerning the arc multiplicities) cannot have deadlocks. If the given property asks for reachability of deadlocks, evaluation of this siphon/trap property permits early abortion of an otherwise time-consuming state space exploration. We evaluate the siphon/trap property by coding it as a Boolean formula [29] which we ship to the Minisat SAT solver [4]. This approach appears to outperform by orders of magnitude other known approaches for evaluating the siphon/trap property. In the model checking contest, this approach is responsible for about 30% of the negative answers (no target state reachable) that LoLA produces to deadlock queries. This takes quite some burden off state space exploration since the on-the-fly principle does not work in the negative case, so negative queries consume much more memory and time.

State Equation. We transform the reachability problem to a list of convex subproblems (only conjunctions of place-based atomic propositions) and ship this list to the Sara tool [44]. Sara generates a linear program for a convex problem that is based on the state equation and uses the `lp_solve` tool. It checks whether the resulting firing count vector can be arranged to an executable firing sequence. If so, it has solved the original problem. If not, it modifies the linear program based on missing tokens that have been detected in the fireability check. If no modification yields a result, it has a negative answer to the original problem. As the siphon/trap property, the state equation approach takes a lot of burden off state space verification, this time for reachability queries other than deadlock checks. Sara is even more supportive in the model checking contest than the siphon/trap check.

For plain reachability problems, the offered methods perfectly complement each other. While state space exploration performs excellent on positive queries (target state reachable), the structural methods have their merit especially (but in case of Sara not limited to) negative queries. Altogether, the portfolio approach of LoLA 2 yields excellent performance. For more complex properties, it is the rewriting and categorisation that propels the performance of LoLA 2. In the model checking contest, about 15% of CTL properties can be characterised as reachability problems and solved using the powerful portfolio described above. Other CTL problems could be categorised as LTL problems, so the more efficient LTL model checker would be available (recently, LoLA solved more than 90% of the LTL queries but only about 70% of the CTL queries). We are not using this opportunity in the model checking contest since there are subtle semantic

deviances in case of deadlocks between LTL and CTL, regarding the semantics fixed for the contest.

With already a few years experience with LoLA 2, we experienced no major problems concerning its modular structure. The most convincing observation is the fact, that all major state space reduction methods have their natural place: symmetries in the encoder, stubborn sets in firelist generation (only the *ignorance problem* [41] must be taken care of in the exploration), Bloom filtering in the store, the sweep-line method and coverability graph generation in the exploration, and the structural methods as part of our portfolio. Several student project showed that the code base of LoLA 2 is much better extensible than LoLA 1. We believe that this fact contributes to the stability of the tool.

6 Conclusion

The title *Petri net model checking* for this article was chosen with two thoughts in mind. First, LoLA 2 reads place/transition nets and is thus a model checker for Petri nets. Second, much of the performance of LoLA comes from Petri net theory and from exploiting the defining features or Petri nets, monotonicity, linearity, and locality. The monotonicity of the firing rule is used for optimising many basic routines, and enables coverability graphs and the siphon/trap property. Linearity, that is the view of Petri nets being vector addition systems, helps for state compression as well as the state equation approach. Locality is useful for the stubborn set method which is the most powerful state space reduction method in LoLA 2. Consequently, the performance of LoLA 2 can hardly be transferred to other modelling formalisms.

LoLA is focussed on verification technology. Instead of offering our own graphical user interface, we designed LoLA for easy integration into other frameworks. These design goals were confirmed in several case studies and actual integration examples. In the future, we shall continue to work on powerful verification technology. Additionally, we shall include the last remaining features of LoLA 1 that have not yet made it into LoLA 2 (such as printing the state space, or searching home states).

References

1. Best, E., Schlachter, U.: Analysis of Petri nets and transition systems. In: Proceedings ICE. EPTCS, vol. 189, pp. 53–67 (2015)
2. Das, D., Chakrabarti, P.P., Kumar, R.: Functional verification of task partitioning for multiprocessor embedded systems. ACM Trans. Des. Autom. Electron. Syst. **12**(4), 44 (2007)
3. Decker, G., Overdick, H., Weske, M.: Oryx – an open modeling platform for the BPM community. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 382–385. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_29

4. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37
5. Emerson, E.A., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Sci. Comput. Program. **2**(3), 241–266 (1982)
6. Dalsgaard, A.E., et al.: Extended dependency graphs and efficient distributed fixed-point computation. In: van der Aalst, W., Best, E. (eds.) PETRI NETS 2017. LNCS, vol. 10258, pp. 139–158. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57861-3_10
7. Dill, D.L., Knapp, M.A., Gage, P., Talcott, C., Laderoute, K., Lincoln, P.: The pathalyzer: a tool for analysis of signal transduction pathways. In: Eskin, E., Ideker, T., Raphael, B., Workman, C. (eds.) RRG/RSB-2005. LNCS, vol. 4023, pp. 11–22. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-48540-7_2
8. Kordon, F., et al.: Homepage of the Model Checking Contest, June 2017. http://mcc.lip6.fr/
9. Billington, J., et al.: The Petri net markup language: concepts, technology, and tools. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 483–505. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44919-1_31
10. Cardozo, N., et al.: Modeling and analyzing self-adaptive systems with context Petri nets. In: Proceedings of the TASE, pp. 191–198. IEEE (2013)
11. Geldenhuys, J., Valmari, A.: More efficient on-the-fly LTL verification with Tarjan's algorithm. Theoret. Comput. Sci. **345**(1), 60–82 (2005)
12. Gerth, R., Kuiper, R., Peled, D.A., Penczek, W.: A partial order approach to branching time logic model checking. In: Proceedings of the International Symposium on Theory of Computing and Systems, ISTCS 1995, Tel Aviv, Israel, 4–6 January 1995, pp. 130–139. IEEE Computer Society (1995)
13. Heiner, M., Richter, R., Schwarick, M.: Snoopy - a tool to design and animate/simulate graph-based formalisms. In: Proceedings of the PNTAP (2008)
14. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proc. IRE **40**, 1098–1101 (1952)
15. Junttila, T.A.: Computational complexity of the place/transition-net symmetry reduction method. J. UCS **7**(4), 307–326 (2001)
16. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 645–659. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_55
17. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)
18. Knuth, D.E.: Efficient representation of perm groups. Combinatorica **11**(1), 33–43 (1991)
19. Kristensen, L.M., Mailund, T.: A generalised sweep-line method for safety properties. In: Eriksson, L.-H., Lindsay, P.A. (eds.) FME 2002. LNCS, vol. 2391, pp. 549–567. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45614-7_31
20. Kristensen, L.M., Schmidt, K., Valmari, A.: Question-guided stubborn set methods for state properties. Form. Methods Syst. Des. **29**(3), 215–251 (2006)
21. Schmidt, K.: Automated generation of a progress measure for the sweep-line method. STTT **8**(3), 195–203 (2006)
22. Kummer, O., Wienberg, F.: Renew - the reference net workshop. In: Petri Net Newsletter, pp. 12–16 (2000)

23. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: verification and participant synthesis. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 46–60. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79230-7_4
24. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting WS-BPEL processes using flexible model generation. Data Knowl. Eng. **64**(1), 38–54 (2008)
25. Lohmann, N., Verbeek, E., Ouyang, C., Stahl, C.: Comparing and evaluating Petri net semantics for BPEL. IJBPIM **4**(1), 60–73 (2009)
26. Meis, B., Bergenthum, R., Desel, J.: travis - an online tool for the synthesis and analysis of Petri nets with final states. In: van der Aalst, W., Best, E. (eds.) PETRI NETS 2017. LNCS, vol. 10258, pp. 101–111. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57861-3_7
27. Mrasek, R., Mülleand, J., Böhm, K., Becker, M., Allmann, C.: Property specification, process verification, and reporting - a case study with vehicle-commissioning processes. Inf. Syst. **56**(C), 326–346 (2016)
28. Niewiadomski, A., Wolf, K.: LoLA as abstract planning engine of PlanICS. In: Proceedings of the PNSEi. CEUR, vol. 1160, pp. 349–350 (2014)
29. Oanea, O., Wimmel, H., Wolf, K.: New algorithms for deciding the Siphon-Trap property. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 267–286. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13675-7_16
30. Oddoux, D., Gastin, P.: LTL 2 BA: fast translation from LTL formulae to Büchi automata. http://www.lsv.fr/~gastin/ltl2ba/
31. Schmidt, K.: LoLA wird Pfadfinder. In: Proceedings of the AWPN, CEUR Workshop Proceedings, p. 26 (1999)
32. Schmidt, K.: Stubborn sets for standard properties. In: Donatelli, S., Kleijn, J. (eds.) ICATPN 1999. LNCS, vol. 1639, pp. 46–65. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48745-X_4
33. Schmidt, K.: How to calculate symmetries of Petri nets. Acta Inf. **36**(7), 545–590 (2000)
34. Schmidt, K.: Integrating low level symmetries into reachability analysis. In: Graf, S., Schwartzbach, M. (eds.) TACAS 2000. LNCS, vol. 1785, pp. 315–330. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46419-0_22
35. Schmidt, K.: LoLA: a low level analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 465–474. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44988-4_27
36. Schmidt, K.: Using Petri net invariants in state space construction. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 473–488. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36577-X_35
37. Stahl, C., Reisig, W., Krstic, M.: Hazard detection in a GALS wrapper: a case study. In: Proceedings of the ACSD, pp. 234–243. IEEE (2005)
38. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)
39. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. J. ACM **22**(2), 215–225 (1975)
40. Valmari, A.: Stubborn sets for reduced state space generation. In: Rozenberg, G. (ed.) ICATPN 1989. LNCS, vol. 483, pp. 491–515. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-53863-1_36

41. Valmari, A.: The state explosion problem. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 429–528. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_21
42. van der Aalst, W.M.P., et al.: ProM: the process mining toolkit. In: Proceedings of the BPMDemos. CEUR, vol. 489 (2009)
43. Vergauwen, B., Lewi, J.: A linear local model checking algorithm for CTL. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 447–461. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57208-2_31
44. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. Log. Methods Comput. Sci. **8**(3) (2012)
45. Wolf, K.: Running LoLA 2.0 in a model checking competition. In: Koutny, M., Desel, J., Kleijn, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency XI. LNCS, vol. 9930, pp. 274–285. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53401-4_13

Integrating Simulink Models into the Model Checker Cosmos

Benoît Barbot[1], Béatrice Bérard[2], Yann Duplouy[3,4(✉)], and Serge Haddad[4]

[1] LACL, Université Paris -Est Créteil, Créteil, France
[2] Sorbonne Université, LIP6, CNRS UMR 7606, Paris, France
[3] IRT SystemX, Paris-Saclay, Palaiseau, France
[4] LSV, ENS Paris-Saclay, CNRS, Inria, Université Paris-Saclay, Cachan, France
duplouy@lsv.fr
http://cosmos.lacl.fr/simulink.html

Abstract. We present an implementation for Simulink model executions in the statistical model-checker Cosmos. We take profit of this implementation for hybrid modeling and simulations combining Petri nets and Simulink models.

Keywords: Performance evaluation · Hybrid systems
Statistical model checking · Simulink

1 Introduction

The validation of safety properties is a crucial concern for the design of computer guided systems, in particular for cyber-physical systems. For instance, in transport systems, a classical approach consists in analyzing the interactions of a randomized environment (roads, cross-sections, etc.) with a vehicle controller, often derived from a Simulink® model. However, while largely used in practice by industrial system designers, Simulink does not benefit from a formal semantics. Thus engineers usually have to infer the behaviours of their models from experiments which weakens the validation process. For this reason, several works proposed formal translations from (subsets of) Simulink blocks to other models like hybrid automata [1,15], or languages like Lustre [16]. Other works directly define exact semantics [10] or operational semantics [11] for Simulink. We follow the latter approach and propose semantics for a significant fragment of Simulink. We proceed in two steps: we first develop an exact version, and then enrich it with effective procedures that were integrated into the statistical model checker Cosmos. With the resulting tool, we obtain performance indices

This research work has been carried out in the framework of IRT SystemX, Paris-Saclay, France, and therefore granted with public funds within the scope of the French Program "Investissements d'Avenir".
S. Haddad—The work of this author is supported by the project ERC EQualIS (FP7-308087).

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 363–373, 2018.
https://doi.org/10.1007/978-3-319-91268-4_19

for models combining a randomized environment described by a stochastic Petri net and a controller described in Simulink, as illustrated for a double heater system.

2 Cosmos and Simulink

2.1 Cosmos

In [2], HASL an expressive temporal logic was introduced in order to analyze stochastic and hybrid discrete event systems. Its formulas are described by a Linear Hybrid Automaton (LHA) and an expression involving state variables and using path and stochastic operators. This logic is supported by a tool named Cosmos which performs statistical model checking of (ordinary or colored) stochastic Petri nets with general distributions. The main algorithm of this tool randomly simulates the net according to its stochastic semantics and synchronizes it with the execution of the formula's automaton. During the synchronization, it evaluates the expression of the formula providing a numerical (or boolean) value per trajectory. A statistical procedure decides when to stop the simulation and produces a confidence interval for the HASL expression. Cosmos includes a family of statistical methods depending on the nature of the formula and the assumptions on the model ([12–14,17]).

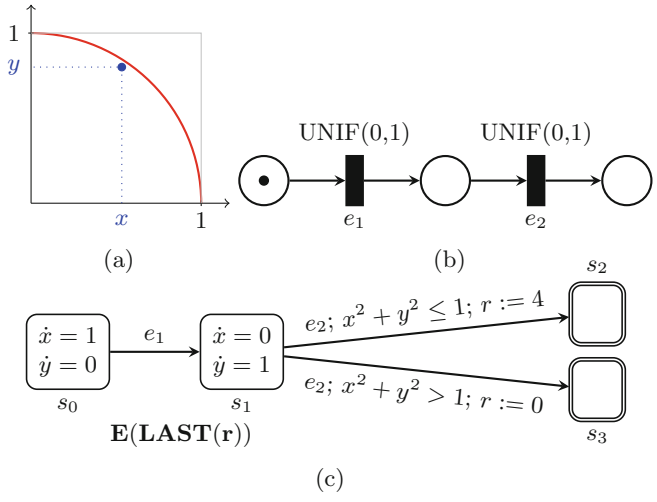


Fig. 1. Computing an approximation of π with a HASL formula over a stochastic Petri net

Figure 1 illustrates the computation of an estimation of π by the Monte-Carlo method. It consists in uniformly sampling points (x, y) in the square $[0, 1] \times [0, 1]$ and checking whether they belong to the quarter disc with measure $\pi/4$ testing

$x^2 + y^2 \leq 1$ (see Fig. 1a). A trajectory is obtained by synchronizing the LHA (Fig. 1c) with the Petri net (Fig. 1b), producing random values for x (firing delay of e_1) and y (firing delay of e_2) respectively. The last value of r is then 4 if $x^2 + y^2 \leq 1$, leading to one final state and 0 otherwise, leading to the other final state. Then the expression $E(LAST(r))$ corresponds to the expected value of r at the end of a trajectory.

The tool Cosmos consists of about 22000 lines of C++ code and is freely available at [4] under GPLv3. It relies on code generation to perform efficient simulation. It is divided into three main parts:

1. The parsing and code generation part reads the input files and the command line to build data structures for the net and the automaton. Then optimised C++ code simulating both the synchronized behaviour of the net and the automaton is generated.
2. The simulator library implements the core algorithm for synchronization and the generation of events using a pseudo random number generator. The main data structure used by the library is an Event Queue corresponding to the next transition firing.
3. The server part launches several copies of the simulator built from the library and the generated code and aggregates their results. According to statistical parameters, a procedure decides whether enough trajectories have been simulated and stops all simulators when needed. Then, HASL expressions are evaluated.

Cosmos has been successfully applied in the context of flexible manufacturing systems [2] and biological networks [3,7]. In addition, it has also been customised in order to address the challenges raised by different projects: simulation of rare events [8], cosimulation of a pacemaker software with a model of the human heart [9], sampling uniform trajectory for timed automata [5].

2.2 Simulink

Simulink®[1] is a graphical programming environment for modeling and simulating dynamic systems. The main interface is a graphical block diagramming tool, with customizable libraries. It is used, for example, in the development of embedded systems for autonomous vehicles.

We introduce Simulink models, called in the sequel *SK*-models, through the example of Fig. 2 (a formal syntax can be found in report [6]).

Informally an *SK*-model is a set of *blocks* connected by *links* transporting *signals*, where a signal is a mapping from a time interval to a real value. A block contains a set of operators generating output signals from input signals. Blocks are classified according to three criteria:

(i) Whether they are continuously evaluated or sampled, in which case the block is said *discrete* and a *sampling delay* must be provided;

[1] https://fr.mathworks.com/products/simulink.html.

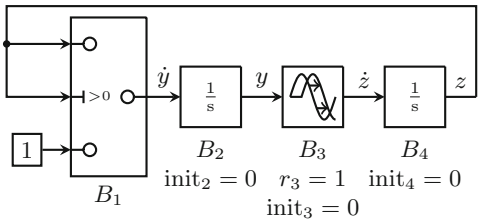


Fig. 2. An example of Simulink model

(ii) Whether there is a *latency* for evaluation of inputs, and whether this latency is *infinitesimal* or not. A latency is infinitesimal if it is either null (the block is said *immediate*) or such that the output value at time t only depends on the inputs on $[t_{\text{init}}, t]$ (like in integration). A non infinitesimal latency is said *positive*. A *non null* latency is a positive or infinitesimal non null latency.

(iii) Whether the output value depends on *threshold crossing* by an input signal, called *critical input* and denoted by i_c. In this case, the threshold values $(v_i)_{i \in I}$ must be specified, as a countable increasing sequence without accumulation point.

For instance, the *SK*-model shown in Fig. 2 features: a Switch block B_1 (continuous, immediate, with null latency and a single threshold) with operator: $\text{op}(i_1, i_c, i_2)(t) = \mathsf{if}(i_c(t) > 0, i_1(t), i_2(t))$; an Integrator block B_2 (continuous with infinitesimal latency) with operator defined by $\text{op}(i)(t) = \int_{t_{\text{init}}}^{t} i(\tau)\mathrm{d}\tau$; and a *Transport Delay* block B_3 (continuous) which outputs its input signal with a latency $r = 1$.

3 Extensions to Cosmos

3.1 Simulink Semantics

Exact Semantics. Simulink® models represent hybrid systems, combining discrete and continuous components. The *trajectory* of an *SK*-model $\mathcal{M}$, if it exists, is the vector $\boldsymbol{w}$ of all values of output signals over the simulation interval $[t_{\text{init}}, t_{\text{end}}]$. Signal evaluation requires to split this interval into a finite sequence of contiguous sub-intervals, on which the trajectory is the solution of a system of differential equations. Discrete samplings and threshold crossings are located at the boundaries of these sub-intervals.

Due to blocks with positive latency, the specification of the differential equations over a sub-interval $[t_i, t_{i+1}]$ depends on the trajectory over previous intervals. Moreover, it also depends on the *mode* of the threshold blocks *i.e.* the position of each critical input signal with respect to its threshold values.

For instance, in the *SK*-model of Fig. 2 over $[0, 2]$, the initial values are $t_0 = 0$, $y(0) = 0$, and $z(0) = 0$. Choosing $t_1 = 1$, the differential equations over $[0, 1]$ are $\dot{y} = 1$ and $\dot{z} = 0$ (since block B_3 has a delay $r_3 = 1$, it uses its default value

$\text{init}_3 = 0$). Those yield $z(t) = 0$ and $y(t) = t$ over $[0, 1[$, which are consistent with the mode of block B_1 ($z \leq 0$). Over the next interval $]1, 2[$ the differential equations are $\dot{z}(t) = t - 1$ (due to the value of y over $[0, 1]$) and $\dot{y}(t) = z(t)$ assuming the mode of block B_1 is $z > 0$. The associated solution is $z(t) = (t-1)^2/2$ and $y(t) = (t-1)^3/6 + 1$ which are consistent with the hypothetical mode.

This example illustrates the issues of defining an appropriate mode for specifying the differential equation systems related to the current interval. This requires to introduce a minimal length of time interval $\varepsilon_{\mathbb{T}}$ such that the solutions of the differential equations remain consistent on such a small interval. Consequently, a Simulink model does not necessarily admit a trajectory: either if such a mode does not exists or if some differential equation (like $\dot{y} = 2^y$) cannot be solved on the simulation interval. With suitable hypotheses on the operators, we prove that if a trajectory exists, it is unique. Unfortunately, as is often the case for hybrid systems, the existence of a trajectory is an undecidable problem.

Approximate Semantics. Since the resolution of differential equations and the determination of threshold crossings are not effective operations, an approximate version of the semantics above is needed for efficient implementation in the tool Cosmos.

The search for a trajectory relies on an iterative construction of a partition into sub-intervals $[t_{\text{init}}, t_{\text{end}}] = \bigcup_{i=0}^{N-1} [t_i, t_{i+1}]$. We emphasize three main features:

(1) The signal values are only stored at times $(t_i)_{0 \leq i \leq N}$: For each output signal o of a block B, an array $W_{B,o}[i]$ of its values is kept for each time t_i. This implies interpolation operations to compute signal values at intermediate times.
(2) When $t_0, \ldots, t_i$ are built, the determination of the next evaluation time t_{i+1} must take into account variable integration steps, as done in the Runge-Kutta-Fehlberg (aka ODE45) method used here. Here, we omit the adaptation details of these classical procedures, performed with the constant mode of t_i, which produce the new evaluating time t_{i+1} and the associated values.
(3) In addition to $\varepsilon_{\mathbb{T}}$, we introduce the accuracy parameter ε_V to handle the termination tests in the adaptative integration method (like Runge-Kutta-Fehlberg).

Besides the implementation objective, the other main purpose of this approximate semantics is to find suitable hypotheses ensuring that if a trajectory exist w.r.t. exact semantics, then there exists a close approximate one: for any $\varepsilon > 0$ there exist $\varepsilon_{\mathbb{T}}$ and ε_V such that for each output o of block B and for all i, $|W_{B,o}[i] - w_{B,o}(t_i)| < \varepsilon$.

Implementation. Integrating this semantics in Cosmos is done through a code generator reading Simulink models (in MATLAB `.slx` format) and producing C++ code with a similar programming interface to the one used for nets. It has one transition, and dynamic arrays (explained above) for each signal. Its single event is scheduled at the next evaluation time. Cosmos currently supports a subset of representative Simulink blocks, showcasing thresholds and integration.

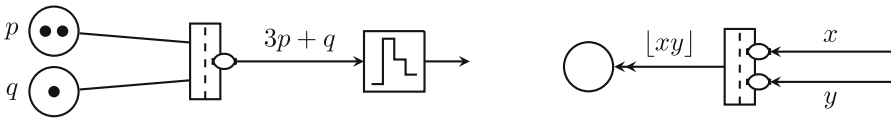
3.2 Simulink/Petri Net Communication

In order to simultaneously manage a Petri net and a Simulink model, we need to specify how they interact, implying transformation of discrete values (like the number of tokens in places) into continuous ones (like signal values), and vice-versa. We have chosen to perform these operations via special transitions called *interface transitions*.

Interface Transitions. There are two kinds of interface transitions: *SK*-in transitions, directed from the net to the *SK*-model and *SK*-out transitions in the other direction.

The input arcs of a *SK*-in transition (see Fig. 3a) are *test arcs* (explained with the firing) connected to places of the net. Any output arc is connected to the input of a Simulink block. An *SK*-in is enabled when the content of at least one of its input place is modified. The firing of such a transition proceeds as follows: a function is associated with each output arc, taking as parameters the contents of the input places. When the firing takes place, the function is evaluated. This function can be specified by a multiset of tokens as illustrated, or by a C code associated with the arcs.

The input arcs of an *SK*-out transition (see Fig. 3b) are output signals of an *SK*-model and the output arcs are *overwriting* arcs connected to places that can only be connected to ordinary transitions of the net by *read* arcs. Similarly to *SK*-in transitions, the output arcs are labelled by functions of the incoming signals. Such a transition is activated at every sampling time of the *SK*-model. Upon firing, it rewrites the contents of the output places according to the evaluation of the function.



(a) An *SK*-in transition (b) An *SK*-out transition

Fig. 3. Petri net/Simulink Interface transitions

Simulation Loop. We now describe in more details the simulation loop, which enhances the standard Cosmos simulation loop. All enabled transitions are stored into an *event* queue implemented as a binary heap, with their time of occurrence, their priority and weight. The next Simulink step is added as a possible event. At each simulation step, the earliest event is chosen. Among simultaneous events, the (decreasing) priority order is the following: *1. SK*-out firings, *2.* ordinary transition firings, *3. SK*-in firings, *4. SK*-event. In case of equal priorities, the choice is randomized according to the weights. Once an event is selected:

– If it is an ordinary transition firing, the marking is updated, the associated C code is executed; transitions that are newly enabled trigger new events while events corresponding to disabled transitions are removed.

– If it is an *SK*-in firing, the Simulink signals are updated and the time of the Simulink event is set to the current time.
– if it is the *SK*-event, all output signals are updated and the time of the Simulink event is updated as presented in Sect. 3.1. Finally, the *SK*-out transitions are added to the event queue with the current time.
– if it is a *SK*-out firing, the contents of output places are updated.

To simulate a discrete event system, at each step, one only has to compute what the next event will be and increase the simulation time to the time of this event. This is how ordinary Petri net transitions are fired in Cosmos. This leads to aoDon efficient simulation of such system as the time to compute a simulation depends on the number of events and not on the simulated time. Unfortunately, this property is lost when simulating hybrid systems: the *SK*-event is triggered at least at a fixed frequency (δ_{max}). In the next section we experimentally study the impact of integration on simulation time.

Implementation. Given a stochastic Petri net and a Simulink model, Cosmos generates code simulating each model as well as a code synchronizing them. This synchronization is done by dispatching the events according to their model, with a *ad hoc* handling of sychronisation transitions (*SK*-in and *SK*-out) which may update both signals and markings.

4 Benchmarks

Among the multiple systems that can be modeled within this framework, we choose a well known toy (but still relevant) example: a device with two heaters prone to faults and using bang-bang controllers to keep the temperature in a room between 20 °C and 25 °C. The system is modeled by a stochastic Petri net (Fig. 4) with randomized faults and repairs. The evolution of room temperature and heater behaviours are hybrid and thus are modeled in Simulink (Fig. 5). The fault transitions of the net have an exponential time distribution (with different rates). The repairman, initially at the Idle state, randomly chooses which (faulty) heater he will repair, then proceeds in fixed time and goes back to the Idle state. By default, both heaters are working (places Op_1 and Op_2 have a token).

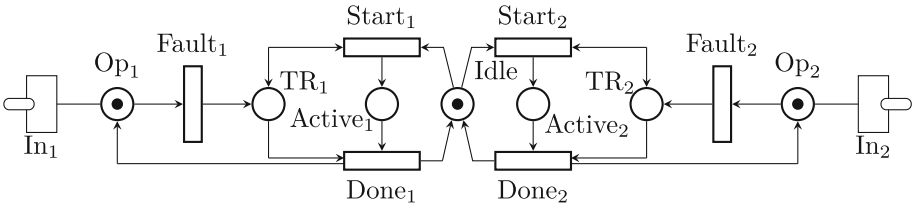


Fig. 4. Petri net handling faults and repairs of a double heater

The Simulink model handles the differential equations for both heaters, and for the outside temperature which is modelled by a sine wave (T_{ext}). The differential equation is: $\dot{T} = \mathbb{1}_{On_1} c_1(T_{h_1} - T) + \mathbb{1}_{On_2} c_2(T_{h_2} - T) + c_{ext}(T_{ext} - T)$ where c_1, c_2, and c_{ext} are the respective thermal conductivity coefficients, T_{h_1} and T_{h_2} are the respective temperatures at which each heater functions, and On_1 and On_2 are the respective states of each bang-bang controller which should maintain the temperature between $T_{min} = 20\ °C$ and $T_{max} = 25\ °C$. A bang-bang controller is a very simple hysteresis controller where the heater is switched on ($On_i = 1$) when the temperature decreases to T_{min} and switched off ($On_i = 0$) when the temperature increases to T_{max}. The inputs $F1$ and $F2$ receive respectively the content of places Op_1 and Op_2.

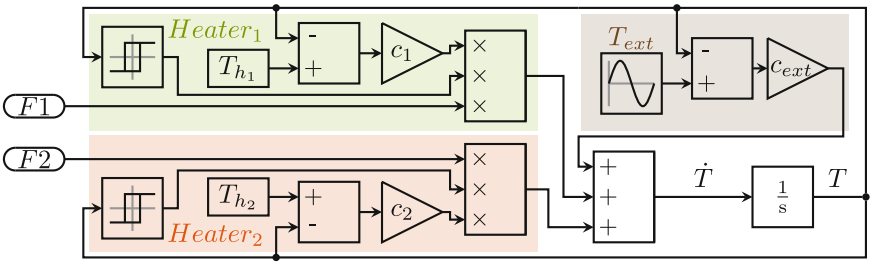


Fig. 5. A Simulink model computing differential equations for the double heater. It contains four parts: the two heater temperatures, the external temperature and the block completing the differential equation

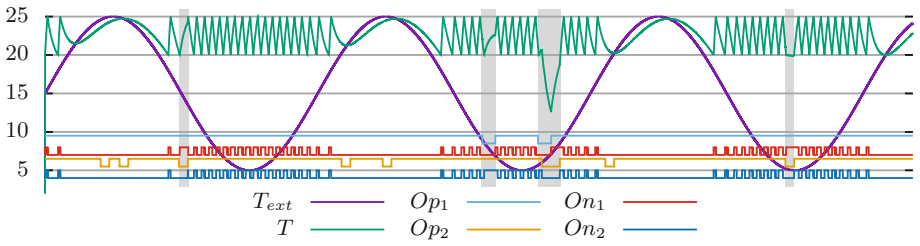


Fig. 6. A trace of simulation: T and T_{ext} represent the inside and outside temperatures, Op_i corresponds to heater i being operational and On_i corresponds to heater i being switched on. Gray areas highlight failure of at least one heater when $T_{ext} < T_{min}$.

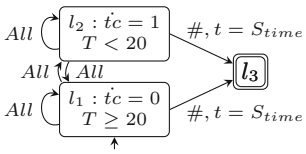
Figure 6 shows a simulation of the system. In the first period there is only a small failure of heater 2, and we can observe the bang-bang behaviours of the system. In the second period both heaters fail at the same time while the outside temperature is low, thus the temperature quickly drops to 13 °C before the first heater is repaired.

We are interested in several performance indices. The first type concerns the reliability of the model measured by two indices: *the minimal temperature observed along a trajectory* (I_1) and *the time spent in a state where the temperature is below* 20 °C (I_2). The second type concerns the average behaviour: *the average temperature* (I_3), *the average number of switched-on heaters* (I_4), which is correlated with the energy consumption of the system, and *the average time during which the repairman is idle* (I_5).

These indices are specified in HASL with an LHA (Fig. 7) which accepts the trajectories after S_{time} time units. The LHA contains a hybrid variable tc with derivative 1 when the temperature is below 20 °C and 0 otherwise. The HASL expressions start with a *probability operator*: here AVG is used for all indices to specify the average value over all trajectories; then a *path operator* (Min, Last, Mean) which is defined along each path. Path operators take as parameters algebraic expressions over the Petri net places and the LHA variables. For example, I_1 specifies the minimal temperature along a trajectory and then the average value over all trajectories.

I_1 : AVG(Min(T))
I_2 : AVG(Last(tc))
I_3 : AVG(Mean(T))
I_4 : AVG(Mean($Active_1 + Active_2$))
I_5 : AVG(Mean($Idle$))

(a) HASL formulas



(b) LHA

Fig. 7. HASL specification for performance indices

The model as it is described above is referred to as M_0. In order to study the overhead of integral computations over the stochastic simulation, we build two additional alternative models. The first one M_1 is a model where the integration block in the Simulink diagram has been replaced by a discrete time integrator. In the model M_2, the simulink part is omitted, keeping only events that are transition firings.

Table 1. Simulation results

Indices	M_0	M_1
I_1	[18.698 ; 18.716]	[18.272 ; 18.289]
I_2	[66.344 ; 67.217]	[92.921 ; 93.927]
I_3	[22.439 ; 22.442]	[22.485 ; 22.488]
I_4	[0.4999 ; 0.5006]	[0.4884 ; 0.4890]
I_5	[0.9239 ; 0.9242]	[0.9239 ; 0.9241]

Models	Build time	Sim. time
M_0	5.74s	6 885s
M_1	5.73s	1 145s
M_2	1.31s	1.810s

Each model was run for 500 000 simulations of 2 000 s, with $\varepsilon_V = 0.01$ and $\delta_{max} = 1$. The sine wave frequency was 0.01 and oscillating between 5 °C and

25°C, and the time step of the discrete-time integrator was δ_{max} (1 s). We used $T_{h_1} = 55$ °C, $T_{h_2} = 65$ °C, $c_1 = 0.02$, $c_2 = 0.013$ and $c_{ext} = 0.04$. Results are reported in Table 1. The left table reports the computed confidence interval for the different indices, the right one reports simulation and building times.

Tool Analysis. The build time is always less than the simulation time and becomes negligible when models include a Simulink part. The critical factors for simulation time are: (i) the speed of step firing, about 10^{-7} s. for net firing compared to 10^{-6} s. for Simulink steps, and (ii) the number of steps per trajectory, about 40 for M_2 vs. 2000 for M_1. As expected, the use of a discrete-time Integrator yields a faster simulation, albeit still far longer than the net alone, while it affects the accuracy of the index values and more precisely triggers a larger variation of temperature over time.

Property Analysis. We focus on the most pertinent model M_0, with two antagonist goals: minimizing the installation cost (depending on the parameters of heaters and repairman), and maximizing the comfort of the user (depending on the temperature evolution). With the current parameters, each heater is active about 1/4 of the time and the repairman is idle 92% of the time. The average temperature is about 22 °C, reaching the objective, while the minimal temperature is slightly above 18 °C.

5 Conclusion and Future Work

We have presented a standalone tool that synchronously simulates a stochastic Petri net and a Simulink model. The simulation trace is defined by a formal semantics. Using HASL formulas, one can define complex performance indices, on which the tool evaluates confidence intervals.

We plan to apply this approach to more challenging models, such as the evaluation of autonomous vehicle controllers into various vehicular environments or for analyzing strategies for energy consumption in data centers.

We plan to increase the number of Simulink blocks supported by Cosmos, and enhance the expressivity of high-level Petri nets with floating-points colors.

References

1. Agrawal, A., Simon, G., Karsai, G.: Semantic translation of simulink/stateflow models to hybrid automata using graph transformations. Electr. Notes Theor. Comput. Sci. **109**, 43–56 (2004)
2. Ballarini, P., Barbot, B., Duflot, M., Haddad, S., Pekergin, N.: HASL: a new approach for performance evaluation and model checking from concepts to experimentation. Perform. Eval. **90**, 53–77 (2015)
3. Ballarini, P., Duflot, M.: Applications of an expressive statistical model checking approach to the analysis of genetic circuits. Theor. Comput. Sci. **599**, 4–33 (2015)
4. Barbot, B., Ballarini, P., Djafri, H.: http://cosmos.lacl.fr

5. Barbot, B., Basset, N., Beunardeau, M., Kwiatkowska, M.: Uniform sampling for timed automata with application to language inclusion measurement. In: Agha, G., Van Houdt, B. (eds.) QEST 2016. LNCS, vol. 9826, pp. 175–190. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43425-4_13
6. Barbot, B., Bérard, B., Duplouy, Y., Haddad, S.: Integrating simulink models into the model checker cosmos. Research report, March 2018. https://hal.archives-ouvertes.fr/hal-01725835
7. Barbot, B., Haddad, S., Heiner, M., Picaronny, C.: Rare event handling in signalling cascades. Int. J. Adv. Syst. Meas. **8**(1–2), 69–79 (2015)
8. Barbot, B., Haddad, S., Picaronny, C.: Coupling and importance sampling for statistical model checking. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 331–346. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_23
9. Barbot, B., Kwiatkowska, M., Mereacre, A., Paoletti, N.: Building power consumption models from executable timed I/O automata specifications. In: Proceedings of HSCC 2016, pp. 195–204. ACM (2016)
10. Benveniste, A., Bourke, T., Caillaud, B., Pouzet, M.: Non-standard semantics of hybrid systems modelers. J. Comput. Syst. Sci. **78**(3), 877–910 (2012)
11. Bouissou, O., Chapoutot, A.: An operational semantics for simulink's simulation engine. In: Proceedings of the 13th ACM SIGPLAN/SIGBED, LCTES 2012, pp. 129–138. ACM, New York (2012)
12. Chow, Y.S., Robbins, H.: On the asymptotic theory of fixed-width sequential confidence intervals for the mean. Ann. Math. Stat. **36**, 457–462 (1965)
13. Clopper, C., Pearson, E.S.: The use of confidence or fiducial limits illustrated in the case of the binomial. Biometrika **26**, 404–413 (1934)
14. Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301), 13–30 (1963)
15. Tiwari, A.: Formal semantics and analysis methods for simulink stateflow models. Technical report, SRI (2002)
16. Tripakis, S., Sofronis, C., Caspi, P., Curic, A.: Translating discrete-time Simulink to Lustre. ACM Trans. Embed. Comput. Syst. **4**(4), 779–818 (2005)
17. Wald, A.: Sequential tests of statistical hypotheses. Ann. Math. Stat. **16**(2), 117–186 (1945)

LocalProcessModelDiscovery: Bringing Petri Nets to the Pattern Mining World

Niek Tax[1,2(✉)], Natalia Sidorova[1], Wil M. P. van der Aalst[3], and Reinder Haakma[2]

[1] Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{n.tax,n.sidorova}@tue.nl

[2] Philips Research, Prof. Holstlaan 4, 5665 AA Eindhoven, The Netherlands
{niek.tax,reinder.haakma}@philips.com

[3] RWTH Aachen, Aachen, Germany
wvdaalst@pads.rwth-aachen.de

Abstract. This paper introduces the tool *LocalProcessModelDiscovery*, which is available as a package in the process mining toolkit ProM. *LocalProcessModelDiscovery* aims to discover *local process models*, i.e., frequent patterns extracted from event logs, where each frequent pattern is expressed in the form of a Petri net. Local process models can be positioned in-between process discovery and Petri net synthesis on the one hand, and sequential pattern mining on the other hand. Like pattern mining techniques, the *LocalProcessModelDiscovery* tool focuses on the extraction of a set of frequent patterns, in contrast to Petri net synthesis and process discovery techniques that aim to describe all behavior seen in an event log in the form of *a single model*. Like Petri net synthesis and process discovery techniques, the models discovered with *LocalProcessModelDiscovery* can express a diverse set of behavioral constructs. This contrasts sequential pattern mining techniques, which are limited to patterns that describe sequential orderings in the data and are unable to express loops, choices, and concurrency.

Keywords: Petri nets · Frequent pattern mining · Process discovery

1 Introduction

LocalProcessModelDiscovery is a novel tool for the discovery of frequent patterns in the form of Petri nets from event logs. This paper aims to present this Petri-net-based tool and provide some insights into the discovery techniques used. The tool is implemented as a package in the Java-based *process mining* framework ProM [12] and is publicly available at https://svn.win.tue.nl/repos/prom/Packages/LocalProcessModelDiscovery/ and in the ProM package manager. After installing the *LocalProcessModelDiscovery* package to ProM, the tool can be started by importing an event log in XES [26] format into ProM and then

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 374–384, 2018.
https://doi.org/10.1007/978-3-319-91268-4_20

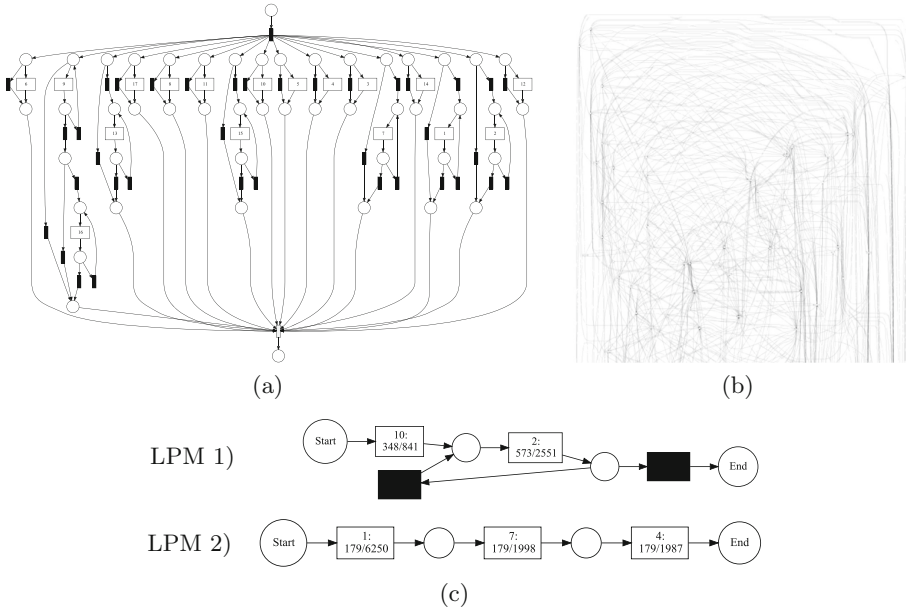


Fig. 1. The Petri net model mined from the MSNBC dataset with *(a)* the Inductive Miner [21], *(b)* the ILP Miner [25], and *(c)* two LPMs mined from the MSNBC dataset. Black transitions correspond to silent transitions.

running the ProM plugin *Search for Local Process Models* using this event log as input. The algorithms that we developed for the mining of frequent Petri net patterns from event logs [10,22–24] form the core of the *LocalProcessModelDiscovery* tool.

Mining of Local Process Models (LPMs) can be positioned in-between the research areas of *Petri net synthesis* and *process discovery* on the one hand and *frequent pattern mining* on the other hand. *Frequent pattern mining* [17] techniques focus on extracting local patterns from data. *Sequential pattern mining* [14] techniques are a type of frequent pattern mining that focuses on the extraction of frequent patterns from sequence data. While process discovery [1] and Petri net synthesis techniques aim to discover an *end-to-end process model*, sequential pattern mining techniques aim to extract a *set of patterns* where each pattern describes a subsequence that frequently occurs in the event log. Sequential pattern mining techniques can be used to generate insights from event data that only contain weak relations between the activities, i.e., that have a relatively high degree of randomness. From such event logs, process discovery and Petri net synthesis techniques generate either overgeneralizing models that allow for too much behavior, or generate a 'spaghetti'-model that is accurate in the allowed behavior but is not understandable and often overfitting. Figure 1a gives an example of an overgeneralizing process model, showing the process model discovered with the Inductive Miner [21] from web click data from the

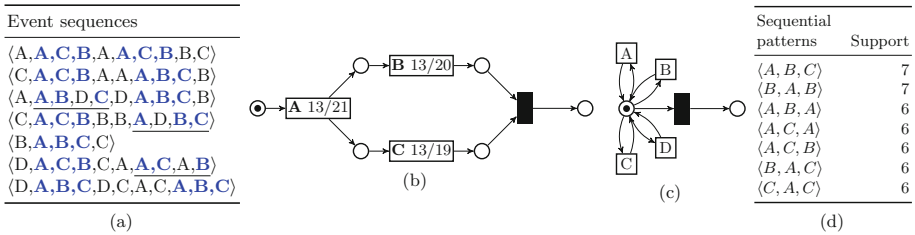


Fig. 2. *(a)* A log L with highlighted instances of the frequent pattern. *(b)* An example local process model that shows some frequent behavior in L. *(c)* The Petri net discovered from L with the Inductive Miner tool [21]. *(d)* The sequential patterns discovered from L with the PrefixSpan algorithm [18] (with minimum support = 6).

MSNBC.com news website[1] (note that most activities can be skipped and/or repeated allowing for any behavior). Figure 1b gives an example of a non-interpretable 'spaghetti'-like process model, discovered from the same dataset with the ILP Miner [25]. The first LPM of Fig. 1c, however, shows that activity 10 is generally followed by multiple instances of activity 2.

2 What Is Local Process Model Mining?

The process models that can be discovered with process discovery and Petri net synthesis techniques can describe a rich set of process constructs, such as concurrency, inclusive and exclusive choices, loops, and sequential execution. In contrast, the patterns that are discovered with sequential pattern mining techniques are limited to sequential orderings. The mining of *Local Process Models* (LPMs) [24] extends sequential pattern mining techniques to Petri nets, *allowing for the discovery of local patterns of non-sequential behavior, including choices, concurrency, and loops*. Figure 2 illustrates an example local process model on an example event log, and highlights the instances of the LPM in the event log in **blue**. Note that instances of an LPM in the event log do not have to consist of consecutive events, i.e., there can be *gaps* within a pattern instance. The LPM instances that contain gaps are indicated in the event log with underline.

The *LocalProcessModelDiscovery* tool provides an implementation of the local process model mining algorithm presented in [24]. Intuitively, the local process model mining algorithm works by iteratively expanding patterns into larger candidate patterns, for which it calculates the support by calculating an alignment [2] between the pattern and the event log. Patterns that satisfy a minimum *support threshold* that is provided by the user are then expanded further in the next expansion iteration of the algorithm. The local process model mining algorithm returns a ranked list of patterns, where the patterns are ranked according to a weighted average over the following quality criteria:

[1] http://kdd.ics.uci.edu/databases/msnbc/msnbc.data.html.

Support. Relates to the number of times that the behavior that is described by the Petri net pattern is found in the event log.

Confidence. A pattern has high confidence when a high ratio of the events in the event log of the activities that are described in the pattern belong to instances of the pattern.

Language Fit. Relates to how much behavior that is allowed by the Petri net pattern is actually observed in the event log at least once. A Petri net that allows for many behavior that was never observed has low language fit.

Determinism. Relates to the average number of enabled transitions during replay of the pattern instances on the pattern. A purely sequential model has optimal determinism, while a model that allows for more behavior might have higher support but will have lower determinism.

Coverage. Relates to how many events in the event log are described by the pattern.

Related Tools

Several tools for Petri net synthesis (see [3] for an overview of synthesis techniques) have been implemented throughout the years, including APT [5], GENET [6], and Petrify [9]. APT [5] allows for the synthesis and analysis of Petri nets and transition systems. GENET [6] allows for the synthesis of a Petri net from automata. Petrify [9] allows for the synthesis of Petri nets and asynchronous circuits. VipTool [4] allows for the synthesis of a Petri net from a finite partial language. Furthermore, ProM [12], APROMORE [20], PMLAB [7], and bupaR [19] are tools that provide implementations of process discovery algorithms. SPMF [13] is a pattern mining tool that provides implementations of a wide variety of frequent pattern mining algorithms, including sequential pattern mining algorithms. Additionally, several frequent pattern mining algorithms have been implemented in the data mining toolkit WEKA [16]. Another related tool is WoMine [8] which provides an implementation of the *w-find* algorithm [15] for mining frequent behavioral patterns from a process model, allowing a user to find frequent subprocesses in a process model. The Episode Miner [21] provides functionality to mine frequent partial orders from an event log. Diamantini et al. [11] provide a related tool to mine frequent behavioral patterns from event logs. Unlike *LocalProcessModelDiscovery*, the behavioral patterns mined with this tool and with the Episode Miner are not able to discover choice relations and loops.

3 The LocalProcessModelDiscovery Tool

Figure 3 shows the main screen of the *LocalProcessModelDiscovery* tool, consisting of three panels: on the left side a panel to configure the mining parameters, in the middle a panel that presents the mining results when mining has completed, and on the right side a panel to interact with and navigate through the mined local process models.

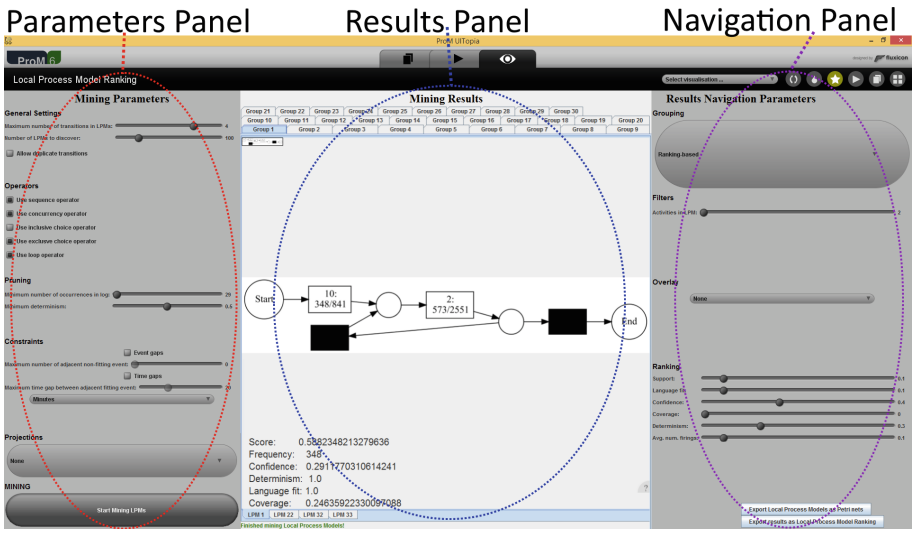


Fig. 3. The home screen of the *LocalProcessModelDiscovery* tool.

3.1 Configuring the Local Process Model Miner

Figure 4 shows the mining parameters panel. Located at the top of the panel is a **maximum number of transitions in the LPMs** slider, which allows the user to set the maximum number of non-silent transitions for the local process models. The maximum number of non-silent transitions puts an upper bound on the number of expansion iterations in the local process model mining algorithm, therefore, setting this slider to higher values enables the tool to discover larger patterns at the cost of higher computation time.

The **number of LPMs to discover** slider lets the user specify a maximum number of patterns that he or she wants to obtain, allowing him or her to prevent mining an overload of patterns. Note that the local process model discovery algorithm returns a ranked list of patterns, therefore, the algorithm returns the patterns with the highest weighted score to the user.

The **allow duplicate transitions** toggle allows the user to specify whether or not he or she wants to mine patterns that contain multiple transitions with the same label. Enabling this option comes at the price of higher computation cost.

The **operator** section of the parameter panel allows the user to include or exclude patterns with certain control-flow constructs in the search space of the mining procedure.

In the **pruning** section of the parameter panel the user can specify a **minimum number of occurrences** of the pattern in the log (i.e., minimum support), and a minimum **determinism** value for the pattern. Patterns that do not meet these thresholds set by the user are not presented in the results and are not expanded into larger patterns, thereby pruning the search space of the local process model mining algorithm.

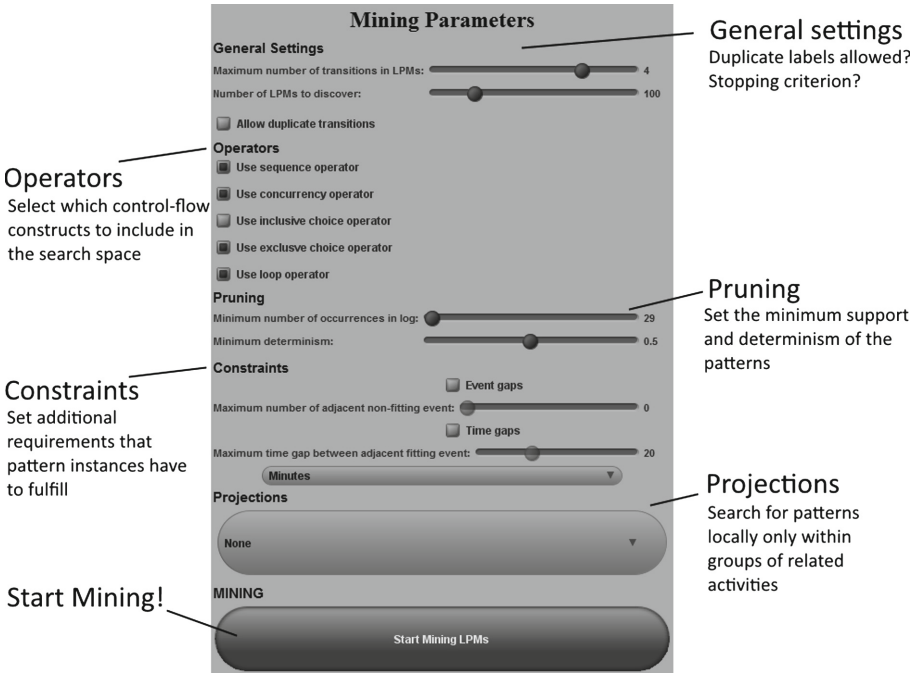
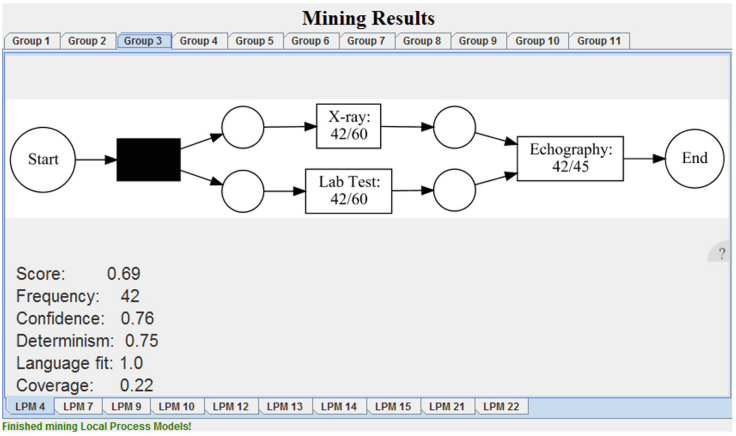


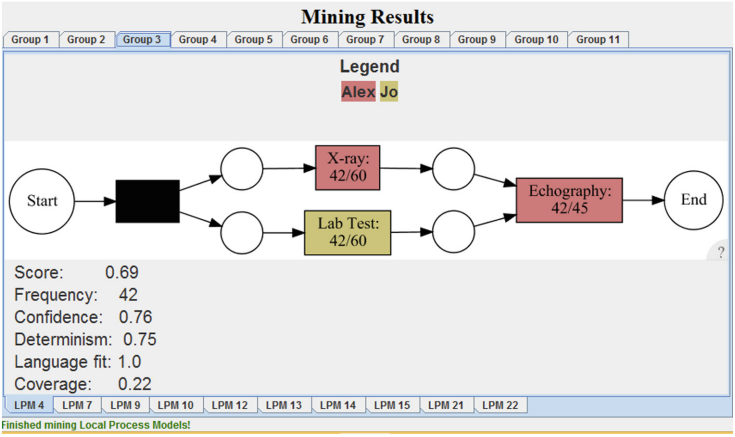
Fig. 4. The parameters panel of the *LocalProcessModelDiscovery* tool.

Instances of a local process model in the event log do not have to be consecutive, i.e., there can be other events that do not fit the behavior specified by the LPM in-between. When this is not desired, constraint-based mining of LPMs can be used to put restrictions on the instances of LPMs, thereby not including instances in the event log that do not comply with there constraints, even when it fits the behavior specified by the LPM. The parameters section allows the user to specify **event gap constraints** and **time gap constraints**. An event gap constraint puts a maximum on the number of non-fitting events in-between two fitting events of an instance of an LPM in the log. For example, a sequence $\langle a, b, x, x, c \rangle$ is considered to be an instance of a Petri net N with language $\mathfrak{L}(N) = \{\langle a, b, c \rangle\}$ when the event gap constraint is set to 2, but is not considered to be an instance when the event gap constraint is set to 1. For event logs containing timed events, time gap constraints can be used to specify an upper bound on the time difference between two consecutive events that fit the behavior of an LPM.

The computational complexity of mining LPMs is exponential in the number of activities in the event log [24]. Several heuristic mining techniques have been proposed in [22] to make the mining of LPMs on event logs with large numbers of events feasible. These heuristic techniques work by detecting clusters of activities that frequently occur close to each other in the event log, and then mining the LPMs for each cluster individually, restricting the expansion steps of the



(a)



(b)

Fig. 5. *(a)* An example LPM in the results panel of the *LocalProcessModelDiscovery* tool, and *(b)* an example of the *overlay* feature in the *navigation* panel, projecting the *resource* information on top of the pattern.

LPM mining to activities that are part of the same cluster of activities. The **projections** section of the mining parameters panel allows the user to configure the tool to use these approaches.

When starting the *LocalProcessModelDiscovery* tool, it automatically sets the minimum support parameter and the projections configuration to a default value that is dependent on the event log, using information such as the number of events and the number of activities in the event log.

3.2 Interpreting Local Process Model Results

Figure 5a shows grouping of local process models in the mining results panel. The figure shows a single local process model mined from a hospital event log that

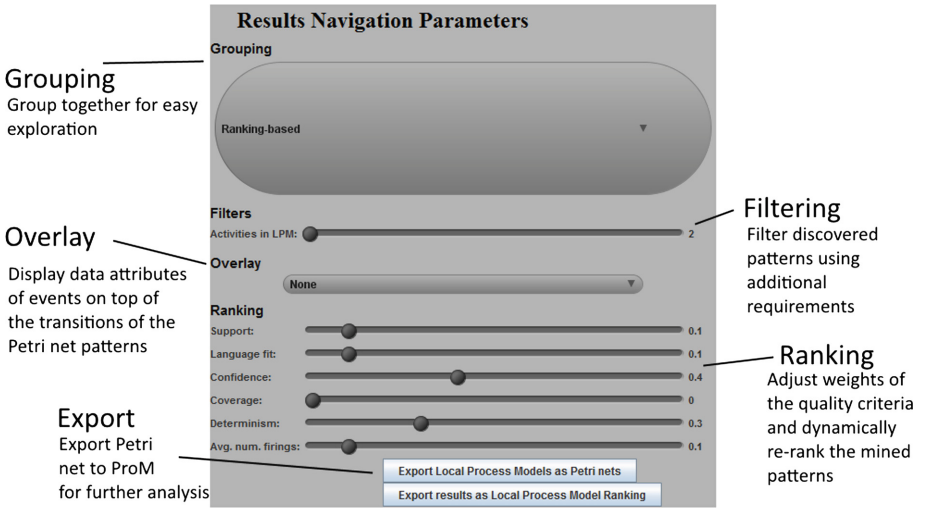


Fig. 6. The navigation panel of the *LocalProcessModelDiscovery* tool, which allows the user to interact with the mining results and perform more in-depth follow-up analysis.

describes the behavior that a *lab test* and an *X-ray* are performed in arbitrary order, finally followed by an *echography*. Printed below the Petri net are the scores of the pattern in terms of the local process model quality criteria. The local process models are ranked by the aggregated score of the LPM (shown as **score**), and the tabs in the bottom of the panel can be used to navigate through the LPMs in the ranking. Typically, the mining procedure results in multiple local process models that specify behavior over the same alphabet of activities. By default, the resulting local process models are grouped by the alphabet of activities that they describe. The **group** tabs above the Petri net can be used to explore the LPMs that describe different alphabets of activities.

3.3 Navigating Local Process Model Results

The navigation panel provides several functionalities to interactively navigate through the resulting local process models obtained through mining. A weighted average over the quality criteria of local process models is used to rank the resulting local process models, and the results are presented in the order of the ranking. The user can reconfigure the weights assigned to each of the quality criteria in the **ranking** section of the navigation panel, resulting in an updated ranking of local process models in the **results panel**.

The **overlay** functionality in the navigation panel allows the user to project data attributes of the events in the log onto the local process models. The overlay functionality consists of a drop-down selector where the user can select one of the global event attributes (i.e., an event attribute that is set for every event in the log). Figure 5b illustrates the overlay functionality and shows the mining results panel when selecting the *org:resource* event attribute for the local process

model of Fig. 5. The results show that the *X-ray* and *Echography* events that fit the behavior of this local process model are most frequently performed by employee *Alex*, while the *Lab Test* events that fit the behavior of this pattern are most frequently performed by employee *Jo*. Note that this does not say anything about the *X-ray* events that do not fit the behavior of this pattern, i.e., the X-ray events that are not performed concurrently to the *Lab Test* and before *Echography*.

The **filters** section of the results panel allows the user to filter out local process models from the results that do not comply with certain specifications that are provided by the user, such as a minimum number of **activities in the log**. In the **grouping** section, the user can select a strategy for grouping mined local process models into groups of local process models for the visualization in the results panel. By default, the **ranking-based** grouping strategy is used, which adds one local process model A to the same group as another local process model B if (1) the set of activities of A is a subset of the set of or equal to the activities of B and (2) A has a lower aggregated score than B.

4 Conclusion

This paper presents the tool *LocalProcessModelDiscovery*, which allows for the mining of frequent patterns (called local process models) from event logs that are expressed as Petri nets. Local process models are positioned in-between process discovery and Petri net synthesis on the one hand, and frequent pattern mining on the other hand. The local process models that can be mined with this tool extend existing sequential pattern mining approaches: while sequential patterns are restricted to mining frequent sequential behavior, local process models allow the frequent patterns to describe a more general language over the activities by expressing the patterns as Petri nets. *LocalProcessModelDiscovery* supports the mining of local process models as well as functionality to navigate through the mining results and to relate discovered local process models back to the event log for a more in-depth analysis.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisc. Rev.: Data Min. Knowl. Discovery **2**(2), 182–192 (2012)
3. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. TTCSAES. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47967-4
4. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of Petri nets from finite partial languages. Fundamenta Informaticae **88**(4), 437–468 (2008)
5. Best, E., Schlachter, U.: Analysis of Petri nets and transition systems. In: Proceedings of the 8th Interaction and Concurrency Experience (2015)

6. Carmona, J., Cortadella, J., Kishinevsky, M.: GENET: a tool for the synthesis and mining of Petri nets. In: Application of Concurrency to System Design, pp. 181–185. IEEE (2009)
7. Carmona, J., Solé, M.: PMLAB: an scripting environment for process mining. In: Proceedings of the BPM Demo Sessions, pp. 16–20 (2014). CEUR-ws.org
8. Chapela-Campa, D., Mucientes, M., Lama, M.: Towards the extraction of frequent patterns in complex process models. Jornadas de Ciencia e Ingeniería de Servicios, pp. 215–224 (2017)
9. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. IEICE Trans. Inf. Syst. **80**(3), 315–325 (1997)
10. Dalmas, B., Tax, N., Norre, S.: Heuristics for high-utility local process model mining. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, pp. 106–121 (2017). CEUR-ws.org
11. Diamantini, C., Genga, L., Potena, D., Storti, E.: Discovering behavioural patterns in knowledge-intensive collaborative processes. In: Appice, A., Ceci, M., Loglisci, C., Manco, G., Masciari, E., Ras, Z.W. (eds.) NFMCP 2014. LNCS (LNAI), vol. 8983, pp. 149–163. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17876-9_10
12. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: a new Era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005). https://doi.org/10.1007/11494744_25
13. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S.: SPMF: a Java open-source pattern mining library. J. Mach. Learn. Res. **15**(1), 3389–3393 (2014)
14. Fournier-Viger, P., Lin, J.C.W., Kiran, R.U., Koh, Y.S., Thomas, R.: A survey of sequential pattern mining. Data Sci. Pattern Recogn. **1**(1), 54–77 (2017)
15. Greco, G., Guzzo, A., Manco, G., Pontieri, L., Saccà, D.: Mining constrained graphs: the case of workflow systems. In: Boulicaut, J.-F., De Raedt, L., Mannila, H. (eds.) Constraint-Based Mining and Inductive Databases. LNCS (LNAI), vol. 3848, pp. 155–171. Springer, Heidelberg (2006). https://doi.org/10.1007/11615576_8
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. ACM SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)
17. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Min. Knowl. Discovery **15**(1), 55–86 (2007)
18. Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceedings of the International Conference on Data Engineering, pp. 215–224. IEEE (2001)
19. Janssenswillen, G., Depaire, B.: BupaR: business process analysis in R. In: Proceedings of the BPM Demo Sessions, pp. 160–164 (2017). CEUR-ws.org
20. La Rosa, M., Reijers, H.A., Van Der Aalst, W.M.P., Dijkman, R.M., Mendling, J., Dumas, M., García-Bañuelos, L.: APROMORE: an advanced process model repository. Expert Syst. Appl. **38**(6), 7029–7040 (2011)
21. Leemans, M., van der Aalst, W.M.P.: Discovery of frequent episodes in event logs. In: Ceravolo, P., Russo, B., Accorsi, R. (eds.) SIMPDA 2014. LNBIP, vol. 237, pp. 1–31. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-27243-6_1

22. Tax, N., Sidorova, N., van der Aalst, W.M.P., Haakma, R.: Heuristic approaches for generating local process models through log projections. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, pp. 1–8. IEEE (2016)
23. Tax, N., Genga, L., Zannone, N.: On the use of hierarchical subtrace mining for efficient local process model mining. In: International Symposium on Data-Driven Process Discovery and Analysis (2017). CEUR-ws.org
24. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Mining local process models. J. Innov. Digit. Ecosyst. **3**(2), 183–196 (2016)
25. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS, vol. 5062, pp. 368–387. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68746-7_24
26. XES Working Group: IEEE standard for eXtensible Event Stream (XES) for achieving interoperability in event logs and event streams. IEEE Std 1849-2016, pp. 1–50, November 2016

A Model Checker Collection for the Model Checking Contest Using Docker and Machine Learning

Didier Buchs, Stefan Klikovits, Alban Linard(✉), Romain Mencattini, and Dimitri Racordon

Software Modeling and Verification (SMV) Group, Faculty of Science, University of Geneva, Geneva, Switzerland
{didier.buchs,stefan.klikovits,alban.linard,romain.mencattini, dimitri.racordon}@unige.ch

Abstract. This paper introduces `mcc4mcc`, the Model Checker Collection for the Model Checking Contest, a tool that wraps multiple model checking solutions, and applies the most appropriate one based on the characteristics of the model it is given. It leverages machine learning algorithms to carry out this selection, based on the results gathered from the 2017 edition of the Model Checking Contest, an annual event in which multiple tools compete to verify different properties on a large variety of models. Our approach brings two important contributions. First, our tool offers the opportunity to further investigate on the relation between model characteristics and verification techniques. Second, it lays out the groundwork for a unified way to distribute model checking software using virtual containers.

1 Introduction

The Model Checking Contest @ Petri Nets [16] (MCC) is an annual event, held during the Petri Nets Conference. Initially launched in 2011, its objective is to investigate on the relation between model characteristics and verification techniques. Over the years, the MCC has proven successful to compare the performance of model checking tools with respect to different problems, and increase the confidence in the results they produce. For perspective, the 2017 edition saw 10 tools competing in 9 categories on 77 models. Despite the results being interesting and valuable to the model checking community, they do not clearly address the primary objective of the MCC. Indeed, a clear relation between model characteristics and verification techniques has still to be found.

Although a compelling theoretical answer to this problem may still be out of reach, seven years of MCC results opened the door to an empirical study. As each

D. Racordon—This project is supported by: FNRS STRATOS: Strategy based Term Rewriting for Analysis and Testing Of Software, the Hasler Foundation, 1604 CPS-Move and COST IC1404: MPM4CPS.

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 385–395, 2018.
https://doi.org/10.1007/978-3-319-91268-4_21

tool exports the list of employed verification techniques (e.g. decision diagrams, explicit or bounded model checking, etc.), it is possible to use the results of previous iterations of the MCC to shed light on this relation. We therefore created the *Model Checker Collection for the Model Checking Contest* (`mcc4mcc`), a tool that aims at establishing a relation between model characteristics and verification techniques. The tool serves as a wrapper around the model checking tools that competed in past iterations of the MCC, and leverages machine learning algorithms to select the most appropriate one, given a model and examination type. `mcc4mcc` uses virtual containers to provide homogeneous packaging and execution.

Our contributions to the research community are two-fold:

- While the current way of participation to the MCC (submission of virtual machines) is a valid means to enter, the re-use of tools can be burdensome and requires a lot of resources. We show that virtual container systems such as Docker[1] can be leveraged to create lightweight, uniform distribution systems that invite re-use of the tools.
- Based on the availability of such lightweight tools, we propose the Model Checker Collection as a means to empirically evaluate the relation between model characteristics and verification techniques. We show that `mcc4mcc` is capable of finding the best-suited tool amongst a range of candidates and provide details about machine learning algorithms we use for decision making.

This paper describes the methodology we have used to create Docker images containing the MCC model checkers, as well as the machine learning process that is implemented in `mcc4mcc`. Our tool is available at https://github.com/cui-unige/mcc4mcc, and is released under the open source MIT license.

2 Creation of Docker Containers

Tool authors who wish to compete in the MCC have to provide a virtual machine (VM) for their tool. Submissions have to be completely self-contained, i.e. contain the tool, its dependencies, models and their precomputed equivalents (for some tools). In the context of the MCC, this approach presents two significant advantages. First, examiners do not need to struggle with installation procedures, and authors can ensure their tool is run under the best configuration possible. Second, tools are guaranteed to be executed in identical, reproducible environments, which is obviously desirable for a competition.

However, when the tool is to be used as a classic software, such VM compartmentalization might present some inconveniences. Exchanging data with a virtual machine can only be done by the means of virtual networks, usually through a secure shell[2]. This limits, or at least significantly complicates the options a final user has to run the tool in a pipelined process. Disk space can

[1] https://www.docker.com/.
[2] https://en.wikipedia.org/wiki/Secure_Shell.

be listed as another major inconvenience. Table 1 shows the size of each tool's virtual machine, submitted to the MCC. A virtual machine for a MCC tool typically uses at least 2 Gigabytes with some requiring up to 8.5 Gigabytes of disk space. In a setting such as our research goal, where multiple tools need to be used, this may quickly reach problematic proportions.

Table 1. MCC tools and their licenses, virtual machine sizes and docker image sizes. Docker images include the models translated into the tool-specific format taken from the virtual machines.

			Virtual Machine		Docker	
Tool	License	granted	Size	System	Size	System
GreatSPN [7]	Closed	✔	2.9 Gb	Debian	278 Mb	Debian
ITS-Tools [27]	GNU GPL v3	✔	3.3 Gb	Debian	843 Mb	Debian
LoLA [25]	GNU APL v3	✔	8.5 Gb	Debian	14 Mb	Alpine
LTSMin [15]	BSD 3 Clause	✔	3.4 Gb	Debian	668 Mb	Debian
Marcie [11]	Non commercial	✔	2.3 Gb	Debian	21 Mb	Alpine
Smart [8]	Closed	✖	2.7 Gb	Debian	284 Mb	Debian
Spot [10]	GNU GPL v3	✔	7.0 Gb	Debian	2 Gb	Debian
Tapaal [14]	GNU GPL v2	✔	2.3 Gb	Debian	69 Mb	Alpine
Tina [3]	Freeware	✔	2.6 Gb	Debian	830 Mb	Debian
Total			35 Gb		4.9 Gb	

This caveat lead us to question the means of packaging model checkers within VMs. In general, we see three possibilities to produce an easily distributable, reusable `mcc4mcc` wrapper that incorporates the MCC submissions:

Embedding existing virtual machines. One possibility is to create one monolithic VM that wraps around the VMs that were submitted by the developers. This requires the ability to run nested virtualization (i.e. virtual machines within virtual machines). While it is technically possible, this technique usually results in poor performances [21]. Moreover, the result would suffer from the disk space issue mentioned above, as the enclosing virtual machine would weigh around 40 Gigabytes. This approach clearly violates the constraint of facilitated distribution.

Merging of virtual machines. This technique addresses some of the performance issues. In addition, its also leads to a far lighter virtual disk, as it does not involve the duplication of many shared files and binaries. Unfortunately, such a virtual machine is very susceptible to possible configuration conflicts between the dependencies of the different tools. This approach heavily increases maintenance, modification and update complexity.

Packaging tools in virtual containers. The third approach relies on virtual *containers* [2], rather than virtual *machines*. Contrary to VMs, virtual containers do not provide their own operating system and do not require hardware support. They share central resources such as a file system or a network

interface with their host machine, if required. Due to their isolated virtual-memory region they remain independent. Virtual containers cause less system overhead and have the potential for increased performance. They are usually lightweight and include only the bare necessities for program execution. This means that most of them abstain from including a user interface and unnecessary libraries.

We chose the third approach to create the `mcc4mcc`. We use Docker[3] to compose and execute the virtual containers. The tools inside the containers are built either from the tool sources when available, or by copying the required files from the virtual machines or binary distributions of the tools.

Docker containers [20] are instantiated virtual containers that package programs and files. These instances are created based on "container snapshots" called docker *images*. Images store a container's file system state and, upon running, create exact runtime copies of the snapshot. For our purposes we create Docker images that contain the tools and any other files which are required for the tool execution. Additionally, certain Linux files and programs are provided by the *base image*. Base images are docker images that contain a minimal execution environment and are extended by our images.

The disk space required by the tool docker images is stated in the last two columns of Table 1 (alongside the base image). The severe reduction results in images sizes of less than 400 megabytes, which facilitates distribution. On average, we achieved a reduction of over 90%. For LoLA and and Marcie we achieved drastic size reductions of 99.9% and 99.1%, respectively. This is due to the fact that we could use the *Alpine* base image, which measures only around 4 megabytes.

In general, Docker container images can be stored within private or public repositories for easy distribution. The usual naming scheme `<organization>/<tool>:<tag>`, `organization` is e.g. the university, laboratory or team name, `tool` is the tool name, and `tag` is an optional tag, such as version number or variant, facilitates the identification of each specific image. We follow this naming scheme in this article, and propose a standardized way to provide Docker images for tools, designed to ease both the diffusion of the tools, and their wrapping for the MCC. We split each tool into two Docker images:

- `<organization>/<tool>`, an image that contains only the tool, with the manual page and examples if needed, for instance `unirostock/LoLA`. This image should ideally be provided and maintained by the tool developers, and may be tagged for instance with the tool version.
- `mccpetrinets/<tool>`, a wrapper dedicated to the Model Checking Contest, that is built on top of `<organization>/<tool>` and creates an entry point (the default executable script in the image) conforming to the Model Checking Contest specifications. This image may also embed additional data, such as precomputed data for known models, and may tagged, for instance with the year of the participation.

[3] https://www.docker.com.

The two images are built using docker's configuration files, called *Dockerfiles*[4]. These typically describe the build process (e.g. how to build a tool from source), various configuration settings and more importantly an entry point that points to the command (or set thereof) that is ran when the container starts. The Dockerfiles we used for the build are available in the respective subfolders of our tool repository[5].

Once the Docker images are configured, a tool can be run within its confined environment. By default, tools run in a completely isolated environment, and do not have access to the host file system. Hence it is necessary to explicitly mount a path to the host directory holding the models' data. We communicate other information by the use of environment variables, as it is the MCC's convention.

```
$ docker run --volume=<path to model>:/mcc-data \
             --env BK_TOOL="${BK_TOOL}" \
             --env BK_EXAMINATION="${BK_EXAMINATION}" \
             mccpetrinets/<tool>:<year>
```

Running a container will try to find the image within the local repository, and – if not available – search in online repositories (e.g. docker hub[6]) for the image and download it if found. Online distribution is useful for tool developers to share their tool more easily than using binary archives. Uploading an image does not distribute the tool sources, as it contains the result of executing the Dockerfile, and thus the binaries.

This is especially important for tool developers who wish to keep the tool's source code private, as is the case for some MCC competitors. While most tools are open-source, such as LoLA or Spot, closed-source but freeware, such as Tina, a few programs explicitly require licenses to be used (e.g. GreatSPN, Smart). Table 1 lists the individual licenses for the tools that competed in the 2017 MCC. The table also states a license has been granted, if necessary.

3 Using Machine Learning to Choose the Right Tool

Our second research question examines whether we can use empirical methods to choose a well-suited tool for a model checking problem. This evaluation is based on the results of the 2017 MCC. In the MCC, tools participate in different *examinations* such as state space exploration or deadlock reachability. Table 2 cross-references the MCC's competing tools and the respective examinations they participated in.

In each examination the tools try to solve their objective on three categories of models: *known*, *stripped* and *surprise*. Known models are taken from a catalog and known before the competition. Tool developers can hence optimize their tools towards solving them. Stripped models, similar to known models are taken from a catalog, however the precise model will not be identified beforehand.

[4] https://docs.docker.com/engine/reference/builder/.
[5] https://github.com/cui-unige/mcc4mcc.
[6] https://hub.docker.com/.

Table 2. Examinations performed by the tools

Tool	State Space	Upper Bounds	Reachability			CTL		LTL	
			Cardinality	Deadlock	Fireability	Cardinality	Fireability	Cardinality	Fireability
GreatSPN	✔	✔	✔	✔	✔	✔	✔	✖	✖
ITS-Tools	✔	✔	✔	✔	✔	✔	✔	✔	✔
LoLA	✖	✔	✔	✔	✔	✔	✔	✔	✔
LTSMin	✔	✔	✔	✔	✔	✔	✔	✔	✔
Marcie	✔	✔	✔	✔	✔	✔	✔	✖	✖
Smart	✔	✖	✖	✖	✖	✖	✖	✖	✖
Spot	✖	✖	✖	✖	✖	✖	✖	✖	✔
Tapaal	✔	✔	✔	✔	✔	✔	✔	✖	✖
Tina	✔	✖	✖	✖	✖	✖	✖	✖	✖

The challenge lies in performing the correct optimizations and try to provide a performance as if the model were known. Surprise models are new models that have not been used before. It is the hardest category, as tool developers cannot create optimizations for their tools.

Each model might also be parameterized, meaning that there are different configurations for this model. An example is the model *philosophers*, which can specify the number of dining philosophers as parameter. This information is published in a supporting document alongside the respective model, which contains all other model characteristics. These characteristics can be e.g. whether the model is *reversible*, contains *sink places* or *quasi liveness*. Table 3 lists the model characteristics for a few models. Note that the characteristics are Boolean, but unknown for certain models (marked with "?").

Table 3. Characteristics of some models

Model	Ordinary	Strongly Connected	Reversible	Colored	Place/Transition	Sink Place	Simple Free Choice	Sub-Conservative	Deadlock	State Machine	Marked Graph	Source Transition	Extended Free Choice	Source Place	Quasi Live	Parameterized	Live	Safe	Sink Transition	Connected	Loop Free	Conservative	Nested Units
Eratosthenes	✔	✖	✖	✖	✔	✔	✖	✔	?	✖	✖	✖	✖	✔	✔	✔	✖	✔	✖	✖	✖	✖	✖
CSRepetitions	✔	✖	?	✔	✔	✖	✖	✔	?	✖	✖	✖	✖	✖	?	✔	?	✖	✔	✔	✖	✖	✖
TokenRing	✔	✔	✖	✔	✔	✖	✖	✔	?	✖	✖	✖	✖	✖	✔	✔	✔	?	✖	✔	✖	✔	✖
IBM703	✔	✖	✖	✖	✔	✔	✔	✖	?	✖	✖	✖	✔	✔	✔	✖	✖	✔	✖	✔	✔	✖	✖
PhilosophersDyn	✖	✔	?	✔	✔	✖	✖	✖	?	✖	✖	✖	✖	✖	?	✔	?	?	✖	✔	✖	✖	✖

The MCC results are provided in the form *Tool*, *Instance*, *Examination* → *Time*, *Memory*, where *Instance* is the combination of a *Model* and its *Parameter*. Using these results and the information about model characteristics, we can evaluate which tools are best-suited for a model with certain characteristics. Self-evidently the term "best-suited" varies depending on the application. We might e.g. be interested in choosing the tool that computes the highest number of instances for a model, requires the least amount of time or resources, or produces the fewest of errors.

To perform this selection empirically `mcc4mcc` employs machine learning techniques to choose the best-suited tool. The principle of machine learning is "to predict the future based on the past" [19]. This means that existing data is analyzed and used to draw conclusions about new data.

There are numerous different analysis techniques, of which we use a subset that is commonly referred to as *classification algorithms*. These kinds of problems typically involve an existing data set providing observations and their classification, i.e. their correlated value. This data set is referred to as *learning* or *training set*. Through analysis of this data set, classification predictions can be made for new data, i.e. the *test set*.

For `mcc4mcc` five of the algorithms provided in the `scikit-learn` [22] Python library were evaluated:

k **nearest neighbors** [26] This algorithm places the data points in a multidimensional coordinate system. New data is classified by choosing a predefined number (i.e. k) of neighbours based on the Euclidean distance (or different distance measure) and choosing the best-suited classification using this information – usually the most common class amongst the neighbors.

Support vector machines [9] represent data as points in space, divided by a clear gap that is as wide as possible. This algorithm builds a hyper-plane based on the inner product of the new data's vectors and all observation's vectors.

Neural network classification [17] uses a learning algorithm that establishes a layered graph. The data is mapped onto the input layer which represents the data's features. Each node (*neuron*) in the graph takes input from the previous layer, performs a small calculation and offers this information to the following layer adding a weight measure. The final (output) layer is the classification. Using the learning set it is possible to adjust the weight of individual neurons to guide the classification.

Naive Bayes classifier [24] This method uses the Bayes theorem [6] to calculate the new data's membership probability for each class. The class with the highest probability is chosen.

Decision Trees [5] The creation of a decision tree is a supervised learning method where characteristics are used to create a tree structure. The nodes in this tree are decision points on individual characteristics, the classes are the leaf nodes of the tree. Decision trees can be represented simply using highly nested if-then-else nodes. The goal of the algorithm is to find data characteristics that serve as discriminators for the classification.

In order to evaluate the algorithms' efficiency, it is interesting to compute the score that `mcc4mcc` would have obtained during the Model Checking Contest. Table 4 provides these results, where the score is computed by awarding 12 points to a tool whenever it can find an answer for an instance and examination, and 2 points if it uses the least amount of time or memory. Note that the computation of the scores differs slightly from the rules of the Model Checking Contest for simplicity of implementation. For instance, the score is computed for each tool, instance and examination, whereas the Model Checking Contest computes a score for each formula contained in the examination. Multipliers depending on the status of models (known, stripped or unknown) are also not applied in the custom scoring function of this paper. Thus the reader can observe that the scores provided in this article do not correspond to the scores available on the webpage[7] of the contest.

Table 4. Comparison of scores for tools and machine learning algorithms

	Total	State Space	Upper Bounds	Reachability			CTL		LTL	
				Cardinality	Deadlock	Fireability	Cardinality	Fireability	Cardinality	Fireability
`mcc4mcc` – Decision Tree	11.173	1.171	1.266	1.228	1.241	1.235	1.240	1.262	1.233	1.293
`mcc4mcc` – SVM	10.919	566	1.287	1.260	1.253	1.269	1.281	1.324	1.316	1.360
`mcc4mcc` – Neural Network	10.552	426	1.224	1.268	1.220	1.253	1.270	1.241	1.319	1.237
LoLA	10.540	0	1.287	1.335	1.253	1.345	1.318	1.324	1.316	1.360
`mcc4mcc` – KNN	9.889	584	1.259	1.232	962	1.243	1.245	1.245	1.062	1.053
LTSMin	8.680	718	1.051	1.039	594	1.044	1.043	1.043	1.088	1.057
ITS-Tools	6.746	989	680	747	851	745	522	594	830	785
`mcc4mcc` – Naive Bayes	6.406	975	632	743	824	775	749	750	599	355
Marcie	4.392	907	671	539	642	577	514	539	0	0
Tapaal	4.391	435	325	768	761	612	837	650	0	0
GreatSPN	4.211	1.052	742	438	645	441	464	424	0	0
Tina	1.096	1.096	0	0	0	0	0	0	0	0
Smart	714	714	0	0	0	0	0	0	0	0
Spot	632	0	0	0	0	0	0	0	632	0

4 Conclusion and Future Works

This article presents the Model Checker Collection for the Model Checking Contest (`mcc4mcc`). Our tool acts as a wrapper around the tools that participate in the Model Checking Contest (MCC), hosted annually at the Petri Nets conference. Using the tools of the 2017 edition and knowledge about their performance, we employ empirical evaluation to choose the most-suited tool for a problem.

[7] https://mcc.lip6.fr/2017/results.php.

Our research contributions are two-fold: First, we present how the packaging of tools in virtual containers is beneficial in terms of disk space and orchestration. `mcc4mcc` relies on Docker images, which are light-weight and closer to the execution hardware, to provide the execution environment. We provide proof-of-evidence of this efficiency by reducing the required disk size of the distributed files by over 85% on average. This allows easier distribution and execution. Docker has been further shown to be more resource efficient as it does not need to emulate an entire operating system, but merely the necessary applications.

Our second contribution, the choice of tool based on an empirical evaluation of existing model checking results uses this framework. This paper introduces the five machine learning algorithms employed for the evaluation and shows that three algorithms have the potential to achieve a higher overall score than the 2017 MCC winners. While `mcc4mcc` still has potential for improvement, it already shows promising results, choosing well-performing tools amongst ten competitors. Our tool has been published online[8] so that others might use information for own analyses. To prove the efficacy of our tool, `mcc4mcc` participates in the 2018 Model Checking Contest.

Although our research shows promising results, we do plan on adding further improvements. We subsequently list the most important ones:

Firstly we plan to augment the prediction capabilities of our machine learning engine, by using additional machine learning algorithms, and adding domain-specific customizations. Additionally we plan on adding the results of all previous MCC competitions to increase the learning data set. To ease the storage and analysis of results we are planning a homogeneous data format that can be used with a standardized API. We propose a collaboration with the Petri nets repository [12], which already contains all the MCC models of the alongside their characteristics.

Second, `mcc4mcc` allows us to discover which model characteristics are most important when selecting the best tool. We can further use the data to propose new characteristics to the MCC organizing committee.

Third, due to specific requirements we were not capable of creating virtual container images for all 2017 MCC tools. We propose a collaboration with tool developers in order to help them create virtual container images themselves. Next to the facilitated distribution, tool developers can then rest assured knowing that the tools are correctly installed and executed in their intended environment. We believe that this will ease the requirements to support different operating systems and configurations. We also plan to create containers for tools that have been developed within our team, and have competed in previous editions of the MCC: AlPiNA [13], StrataGEM [4], and Yadd [23]. The CosyVerif [1] and Ardoises [18] projects, that aim at creating an environment for formal modeling and verification should also use the containers of the various tools.

Lastly, we propose to change the MCC's submission format to virtual container images. The use of such containers requires less effort, disk space and system knowledge, while maintaining the same advantages of virtual machines.

[8] https://github.com/cui-unige/mcc4mcc.

As this change could have a great impact on the infrastructure of the contest, it should be carefully discussed and tested with the organizers.

References

1. André, É., Lembachar, Y., Petrucci, L., Hulin-Hubard, F., Linard, A., Hillah, L., Kordon, F.: Cosyverif: an open source extensible verification environment. In: 2013 18th International Conference on Engineering of Complex Computer Systems, Singapore, 17–19 July 2013, pp. 33–36. IEEE Computer Society (2013)
2. Bernstein, D.: Containers and cloud: from LXC to Docker to Kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2014)
3. Berthomieu, B., Vernadat, F.: Time Petri nets analysis with TINA. In: Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), Riverside, California, USA, 11–14 September 2006, pp. 123–124. IEEE Computer Society (2006)
4. López Bóbeda, E., Colange, M., Buchs, D.: StrataGEM: a generic Petri net verification framework. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 364–373. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07734-5_20
5. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth, Belmont (1984)
6. Broemeling, L.D.: Bayesian Analysis of Linear Models. Statistics: A Series of Textbooks and Monographs. Taylor & Francis, London (1984). https://books.google.ch/books?id=b8wPjIm9wcYC
7. Chiola, G., Franceschinis, G., Gaeta, R., Ribaudo, M.: GreatSPN 1.7: graphical editor and analyzer for timed and stochastic Petri nets. Perform. Eval. **24**(1–2), 47–68 (1995)
8. Ciardo, G., Miner, A.S.: SMART: the stochastic model checking analyzer for reliability and timing. In: 1st International Conference on Quantitative Evaluation of Systems (QEST 2004), Enschede, The Netherlands, 27–30 September 2004, pp. 338–339. IEEE Computer Society (2004)
9. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, Cambridge (2010)
10. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, É., Xu, L.: Spot 2.0 - a framework for LTL and ω-automata manipulation. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 122–129. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_8
11. Heiner, M., Rohr, C., Schwarick, M.: MARCIE – model checking and reachability analysis done efficiently. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 389–399. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_21
12. Hillah, L.M., Kordon, F.: Petri Nets Repository: a tool to benchmark and debug Petri Net tools. In: van der Aalst, W., Best, E. (eds.) PETRI NETS 2017. LNCS, vol. 10258, pp. 125–135. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57861-3_9
13. Hostettler, S., Marechal, A., Linard, A., Risoldi, M., Buchs, D.: High-level Petri net model checking with AlPiNA. Fundam. Inf. **113**(3–4), 229–264 (2011)

14. Jensen, J.F., Nielsen, T., Oestergaard, L.K., Srba, J.: TAPAAL and reachability analysis of P/T nets. In: Koutny, M., Desel, J., Kleijn, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency XI. LNCS, vol. 9930, pp. 307–318. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53401-4_16
15. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: high-performance language-independent model checking. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 692–707. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_61
16. Kordon, F., et al.: Report on the model checking contest at Petri nets 2011. In: Jensen, K., van der Aalst, W.M., Ajmone Marsan, M., Franceschinis, G., Kleijn, J., Kristensen, L.M. (eds.) Transactions on Petri Nets and Other Models of Concurrency VI. LNCS, vol. 7400, pp. 169–196. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35179-2_8
17. Kubat, M.: Neural Networks: A Comprehensive Foundation by Simon Haykin. Macmillan, Basingstoke (1994). ISBN 0-02-352781-7. Knowl. Eng. Rev. 13(4), 409–412 (1999)
18. Linard, A., Buchs, D.: Ardoises: collaborative & interactive editing using layered data. In: 17th International Conference on Application of Concurrency to System Design, ACSD 2017, Zaragoza, Spain, June 25–30, 2017, pp. 136–145. IEEE Computer Society (2017)
19. Mitchell, T.M.: Machine learning. McGraw Hill Series in Computer Science. McGraw-Hill, New York (1997)
20. Negus, C.: Docker Containers, 2nd edn. Addison-Wesley Professional, Boston (2015)
21. Pan, Z., He, Q., Jiang, W., Chen, Y., Dong, Y.: Nestcloud: towards practical nested virtualization. In: 2011 International Conference on Cloud and Service Computing, CSC 2011, Hong Kong, 12–14 December 2011, pp. 321–329. IEEE Computer Society (2011)
22. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
23. Racordon, D., Buchs, D.: Verifying multi-core schedulability with data decision diagrams. In: Crnkovic, I., Troubitsyna, E. (eds.) SERENE 2016. LNCS, vol. 9823, pp. 45–61. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45892-2_4
24. Russell, S.J., Norvig, P.: Artificial Intelligence - A Modern Approach. Prentice Hall Series in Artificial Intelligence, 2nd edn. Prentice Hall, Upper Saddle River (2003)
25. Schmidt, K.: LoLA a low level analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 465–474. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44988-4_27
26. Shakhnarovich, G., Darrell, T., Indyk, P.: Nearest-neighbor methods in learning and vision. IEEE Trans. Neural Netw. **19**(2), 377 (2008)
27. Thierry-Mieg, Y.: Symbolic model-checking using ITS-tools. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 231–237. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_20

Arduino Library Developed for Petri Net Inserted into RFID Database and Variants

Carlos Eduardo Alves da Silva,
José Jean-Paul Zanlucchi de Souza Tavares(✉),
and Marco Vinícius Muniz Ferreira

Universidade Federal de Uberlândia, Uberlândia, Brazil
dasilva.carloseduardo@hotmail.com,
jean.tavares@ufu.br, marcomuniz@outlook.com

Abstract. This work has the purpose of present the implementation of an innovative approach called PNRD (elementary Petri Net inside a RFID distributed Database) and its variation, the inverted PNRD. In this approach, the theoretical model widely studied and developed of Petri Nets is the base to define a data structure to be recorded in RFID tags. The tags are used as an object distributed database. The result is a highly adaptive, distributed, scalable and applicable control system. Therefore, this work describes the design, implementation, and validation of a library for the Arduino® platform which simplifies the development of new applications that uses the PNRD approach as well as the inverted PNRD.

Keywords: Petri net · RFID · PNRD · Inverted PNRD

1 Introduction

Radio frequency identification (RFID) is a wireless communication technology that lets computer read the identity of inexpensive electronic tags from a distance [1]. This technology is based on the detection and modulation of electromagnetics signals created by RFID readers and emitted by antennas. The RFID tags can store information and respond to the reader signal to pass their data. Typically, the tag is associated with an item which is tracked thought a production or distribution system. Although there are many applications for this technology, such as access control, department store security, equipment tracking, baggage and logistics [2], its usage is usually linked with a back-end database which contains the information of several tagged items. This database is used to control the system. This solution depends on a network infrastructure connecting all RFID readers, computers, and subsystems to track items and control the application. As presented in [3] mobile application have connectivity issues. This issue can block an entire system if there is only a centralized approach work on it. In order to reach a more reliable system, a contingent and complementary solution would be desirable.

Petri nets are a graphical and mathematical modeling tool applicable to many systems [4]. They can be used to analyze systems characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. Such versatility

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 396–405, 2018.
https://doi.org/10.1007/978-3-319-91268-4_22

makes the Petri nets applicable on the description and visualization of complex workflows systems as YAWL [5].

A different approach regarding the RFID systems and Petri net was proposed by [6]. Instead of relying on a unique database, they used the tags themselves as a disperse database. Therefore, they defined a formal data structure based on the Petri nets model that contains information about the current state of the item as well as the process it belongs. Such data structure is stored in the tags and it is updated through the process execution. This new approach is called Elementary Petri Nets inside RFID Database (PNRD). Another approach called iPNRD [7] or inversed PNRD changes Petri net data structure between RFID readers and tags compared to the original approach.

To improve the use of the PNRD for different applications, a library was developed for the Arduino® platform. This platform was chosen as a starting point to prototype future low-cost embedded systems. In addition to that, the Arduino® platform also have a large developer community and the variety of compatible electronics systems that includes some RFID readers.

The next section shows the PNRD approach, so that the library can be presented more clearly in Sect. 3. There are two applications in Sect. 4: the first one is a PNRD approach while the other one is an iPNRD approach. At last, Sect. 5 presents this paper conclusions, followed by the acknowledgment and references.

2 PNRD Approach

PNRD is a formal data structure based on the elementary Petri Net formalism or Low-Level Petri Net (LLPN) [6] created to integrate the RFID system with the Petri nets model. It assumes that the process in which the tags are inserted can be described by a Petri net model, so that the workflow of the tagged item can be defined by an incidence matrix (A^T) and its current state by a marking (M_k). These two information, along with the tag ID and a timestamp, characterize the typical tag data structure of a PNRD. The timestamp is included so it is possible to analyze time related performance indicators in the process. The antennas and RFID readers, on the other hand, are responsible for the definition of the adequate trigger vector (u_k) in each situation. The equation that defines the next marking is as follows (Eq. 1):

$$M_{k+1} = M_k + A^T \times u_k \quad k = 0, 1, 2, \ldots, n \tag{1}$$

If the result is a marking with negative values, it means there weren't enough tokens to fire the transition, and the result of the firing is perceived as an exception. This exception analysis allows the PNRD to automatically detect and handle errors at runtime.

In this context, a set of information is needed to determinate the trigger vector, this set may include the tag ID, tag state, antenna ID, and other additional data [8]. In a similar way, a transition is not necessarily linked to a unique antenna. Consequently, the addition of more antennas performing the same type of function does not result in a change in the system control. This allows the system to be able to include more machines without undergoing a major modification on its operation. Therefore, the

PNRD approach helps in the scalability of the process control system. Another feature of the PNRD is the distribution of process data in the tags and readers, reducing the necessity to query a central database, speeding up the data flow and lowering local network costs.

If the PNRD approach can be used to control passive items through a workflow, the inverted PNRD (iPNRD) can be used to control active agents that moves in different places [7] generating a variation of original PNRD. In this approach, the tag contains the trigger vector rather than the token vector and the incidence matrix. This variation was developed in the context of search and rescue of people on walking trails for robots. It makes the PNRD concept practical in applications in which the installation of RFID readers at transition points is too costly. The inverted iPNRD uses mobile RFID antennas to read labels positioned at fixed transition points. Thus, the RFID reader is responsible for storing the marking of the system, while the tag determines its transition. An exception in this context indicates that the RFID antenna carrier is following the wrong path or performing an action it is not supposed to do. This new approach also brought a need to store a history of the readers that visited the tag. In the search and rescue application, for example, it would be essential to determinate which people went through a route.

3 Library Overview

The library was developed using the object-oriented programming paradigm to promote the understandability and usability of the library. The Arduino® Library developed is divided in three modules, which were organized in a top-down hierarchy [9] as indicated by the Fig. 1. The Petri nets module is responsible for the mathematical base of the library. It defines the marking, trigger vector and incidence matrix, computes the resultant marking in firing events and check for exceptions. The PNRD module integrates the RFID system with the Petri nets module. It can select which kind of information is going to be stored on the tags. Consequently, it can support iPNRD applications and other variants. It can also automatically create a tag history containing the state of the tag, the reader Id and the timestamp of the more recent fire events. The PN532 Reader module handles the communication between a tag and a specific reader installed in an Arduino® - the Elechouse's NFC PN532 RFID module V3.

Each module intrinsically depends only on its predecessor. In other words, the Petri nets module can be used separately for purposes other than developing PNRD applications. As it contains the mathematical base of Petri nets firing, it may be useful for developing Petri-nets-based state-machines or communication protocols. In the Petri nets module each transition can be associated with a Boolean condition. The transition can only be fired if the associated condition is true. This tool is used so that the application can define which transition to fire in the case of a conflict. The PNRD and Petri nets modules are independent from the third module to allow these modules to be compatible with different types of RFID readers. This effect was achieved through the usage of an abstract adapter class of the PNRD module that can be used as base for implementation of different kinds of readers. The PN532 Reader module contains the implementation for the Elechouse's PN532 reader, however others can be added in future development.

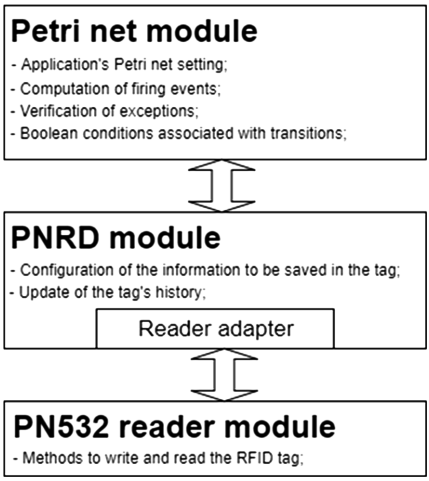


Fig. 1. Architecture of the library modules and their main features

The software development was guided by the constraints faced when dealing with embedded systems. The very limited memory size and processing power of the Arduino® platform compelled the development process to focus on the optimization of the core methods and memory usage of its data structures. Having those considerations in mind, instead of organizing the encapsulated data to match an incidence matrix, the arcs information is stored in adjacency lists. Each transition has two adjacency lists, one containing its inputs places and other containing its output places. Therefore, in the case of a fire event, instead of testing each element of a matrix column to search for inputs and outputs of the transition, it is only necessary to go through the lists. In practical cases, the adjacency list data model will be more efficient to calculate fire events and often be less memory consuming. Most of the Petri nets models have a very limited number of input and output places per transition. Elementary Petri nets, which are the basis of PNRD, have this characteristic in the extreme. To stay safe and its marking to be limited to one token, each transition must have one input and one output only.

In an incidence matrix, each element consumes two bits of information, because it can have three different values: 1, 0, −1. In the adjacency list each element consumes the quantity of memory large enough to store the code of the place. The library uses an 8-bit code to define a place. Consequently, for safe Petri nets with more than 4 places, the adjacency list data model becomes less memory consuming. There were other aspects analyzed before deciding for the adjacency list approach, the resume of these analysis is shown in Table 1. The main downsides of applying an adjacency list structure is an increase of the complexity in the setting of the Petri net's arcs and in the determination of the relationship between a determinate place and a transition. The first is an operation that occurs mostly in the configuration process, executed only once in an application. The second operation is rarely necessary. Therefore, the better performance in the firing computation, which occurs continuously and several times in most of the applications, makes the tradeoff worthy.

Table 1. Analysis of the incidence matrix and adjacency list data structures performances.

Comparison parameters	Structure Comparison		Better performance structure
	Incidence matrix	Adjacency list	
Necessary memory (bytes)	$\frac{t \times p}{4}$	$t(max_{in} + max_{out})$	Depends on the Petri net density
Setting an arc	O(1)	O(k_{in}) or O(k_{out})	Incidence matrix
Determinate the relation between a transition and a place	O(1)	O(k_{in} + k_{out})	Incidence matrix
Verifying if a transition firing results in exception	O(p)	O(k_{in})	Adjacency list
Computing a transition firing event	O(p)	O(k_{in} + k_{out})	Adjacency list

t = number of transitions;
p = number of places;
k_{in} = quantity of places in the input list;
k_{out} = quantity of places in the output list;
max_{in} = maximum quantity of input places;
max_{out} = maximum quantity of output places;

The library also tries to avoid memory fragmentation, which is a problem that may occur when the application constantly reallocates memory during its execution. The result of this action is the portioning of blocks of free memory separated by blocks of memory in use, in such a way it becomes impossible to allocate a large block of memory, because none of the available blocks have the required size. This problem is commonly addressed by the operational system or by allocation algorithms that reorganize the memory blocks. However, the Arduino® platform is deprived of an operational system and the usage of allocation algorithms capable of rearranging memory blocks would come with a computation cost and a EEPROM memory cost, reducing the library performance and the total memory available for the application code. The solution found to this issue is to use mainly static size data structures, to avoid reallocation of memory. This choice makes the library more reliable in long-term execution, however it limits the application. One example of limitation caused by this option is that it is mandatory to inform the maximum quantity of transitions and places upon the creation of an object of the *PetriNet* class. Therefore, any tags containing a Petri net model with a bigger quantity of transitions and places can't be treated by this object. It is also recommended to define the maximum amount of outputs and inputs that each transition can have, so the data structure occupies less memory.

Because of the current limited size of the RFID tags — 128 KB at most — the library has a limitation of a maximum of 255 places and 255 transitions for the Petri net model. Due to the possibility of future adaptation of this library for custom assembled embedded systems, the size of the Arduino® memory wasn't taken into account in the design of this intrinsic limitation. Therefore, depending on the application configuration, on the type of Arduino® used and on the RFID tag used, the size limitation can be more restrictive.

To implement both iPNRD and PNRD applications, the library allows the setting of which kinds of information will be saved in the tags. Considering an application using classical PNRD, the adjacency list information and the token vector must be saved in the tag. In contrast, in a iPNRD application the trigger vector must be stored in the tag.

As it is possible to set separately each of the data structures to be recorded in the tag, it is possible to create other variations of the PNRD and iPNRD approaches, although only PNRD original and iPNRD original have been in use. Table 2 specifies the main possible arrangements of data between the reader and the RFID tag.

Table 2. Arrangement of data between reader and RFID tag

Formalism		Reader	Tag
PNRD	Original	u_K	M_K and A^T
	Variant 1	A^T	M_K and u_K
	Variant 2	M_K	A^T and u_K
iPNRD	Original	M_K and A^T	u_K
	Variant 1	M_K and u_K	A^T
	Variant 2	A^T and u_K	M_K

In addition to these Petri net-based information, the library also makes it possible to record the Boolean conditions state in the tag as well as a tag history, with the result of the latest fire events of the tag.

4 Application Implementation

The usage of the library to implement a PNRD application can be divided in three phases.

1. Firstly, it is important to set the initial configuration of the application. The program needs to instantiate the reader and PNRD objects, start the reader communication and define the data structures to be recorded on the tag.
2. After that, the application can retrieve the information from a tag and realizes a firing and other associated actions, depending on the tags state.
3. Finally, the application updates the information in the tag, so the system can continue its workflow.

To illustrate a simple PNRD application, consider a factory that produces bearings. After the production of a specific part, it is necessary to inspect evaluating the roughness of its surface. If the test fails, the part is placed in a queue to be reworked. If the tests conform to standards, the piece is stored in stock.

Considering a PNRD approach, each machined part is endowed with an RFID tag. This tag contains the Petri net incidence matrix information and the current tag vector. At the beginning of the process, the initial marking is the same as that shown in Fig. 2.

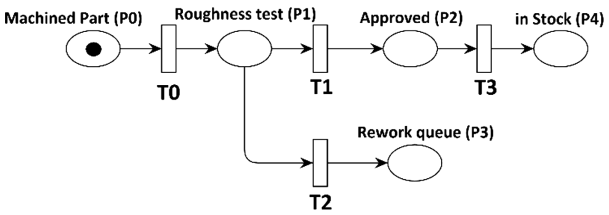


Fig. 2. Petri model of the verification process of a tagged bearing part

The machine that checks the cylindricity of the piece is equipped with an RFID reader/antenna. The reader/antenna, when perceiving a new part, acquires its data, in other words, its marking and its incidence matrix. It performs the transition T_0 firing, computes the tag next state and updates the tag information. Then, in accordance with the new tag status, it sends a command to begin the test. Depending on the test result, the verification system sends a message to the PNRD application. Depending on the message received, the PNRD application fires whether the T_1 or the T_2 transition and updates the tag's new token vector. After firing the transition T_1 or T_2, the piece is sent to recycling or to the stock. Both places are equipped with RFID antennas that read the label and try to fire a new transition. If the result of this firing is an exception, the system realizes the piece was not sent to the correct place. Likewise, untested parts sent to the stock will be detected.

To simulate this system, three different Arduinos® Mega were used, each one equipped with a different RFID reader. The first one made the initial recording of the tag, storing the information of the incidence matrix of the process and the initial marking. The second one is the machine responsible for the verification of the parts. The test was simulated with two buttons as illustrated in Fig. 3, the yellow one indicated that the part was approved, the red one indicated it was reproved. The last Arduino® was the stock manager. It determinates whether the part could be put on the stock or not.

The result of this simulation can be seen in the Fig. 4 that indicates the Arduino® 2 and 3 log messages. The tags with ids *DF* and *EF*, were accepted in the test, so they could be stored in stock. The tag id *6F* was rejected, therefore it caused an exception. After the exception the system indicates that the tag has a token in P_3 (rework queue).

As an example of an iPNRD application, it was developed the code for the simulation of two different robots. One roaming a circle of tags clockwise through five different regions, and another roaming the circle anti-clockwise using the same regions. Each tag contained a trigger vector with one different transition is settle down in each region. The only difference between the robots was the initially defined incidence matrix.

The Petri model for the clockwise robot, a physical schema, and the result of this simulation is presented in the Fig. 5. The PN model presents five places (P_0 to P_4) related with each region, and five transitions (T_0 to T_4) attached to each tag. P_0 is the initial marking. The figure contains a scheme of the tags spatial disposition as well as the log messages of a robot that should follow the clockwise sense. However, after the transition T_3, it goes the wrong way and returns to the tag containing the transition T_1. The result is an exception.

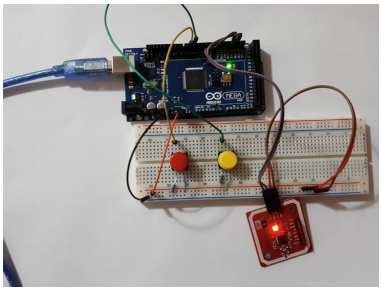


Fig. 3. The assembly of an Arduino® simulating a machine that realizes roughness tests. (Color figure online)

```
Found chip PN532                    Found chip PN532
Firmware ver. 1.6                   Firmware ver. 1.6

Machine 2: Tester                   Machine 3: Stock manager

New part. Part ID: DF.              Part with ID: E5.
Part ready to test.                 Part ready to go to stock.
Part approved.
Ready to go to stock.               Part with ID: DF.
                                    Part ready to go to stock.
New part. Part ID: 6F.
Part ready to test.                 Part with ID: 6F.
Part rejected.                      Error: exception. Part followed wrong workflow.
Ready to rework.                    Token vector:
                                    0
New part. Part ID: E5.              0
Part ready to test.                 0
Part approved.                      1
Ready to go to stock.               0
```

Fig. 4. Log messages from Arduinos® 2 (on the left) and 3 (on the right).

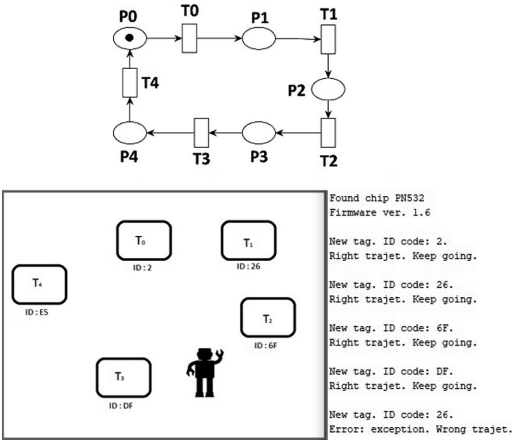


Fig. 5. Petri model and simulation of a robot following a circular track of tags clockwise. On the left the scheme of the tags, on the right the robot log messages.

Those examples are illustrative problems in which the library could be applied. However, real applications tend to have more intricate workflow and, consequently, a more complex Petri net model. Although the increase of the model complexity, the control process remains the same. It continues following the simple steps of configuration, getting the tag data, performing a firing event, and updating the tag information. Consequently, in complex systems it becomes easier to use this kind of approach than a set of intricate network messages. It is noticeable that even the communication between devices is facilitated because the process information flows alongside the tags.

5 Conclusion

The Arduino® library described in this article is a developing interface for the implementation of PNRD and iPNRD applications as well as other variants of distributed data based on Petri nets and RFID systems. The fact the library was developed for an embedded system platform causes it to have certain limitations regarding the size of the Petri net model of the application. Nevertheless, it can be used in several contexts as a developing framework for process control systems. It also can be associated with Petri net analysis tools to store a more reliable and efficient PN process.

The library and the PNRD/iPNRD transform the system into a distributed system in which the logic is spread between devices using well known and low-cost microcontrollers. There are three PNRD implementation types and three iPNRD implementation types and only one was explored. The original PNRD is usually applied to manufacturing industries where the product follows a path in the production line and there is a need to monitor the evolution of the product through the factory. The PNRD variant 1 or iPNRD variant 1 can be applied to update systems configuration. In this case this approach can be a setup for the system to change its' Petri net data. Studies on these variants should be further developed. The PNRD variant 2 also works on manufacturing industry as a reactive system. A tag attached to a product can change the marking in a machine, so another action can be trigged in an assembly line. This approach does not work as a exception state detection such as the original approach. The original iPNRD can be applied to mobile robots in swarms where the robot carries the marking and the incidence matrix, while the environment or another device carries the trigger vector. In this scenario, the marking is related to robot's behavior. The iPNRD variant 2 is applied to systems where the tag stores only information of the marking, such as, in a timed traffic light where the tag must store the color (red, orange, or green), and the reader has the trigger vector and the incidence matrix to toggle colors.

In further development, this library will be used to control the behavior of cooperative robots and can be applied as an easy to implement, scalable, and error handling control tool. New types of readers must be incorporated.

The PNRD approach must be integrated with typical centralized RFID solution and a complete centralized and distributed system can be designed to be complementary and contingent to each other.

The library developed is open source and available at: ftp://ftp.mecanica.ufu.br/LIVRE/MAPL/ArduinoPnrdLibrary.zip. It presents some examples that illustrates how to use the library.

Acknowledgement. CAPES, CNPQ, FAPEMIG and UFU supported this research.

References

1. Nath, B., Reynolds, F., Want, R.: RFID technology and applications. IEEE Pervasive Comput. **5**(1), 22–24 (2006)
2. Ahsan, K., Shah, H., Kingston, P.: RFID applications: an introductory and exploratory study. Int. J. Comput. Sci. Issues **7**(3), 1–7 (2010)
3. Kristensen, L.M., Taentzer, G., Vaupel, S.: Towards verification of connection-aware transaction models for mobile applications. In: Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2017), pp. 227–228 (2017)
4. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
5. ter Hofstede, A.H., van der Aalst, W.M.: YAWL: yet another workflow language. Inf. Syst. **30**(4), 245–275 (2005)
6. Tavares, J.J.P.Z.S., Saraiva, T.A.: Elementary Petri nets inside RFID database (PNRD). J. Int. J. Prod. Res. **48**(9), 2563–2582 (2010)
7. Fonseca, J.P.S.; Tavares, J.J.P.Z.S.: Petri net with RFID distributed database for autonomous search and rescue in trails and crossings. In: Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2017), pp. 229–230 (2017)
8. Tavares, J.J.P.Z.S., Murofushi, R.H., Silva, L.H., Silva, G.R.: Petri net Inside RFID database integrated with RFID indoor positioning system for mobile robots position control. In: Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2017), pp. 157–176 (2017)
9. Meyer, B.: Object-Oriented Software Construction, 2nd edn. ISE Inc., Santa Barbara (1997)

OMPetri - A Software Application for Modeling and Simulation Using Extended Hybrid Petri Nets by Employing OpenModelica

Christoph Brinkrolf(✉) and Philo Reipke

Bioinformatics Department, Faculty of Technology,
Bielefeld University, Bielefeld, Germany
cbrinkro@cebitec.uni-bielefeld.de

Abstract. In this paper we present OMPetri, a new tool for modeling, simulation, and analyzing of a powerful unifying Petri net concept: extended hybrid Petri nets (xHPN). The software features a modern, lightweight, and intuitive graphical user interface that focuses on users who are new to Petri nets, while also enabling experienced users to model more complex and advanced systems. It has not been designed for any specific application cases. Thus it is as universal as the related Petri net formalism that can be applied for modeling systems of various research areas, such as systems biology, business processes, and industrial workflows.

It greatly enables and simplifies the modeling of discrete, continuous, and hybrid Petri nets. New models can be created and existing models can be modified and improved quickly and easily due to two different model views. During the process of modeling, the Petri net may change its class (discrete, continuous, hybrid) and solving strategies for conflicts can be defined.

The tool employs OpenModelica and the advanced Modelica Petri net library PNlib to provide an elaborated and powerful simulation environment. Additionally, it provides basic features to check and evaluate the model and to analyze simulation results generated by the simulation back-end.

Both, the tool, as well as OpenModelica are open source and free of charge for academic usage. Java source code, executable JAR and a tutorial are available at: https://agbi.techfak.uni-bielefeld.de/OMPetri.

Keywords: Petri net editor · User-friendly · Modeling · Simulation
Extended timed hybrid functional Petri net · xHPN · JavaFX
PNlib · OpenModelica

1 Introduction and Objectives

Modeling has always been essential for science and engineering to understand and possibly to acquire new insights into complex systems. Due to decreasing costs for

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 406–415, 2018.
https://doi.org/10.1007/978-3-319-91268-4_23

computational resources, modeling became increasingly popular for applications aside of classical scientific research, and it got more and more relevant for topics such as business workflow or industry process optimization [1].

Modeling and simulating complex systems, nowadays, is often cheaper and faster than performing real-world experiments, but the generation of a sensible model requires existing knowledge of the system. This data might have been acquired in previous experiments. The model based on such data should then be able to reproduce the given results, and testing new hypothesis on a good model can then lead to new insights of the modeled system.

There exist various modeling approaches, such as formal languages/ automata, ODEs, and graphs. The basic formalism of Petri nets [2], which can be represented as graphs, has been extended over the past years. While this also extended the general modeling capabilities of Petri nets, additional simulation environments and software for analyzing generated results made them an even more powerful approach.

The software environments for modeling, simulating and analyzing Petri nets available today feature many different Petri net concepts and formalisms. However, these environments have often been designed for specific scientific topics, mostly related to synthetic or systems biology (i.e. Cell Illustrator [3] and its predecessor Genomic Object Net [4]). Additionally, the underlying Petri net concepts mostly have not been well defined, so that the user does not have all information about the simulation process to fully comprehend the simulation results. For example, the usage of proprietary software and non-complete specified behavior for resolving conflicts (e.g. Snoopy [5]) are reasons for missing transparency and black-box simulations [3,4].

We specified the following requirements for a modern and powerful Petri net modeling and simulation environment that could possibly minimize most limitations of already existing solutions:

- use a timed Petri net formalism, supporting at least HFPN [6]
- fully support this formalism in an easy to use graphical user interface (GUI)
- validate for model inconsistencies (e.g. syntax of mathematical expressions)
- support hierarchical structures
- autonomous and transparent simulation
- visualization of simulation results
- persistent models (import and export)
- best practice for code design and structure, modern technologies
- use open source software for Petri net formalism and simulation.

The xHPN formalism [7] is a powerful formalism for timed hybrid functional Petri nets that is also featuring stochastic and functional components, as well as conditions and strategies for conflict resolution. It is implemented as a Modelica [8] library called PNlib [7]. The comprehensive library, as well as the Modelica compiler shipped with OpenModelica [9], are both open source. Unfortunately, there is no editor which supports all aspects of PNlib in a user friendly way, until today.

2 Simulation Back-End Technologies

The presented software does not implement a new, but relies on an existing, well-elaborated and open source solution for its simulation back-end that features OpenModelica and the Modelica language-based library PNlib.

2.1 Modelica

Modelica is a non-proprietary, object-oriented, equation-based language for component-oriented modeling of complex systems developed by the non-profit Modelica Association [10]. The free Modelica Standard Library offers a large set of models, for example, libraries for electrical, magnetic, mechanical, or fluid components, control systems, and functions.

2.2 xHPN and PNlib

The xHPN formalism has first been implemented in the Modelica library PNlib to enable the graphical hierarchical modeling, hybrid simulation, and animation of processes in life sciences using timed Petri nets. Since then it has been enhanced and improved to enable the modeling of various kind of processes, including any technical or industrial applications. The PNlib allows to model all features associated to an xHPN and transparently implements the activation and firing process as well as solutions for conflicts. Additionally, it allows to assign physical units to markings or transitions functions that can be validated in order to assist the detection of syntax errors in mathematical expressions.

2.3 OpenModelica and the OMC

The Modelica open source environment OpenModelica is a modeling and simulation environment for industrial and academic applications that is developed by the Open Source Modelica Consortium (OSMC) [11]. Its goal is to create a comprehensive open source Modelica modeling, compilation, and simulation environment based on free software, distributed in binary and source code for research, teaching, and industrial usage that allows the reuse and exchange of models in a standardized format. While OpenModelica comes with a set of programs with graphical user interfaces, its heart is the OpenModelica compiler (OMC). This compiler can process Modelica source code and generate executable binary code, for example a simulation of a PNlib model.

3 User Front-End Technologies

The front-end represents the actual software environment that provides all software features to the user and maintains the connection to the simulation back-end. It is written in Java 8, featuring JavaFX and the widely adopted Spring framework.

3.1 JavaFX

JavaFX was completely revised in version 2 before it was finally released as a fully-fledged part of JavaSE, making it the new standard toolkit for developing graphical user interfaces in Java 8 [12]. Swing still exists but is since then in maintenance-only mode, meaning that no new functionality will be added to it in future. JavaFX is a toolkit for desktop (and potentially mobile) application development, supporting model-view-controller (MVC) type architectures via FXML files for defining a (mostly) passive view which can specify controller classes using an attribute on the root element. This feature has been used to decompose the GUI into various smaller, independent and therefore maintainable modules.

3.2 Spring

The Spring framework [13] provides a comprehensive programming and configuration model for modern enterprise applications with core features such as dependency injection, aspect-oriented programming, MVC concepts, and Java database connections. It usually finds heavy adoption for developing web services. It has been implemented in OMPetri for its dependency injection and inversion of control features that greatly simplify connecting of components throughout an application.

4 Design and Implementation

To facilitate future extensions, maintainability and reusability, the application implements two models that were designed and implemented in separate libraries: data model and graphical model.

4.1 Data Model

The data model enables storing of Petri nets. It covers all elements and properties of xHPN, as well as providing some additional features that further extend the usability inside the developed software. This includes:

- arcs, places, transitions, and individual subtypes
- colours, weights, tokens
- functions, and function expression validation
- parameters, usable inside functions.

The model allows the modification of its components (adding, removing, and editing of properties) while autonomously maintaining the model integrity. The removal of an element is always validated before it is performed, and all remaining references and related objects will be removed upon successful deletion as well.

The data model library also provides the basic connection to the PNlib and OpenModelica. Any model can be converted to a Modelica model which is important for the process of simulation.

4.2 Graphical Model

The graphical model has basically been designed to enable visualization of any kind of directed or undirected graphs, providing a set of basic shapes that can be styled dynamically using CSS. It is developed in a separate library and inside the application it acts as the mediator between user and data model, allowing to handle the visual representation separately from the Petri net data. This enables functionalities, such as hierarchical structuring, without altering the model itself. The chosen visualization for Petri nets adapts the visualization of HFPN [6]. Discrete elements have a single border and continuous elements have a double border.

4.3 Architecture

The software has been implemented featuring a modern multi-layer architecture in which presentation (view, controller), application processing, and data access are separated. This highly facilitates future changes and extensions to the application. Interaction between the various layers is facilitated by using dependency injection.

View. The view (or GUI) has been implemented using FXML and is highly modularized, each window consists of multiple components that are joined upon construction. Each frame gets assigned its own controller class. This highly increases code readability and maintainability of controller classes.

Controller and Handler. The controller and handler classes manage all user interactions, transform user inputs, and interact with the application processing (service) layer.

Processing. Application processing is managed by various service classes that mediate between the presentation layer (controller/handler) and the data access layer (data/graphical model). This layer is used for validating user requests, producing responses, and other nonspecific functionality.

Data Access. Access to the data and graphical model is granted only through data access objects. They autonomously manage the application's model integrity by ensuring that no references remain inside the model upon removing an entity. Removing a place will also remove all connected arcs and related parameters.

Simulation and Communication with OMC. For simulation, the selected Petri net model is exported as a Modelica model. The OMC then compiles this Modelica model using the PNlib and generates an executable simulation. The simulation is executed while simulation results are sent via TCP/IP as a byte stream to OMPetri and get processed and stored in OMPetri.

Data Import/Export. A Petri net model, as well as simulation results, can be stored and loaded as XML file. Additionally, a JSBML import for a Petri net is supported as well.

5 Graphical User Interface and Features

The GUI is composed of three different windows which are presented in the following, as well as their main features.

5.1 Main Application Window

The main window features the general modeling and editing functionalities, including the creation, export and import of models. To enhance operationality and to improve support of working on large-scale Petri nets, two different views are available: general graph view and inspector (or explorer) view.

Graph View. The graph view enables the user to create new and edit existing Petri net models using a lightweight and well-structured interface which is shown in Fig. 1. All menu panes can be minimized and expanded. While the general panes on the left, such as model info, tools, and simulation options, are always enabled, the identifier and property panes are only available when an element has been selected. Element properties can be edited quickly by using the available input fields. Malformed or conflicting inputs will be highlighted, and the changes rejected. The graph view also enables the user to structure its model hierarchically, increasing productivity and the general readability of large-scale models. Selected nodes can be clustered to a hierarchical node (tools pane *Cluster*) and a hierarchical node can be flattened by *Restore*. If a selected place enables a conflict, a conflict resolution strategy (probability or priority) can be chosen. For each corresponding edge, if selected, the value of the resolution strategy can be set. Options for simulating an active model can be specified in the simulation pane, and the simulation can be started and stopped.

Features: visualization, modeling (adding/removing/connecting), editing properties, input validation, hierarchical structuring.

Inspector View. The inspector view, shown in Fig. 2, features a different kind of approach than the graph view. Instead of graphically visualizing the model structure, the user can here work on the mere data. This is best suitable when the general model has already been established and only certain element properties must be changed. All available elements can be accessed from a list that can be filtered using element names and/or labels. Selecting an element grants access to its properties, also highlighting elements that are related, which are usually all neighboring places/transitions and/or the connecting arcs.

It also grants access to the list of parameters that can be used inside the functions of arcs and transitions. Parameters feature different types: global (can

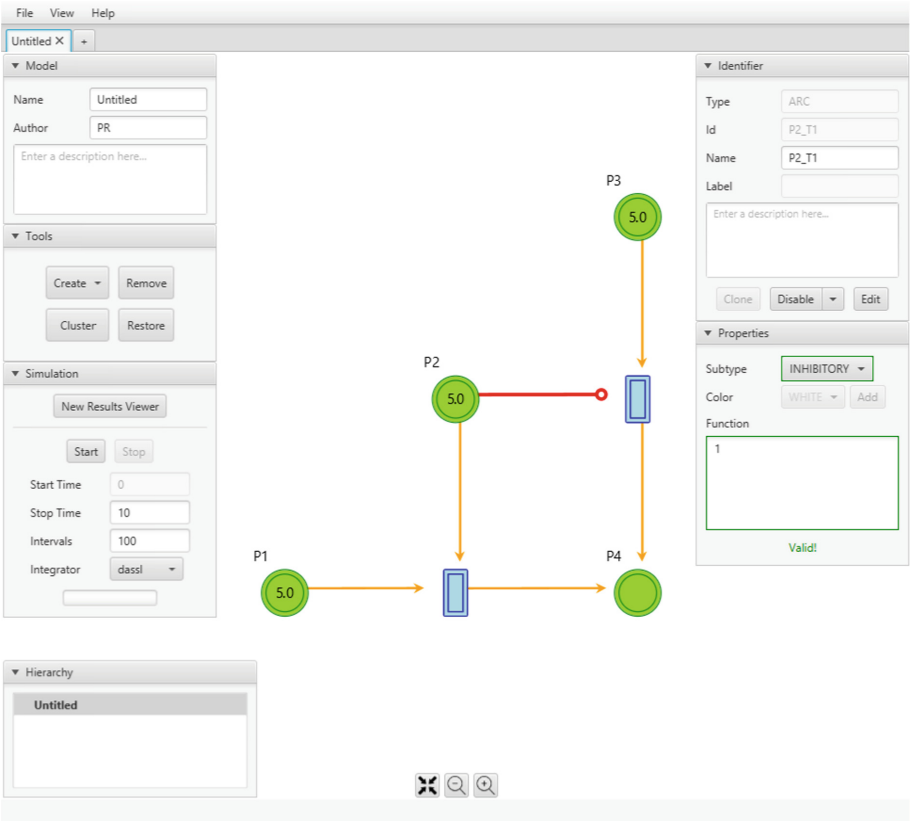


Fig. 1. Shows the interface of the graph view inside the main application window. A model is always stored inside a tab, and the software can manage multiple tabs at the same time. All titled panels in the graph view can be minimized or expanded dynamically, and the panels on the right side are only visible when an element is being selected, allowing to quickly edit element's identifiers and properties.

be used across the entire Petri net), local (only usable for a specific element), and reference (related to the value of an element during simulation, e.g. the current token count of a place). While local and global parameters can be created by the user, referencing parameters are managed by the application.

To support the creation and maintaining of functions, the inspector view provides an enhanced visualization of mathematical expressions by rendering them using a LaTeX library. This comes along with a general syntax validation, as well as ensuring that all used parameters are valid. In functions, a user can reference the current token count of places or the current firing rate/speed of transitions by simply typing an element's name. In the background, the application will then create a referencing parameter, linking it to that distinct element. Future name changes of referenced elements will automatically reflect to the names specified in a function.

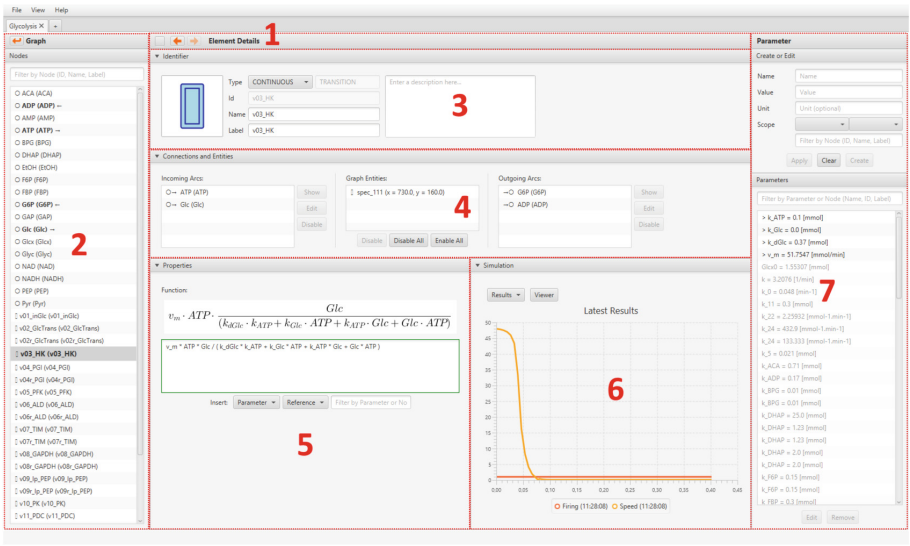


Fig. 2. Outlines the modularization and features of the element inspector view. (1) Navigation bar. (2) List of nodes. (3) Lists the element identifiers. (4) Lists elements related to a select element. (5) Lists element properties. (6) Displays latest simulation results. (7) Lists parameters and allows creation, editing, and removal of parameters.

Features: data listing, editing properties, input validation, function printing, parameter management.

5.2 Result Viewer Window

The results viewer, shown in Fig. 3, features basic results visualization and analysis features for generated simulation results. The user can open multiple windows at the same time. Results can be added individually or for all places or transitions of a model at once, displaying token counts over time, transition firing rates or enabled-states, and/or the token flow across selected arcs. A handy feature is that already chosen elements can be selected and future results are added to the current chart automatically. Using checkboxes in the list of selected results, data can be dropped, disabled, and/or added/removed for future simulation data added to the chart.

Additionally, the application can export results using a combination of XML and CSV. This enables the import of previous results and allows to reassign any results to their models and elements.

Features: multiple windows, individual result selection, result comparison, auto-adding results, result import.

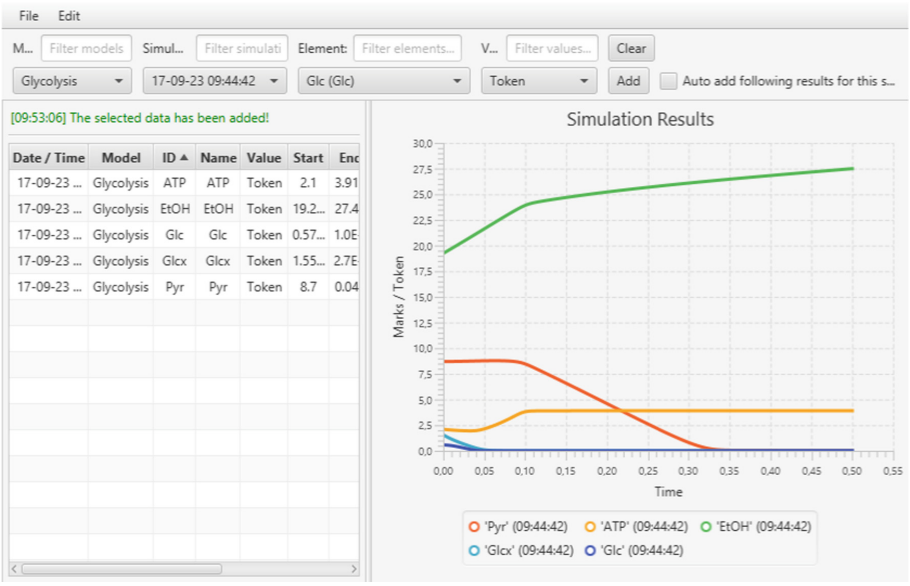


Fig. 3. Shows the result viewer window. The table on the left stores all data that has been selected to be displayed in the current window.

5.3 Logging Window

The application also features its own logging service that currently forwards all captured messages to an additional logging window. The window opens itself on startup, but it can always be re-opened from the main window. Generally, the user should not have to worry about any serious error messages being generated. However, it can still give additional information on why an action did not take place as expected, for example when editing a model. Also, the logging window prints all information that is generated when generating and running a simulation using the OMC.

Features: notification logging, error logging, simulation logging.

6 Conclusion and Outlook

Petri nets are widely used to model and simulate various systems in research and industry. The more complex the systems get, the more features have to be available in the chosen Petri net formalism. Beside the formalism, the transparency of the simulation is important to fully understand the simulation results. Unfortunately, there is no such Petri net modeling and simulation environment for hybrid functional Petri nets. OMPetri closes this gap.

It features the xHPN formalism, implemented as Modelica library PNlib, a formalism which also considers conflict resolution strategies, and simulations are generated using the OpenModelica compiler. All components are open source.

Another advantage is its graphical user interface which is easy to use but still comprehensive, providing access to all properties defined by the xHPN formalism, supporting hierarchical modeling and the use of parameters and mathematical expressions which get validated and rendered using LaTeXengine, a feature that is important especially for functional modeling.

The usage of modern technologies and its architecture facilitate future extensions. Upcoming versions of OMPetri will feature the support of different time concepts for transitions and the support of colored Petri nets.

References

1. Dotoli, M., Fanti, M.P., Giua, A., Seatzu, C.: Modelling systems by hybrid Petri nets: an application to supply chains. In: Petri Net, Theory and Applications, pp. 91–112. InTech (2008)
2. Petri, C.A.: Dissertation: Kommunikation mit Automaten. Schriften des Rheinisch-Westfälischen Institutes für Instrumentelle Mathematik an der Universität Bonn (1962)
3. Nagasaki, M., Saito, A., Jeong, E., Li, C., Kojima, K., Ikeda, E., Miyano, S.: Cell illustrator 4.0: a computational platform for systems biology. Silico Biol. **10**(1), 5–26 (2010)
4. Nagasaki, M., Doi, A., Matsuno, H., Miyano, S.: Genomic Object Net: I. A platform for modelling and simulating biopathways. Appl. Bioinf. **2**(3), 181–184 (2003)
5. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy – a unifying petri net tool. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 398–407. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31131-4_22
6. Matsuno, H., Tanaka, Y., Aoshima, H., Doi, A., Matsui, M., Miyano, S.: Biopathways representation and simulation on hybrid functional Petri net. Silico Biol. **3**(3), 389–404 (2003)
7. Proß, S., Bachmann, B.: Pnlib - an advanced Petri net library for hybrid process modeling. In: Proceedings of the 9th International MODELICA Conference, Munich, Germany, 3–5 September 2012, no. 76, pp. 47–56. Linköping University Electronic Press, Linköpings universitet (2012)
8. Fritzson, P., Bunus, P.: Modelica - a general object-oriented language for continuous and discrete-event system modeling. In: Proceedings of the 35th Annual Simulation Symposium, pp. 14–18 (2002)
9. Fritzson, P., Aronsson, P., Lundvall, H., Nyström, K., Pop, A., Saldamli, L., Broman, D.: The openmodelica modeling, simulation, and software development environment. Simul. News Eur. **44**(45), 8–16 (2005)
10. Modelica: Modelica association. https://www.modelica.org/. Accessed 22 Jan 2018
11. OSMC: Open source modelica consortium. https://openmodelica.org/home/consortium. Accessed 22 Jan 2018
12. Oracle: Java 8 documentation. https://docs.oracle.com/javase/8/docs/. Accessed 22 Jan 2018
13. Pivotal Software: Spring framework. https://projects.spring.io/spring-framework/. Accessed 22 Jan 2018

GreatTeach: A Tool for Teaching (Stochastic) Petri Nets

Elvio Gilberto Amparore(✉) and Susanna Donatelli(✉)

Dipartimento di Informatica, Università di Torino, Turin, Italy
{amparore,susi}@di.unito.it

Abstract. GreatSPN is a collection of tools for modelling and analysis of systems using (stochastic) Petri nets. It features a modern Java-based graphical interface. But being technologically advanced does not mean that the tool learning curve is significantly simplified. In the tool the simple features that are needed for educational purpose are intermixed with more advanced features that require a deeper understanding of the formalisms and of the solvers. This paper presents GreatTeach, a streamlined and enriched version of GreatSPN meant for teaching resulting from our experience in teaching Petri nets to master students.

Keywords: GreatSPN · Teaching · Modelling · Verification

1 Introduction

Tools play a central role in teaching formal methods: most classes we know of use some form of tool support to teach students the basic concepts and to let them experiment the learned concepts on a (more or less realistic) examples with the help of a tool. Petri nets are very appealing when it comes to teach formal methods: the basics of the formalism are very simple (just the enabling and firing rule), and there is a widely accepted graphical representation. But Petri nets comes also in a large variety of definitions, with a rich set of analysis techniques, so typically Petri nets tools are rather rich in features. Moreover many tools also support the extension of Petri nets to consider time (non deterministic or stochastic), which increases the complexity. Once decided to teach Petri nets in a course, the teacher has a wide choice of tools that have rather diverse characteristics: from very simple educational tools that concentrate only on the token game, visualization of the incidence matrix and firing rule exemplification like in [15], to very powerful tools like CPN-tools [14] a tool for constructing, simulating, and performing analysis of Colored Petri Nets (CPN) models (timed and untimed). CPN-tools has a large community of users, a book [13] is available to learn the formalism, the tool and its industrial applications; the book also includes a chapter on teaching CPN as a full course. A feature to automatically grade student exercises (the Grade/CPN tool) is also available. The tool is quite tight to colored Petri nets and the graphical interface has some uncommon features, not obvious for newcomers. In between there are a large

V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 416–425, 2018.
https://doi.org/10.1007/978-3-319-91268-4_24

number of tools stemming from research groups that are made available. Examples of tools for the definition and analysis of P/T nets, Stochastic Petri Nets (SPN) and various extensions are the suite made of the trio Snoopy, Charlie, and Marcie [11,12,16], GreatSPN [3], and TimeNet [20]. There are also tools that have been initially developed for teaching, in particular as a platform for letting students (and other researchers) add new analysis techniques, like, for example, PetriDotNet [19] and the Snoopy suite to a certain extent. A comprehensive survey can be found in [17], while [19] includes a comparison table for various tools.

The authors of [8] advocate that students should use industrial-size tools: indeed while it is true that a simple tool with few features it is likely to be more portable and easier to learn, it is also true that, having to change tool to access to more advanced features can significantly slow down the learning process. As far as we know there is no tool that has been explicitly designed with the objective of offering at the same time a rich set of analysis techniques and an easy-to-use tool for lab classes.

Motivations and Objectives. As a teacher of a course in verification of concurrent processes for master students in computer science at the University of Torino, one of the authors of this paper has taught for many years various formalisms and the associated analysis techniques. Students in the course learn and experiment (colored) Petri nets and CTL with the support of GreatSPN, guarded command languages and LTL/CTL with the support of nuSMV [10], and timed automata with Uppaal [6]. The use of GreatSPN in the course has been one of the main drivers behind the introduction of the new Java-based graphical interface (GUI) of GreatSPN [2] a few years ago. Although with the new GUI we observed a great increase in acceptance and appreciation from the students (GreatSPN surpassed Uppaal in the preferences of the students), the learning curve of the tool proved to be still quite steep. Observing the students during the lab classes and examining their questions and e-mails we realized that the students spent a significant amount of time in struggling among the numerous options and features offered by the tool. And this aspect was not mitigated neither by the precise indications given during lab classes nor by the availability of a number of video tutorials. From our "observation in the wild" we concluded that teaching requires the introduction of a "limitation of scope" in the tool.

We also observed, as many other teachers have done (see for example [8]) that the learning approach of the students has significantly changed over the years and that more and more the students tend to use the "learning by example" approach. The authors of [8] conclude that "formal methods are best taught by examples". In our course we do not totally follow this paradigm, since the course has also the learning objective of reading and writing formal statements and definitions, but over the years we have inverted the order of presentation of the material: we start with examples using the tools (we draw a net and we play the token game) and drive back to the formal definitions, possibly with a number of iterations. Of course to start with examples you need an adequate tool support: for each learning objective there should be adequate *graphical* tool

support. For example teaching decision diagrams (DD) for storing a reachability set (RS) is much easier if the tool also visualize the RS in DD form. In a similar manner, if I want to convey the information that DD size can heavily depend on the variable order (place order) I can use examples of Boolean functions from the literature or I can show a P/T net in which, by changing the place order in the DD, the DD size changes significantly, which requires a tool in which it is easy to change the variable order. In our experience this approach is effective only if the tool provides the required features in a straightforward manner.

Based on the above insights we have decided to produce a *streamlined* and *enhanced* version of GreatSPN [18], called GreatTeach. It is streamlined because it presents to the user an easy access to a subset of the GreatSPN resources (intended as Petri net classes, solution techniques, facilities) and it is enhanced because it has some new visualization possibilities that have been introduced as a support for teaching. At the same time all previous solvers are still available.

The paper develops as follows: Sect. 2 describes the architectural view of the new interplay between GUI and solvers, Sect. 3 describes how GreatTeach supports the learning objectives of a course on Petri nets, and Sect. 4 concludes with a discussion of the current status of GreatTeach and of future work.

2 GreatTeach, an Architectural View

GreatSPN includes facilities for drawing and solving many classes of nets, but for teaching we typically use only four of them: P/T nets and colored "well-formed" nets (WN) [9] in verification courses; (Generalized) stochastic Petri nets - (G)SPN [1] and colored extensions of Stochastic WN (SWN) [9] in performance evaluation courses.

The Status Quo. GreatSPN does not have a strict enforcement of the net type, which is instead computed "on-the-fly" depending on the included elements. For example a P/T net becomes a GSPN if a rate is added to a transition, and a P/T net becomes a WN if a color class is added to the net declarations. This choice is coherent with the idea that qualitative (untimed) and quantitative (stochastic) analysis should work in an integrated and synergistic manner.

Figure 1[A] shows the GreatSPN GUI for a P/T net. There are two menu bars: a command bar with icons to "manage" a net and a "project" (set of nets with associated measures) together with a few commands for quick analysis; a net editing bar which is the drawing bar for the net displayed in the main canvas. Tooltips are associated with the icons to make buttons self-explanatory. The net editing bar is context sensitive: if a WN is loaded in the canvas the editing bar shows additional elements, to include drawing elements for colors (color definition, color variables, etc.). The command bar instead *is not* context sensitive: so the command bar for P/T in Fig. 1[A] includes the unfolding command (useful only for colored nets). A similar situation is present for WN: the bar includes the icons for the computation of P- and T-semiflows, for which there is no solution algorithm in GreatSPN. If this feature is selected the tool notifies that this is

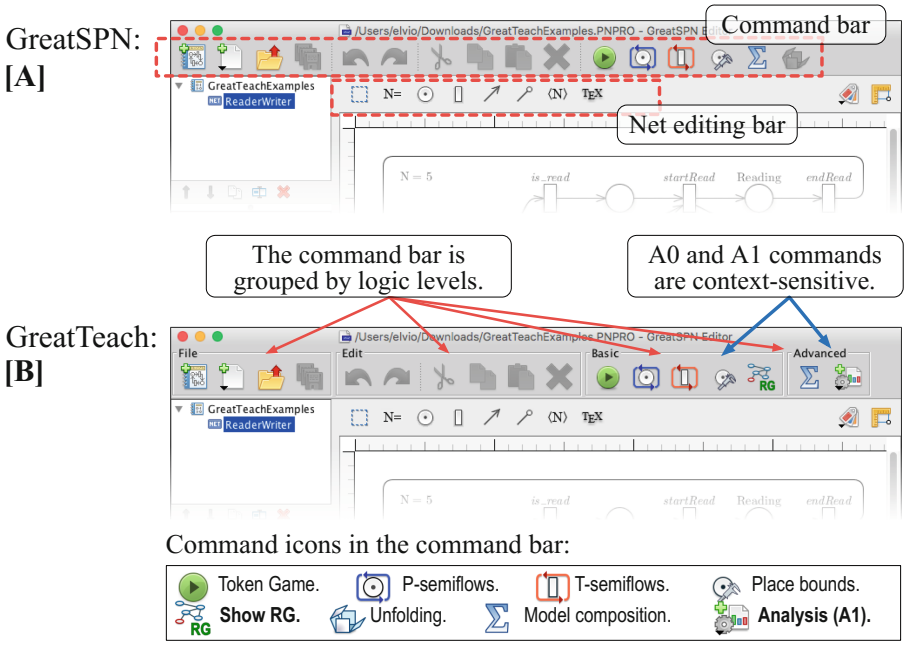


Fig. 1. The GUI for P/T nets: GreatSPN (left) and GreatTeach (right).

not possible for colored net. This is not a problem for an expert user, but it is annoying and quite distracting in lab classes.

Let us now consider how to an analysis when it is part, or when it is not part, of the command bar. P-semiflows *are* on the command bar: by pushing the button the P-semiflows are computed and graphically displayed in the main canvas over the net and textually on a lateral canvas, without any input from the user. A "close" button allows to come back to the normal net visualization. Reachability graph (RG) computation and visualization *is not* on the command bar. To run it the user has to: (1) go to the command bar and click the "add new page" button; (2) select "add a new list of measures" from the drop-down menu, which switches the GUI to the frame depicted in Fig. 2[A]; the figure depicts the remaining steps that are: (3) select the Petri net for which to perform the analysis; (4) select which target measure (in this case RG) to compute from the drop-down menu of the "Add measure" button; (5) select which solver to use; (6) set the additional parameters required by the solver (for example the type of algorithm and the maximum number of markings that we want to graphically display); (7) push the "compute" button, which opens a window with a log that allows, if necessary, to kill the solution; (8) once the computation has terminated push the "view result" button for the computed measure that, in this case, will open a pdf file with the RG visualization as a graph. Obviously 8 steps for an RG computation lead to a high cognitive load for a newcomer to the tool (imagine having to explain this in a lab class), while they could be acceptable for an expert, who may desire exactly this flexibility to try different solution algorithms (there are many ways of computing the RG in GreatSPN).

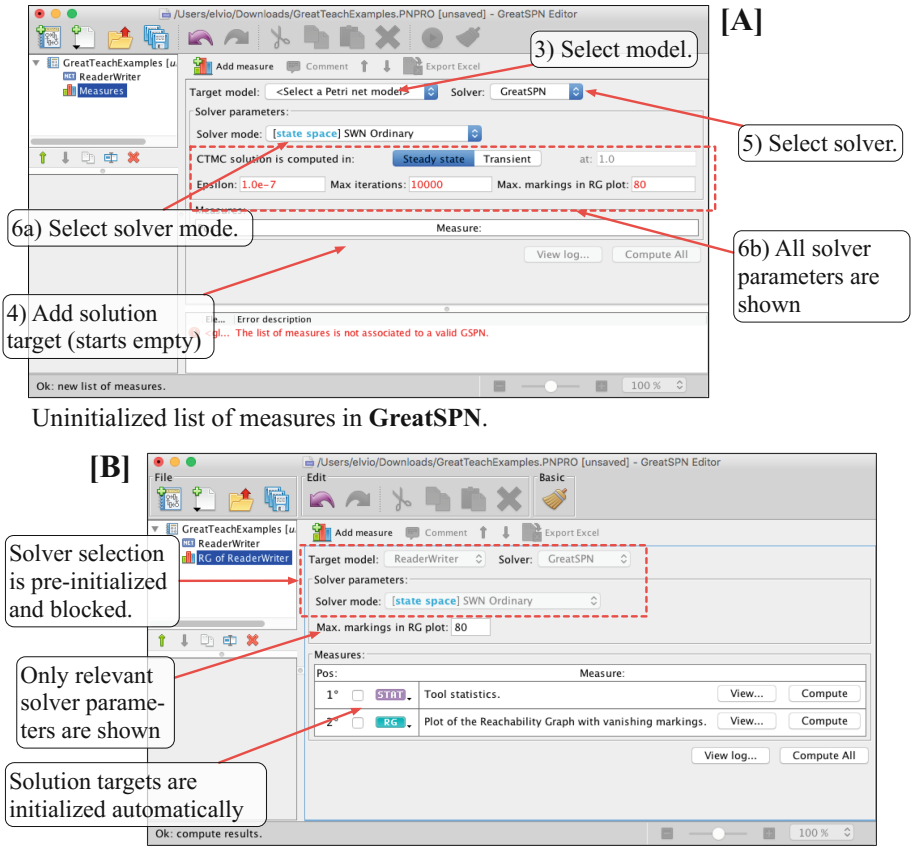


Fig. 2. Calling a solution in GreatSPN and in GreatTeach.

GreatTeach Architecture. The previous example clearly shows that GreatSPN and its GUI were designed for researchers, and that using a research tool to teach, although very desirable in our opinion, may not be effective with students. We have therefore re-structured the interplay between nets, solvers and target measures into three classes, called *architectural levels*:

A0. Commands accessible with a "push button". These are commands for which we have decided that there should be no additional interaction with the user, not even to kill the solver or to assign required parameters, that should be substituted by pre-defined, built-in, values.
A1. Commands that require an interaction with the user, but this interaction is in a simplified form; again, if a more complex interaction is needed by the solver, it should be simplified by making pre-defined choices.
A2. The full set of commands, solver and options currently available through the "list of measures" page.

The command bar has been re-organized into boxes: File, Edit, Basic, and Advanced, as in Fig. 1[B]. Commands designed according to the architectural level A0 are all placed in the "Basic" box. The ones designed at level A1 are put in the "Advanced" box, and they can be selected from a drop-down menu that is associated to the "Analysis" button (rightest button on the command bar). An example of the menu for P/T net is shown in Fig. 3[A]. Commands at level A3 are accessible through the standard "List of measure" page.

A good example for explaining the three levels is again the RG computation for which users of different expertise may require different types of computation. Indeed the first net drawn by a student has good chances to have an unbounded number of states, or too many states to make the picture of the RG meaningful, so the "push button" could be a good option but it should be carefully designed. A researcher working on very large nets may want instead to tailor all possible parameters. GreatTeach therefore has one RG computation for level. In A0 there is a "limited RG" solution, accessible through the button "showRG" (see Fig. 1[B]), that generates at most 10.000 states and displays the pdf of only the first 80 states and relative (possibly dangling) arcs. The pdf also includes standard statistics (number of states and of dead markings). This is a solver that is new to GreatTeach. At level A1, when RG computation is selected from the drop-down menu, a customized measure page is presented. This page is shown in Fig. 2[B]: the net name (target model) and the solver have been pre-selected and they are not modifiable, and the user can select a single parameter (the number of states to be drawn in the pdf of the RG). The target measures and parameters are pre-initialized. So the user can simply press the "Compute" button and, in two clicks, he/she gets with GreatTeach the same results that requires 8 steps in GreatSPN. Again, this feature has been developed specifically for GreatTeach. At level A2 the computation is as in GreatSPN: fully flexible but with 8 interactions.

Another important architectural change in GreatTeach is that also the command bar has been made context-sensitive to the type of the net currently displayed in the tool canvas. As can be observed by comparing the command bar portions of the GreatTeach GUI shown in Fig. 3 for P/T net (case A), for GSPN (case B) and for WN (case C) the available commands in the Basic and in the advanced box and drop down menu are now different. Note that the command bar implements the same full flexibility that GreatSPN has in changing the net editing bar: in GreatTeach if the canvas shows a P/T net and we add a color to a place, both the net editing bar and the command bar change accordingly to provide support for WN drawing and solution.

3 GreatTeach, a Teaching View

The choice of having three architectural levels was guided by a restructuring of our learning objectives into three levels. This classification is totally based on our experience since, as far as we know, there is no shared and accepted notion of basic or advanced knowledge for Petri nets. The idea is that the first two levels are for master courses, while the third level is for final projects. A generic description of the levels is:

L0: basic knowledge, sufficient for a student to produce and analyse simple properties on simple models, for all net classes
L1: intermediate knowledge, aimed at understanding an extended set of analysis techniques
L2: advanced knowledge, in particular to experiment and compare different solvers for the same target objectives.

In the remaining of the section we identify the learning objectives for each net class and we connect it with the organization into the three architectural levels of the solvers. For P/T we have, at a minimum:

L0: Understand and experiment with enabling and firing rule, RS and RG definition, bound of places, P- and T-invariants.
L1: Understand use of decision diagrams for RS computation, define and model-check CTL properties, build nets through net composition.
L2: Every other learning objective supported by GreatSPN, like to understand the differences between vanishing vs. tangible states, to generate plots based on parametric analysis, and to experiment and compare different solvers.

The matrix of learning objective levels vs. architectural levels of solvers should have a full diagonal: for an objective at level Li there should be *at least* one method at architectural level Ai, and this is indeed what we have now available in GreatTeach. The association of commands to a learning objective can be easily described with the support of the screenshots of the command bar included in Fig. 3. For P/T nets (Fig. 3[A]) the Basic box (level A0 commands) includes the icons for Token Game (to learn enabling and firing), place bounds computation, P- and T- semiflows computation and visualization and (limited) RS and RG

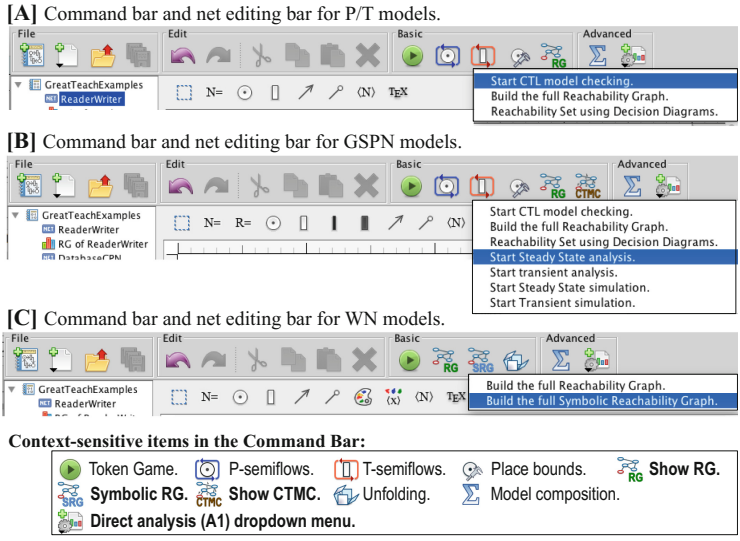


Fig. 3. Command bars in GreatTeach (Color figure online)

computation and visualization. In the legenda of the icons, we have used the boldface to emphasize the solver that have been *expressly developed or modified* for GreatTeach. The icon for the "Direct analysis (A1) drop-down menu" is in bold since *all the solvers in the drop-down menu* (level A1 solvers) have been explicitly developed or modified for GreatTeach. The Advanced box (level A1 commands) includes an icon for Model Composition (according to [7]) and a drop down menu with the three commands shown in the Figure, that meets the learning objectives in L1 and that can support a more complete analysis of the RS and RG computation (with respect to the level L0 commands present in the Basic box). Due to lack of space we cannot show an example, but the "Reachability set using Decision Diagram" builds the RS and produces a pdf of the decision diagram, a graphical feature that we have explicitly developed for GreatTeach, "Build the full RG" leads to the page shown in Fig. 2, and "Start CTL model checking" leads to a page in which the user can, with a single click, evaluate [4] a predefined CTL formula, or change it to fit their needs. The user can also specify a predefined variable order or a manual one. CTL can also be run at level A2, to experiment with different heuristics [5] for variable ordering. For GSPNs we have identified, at a minimum:

L0: as for P/T, plus the distinction between vanishing and tangible markings, the construction of the Continuous Time Markov Chain (CTMC) of the net with attention to the probabilities computed for conflict resolution.
L1: as for P/T, plus CTMC solution (transient and steady-state) based on RS or on simulation and computation and visualization of basic performance indices (token distribution in places and throughput of transitions).
L2: everything else, including model checking of the stochastic logic CSL^{TA}, definition of generic performance indices, advanced solution techniques.

Figure 3[B] is the command bar for GSPN. The solvers in the Basic box are the same as for P/T plus a CTMC computation and visualization at level A0. The visualized CTMC has the explicit indication of the rate computation, to learn the interplay between rates of exponential transitions and weights of immediate ones. The showRG icon builds a RG in which vanishing and tangible states are distinguished. The Advanced box has a drop down menu with four more options for transient/steady-state solution based on CTMC computation or simulation. These are level A1 solvers: they build a measure page that requires the minimum interaction from the user.

A similar identification of learning objectives has been done for WN and SWN. Without listing the full set of objectives, we can emphasize that the learning objectives for WN are basically the same as for P/T nets, plus the specific objectives connected to the definition and comprehension of the colors, of the colored and symbolic RG definition and of their differences (level L0). Figure 3[C] shows the command bar of WN. In the Basic box we have the Token game (colored), limited RG and limited SRG computation and the unfolding of a WN into an equivalent P/T, while the Advanced box includes Model Composition and the full construction of RG and SRG. For SWN the menu of the advanced box is enriched with the four stochastic solvers, as in the GSPN case, combined with the choice of using a CTMC built on the RG or on the SRG.

4 Conclusions and Future Work

Literature reports many studies on teaching formal methods in general and Petri nets in particular. In almost all cases, the authors agree that a course needs a mixture of theoretical concepts and practical experiments with software tools. In our teaching experience we could observe that a graphically nice interface helps, but it is not enough. The interface should be designed with teaching in mind and this is what we have done with GreatTeach. GreatTeach has some enhanced features for graphical visualization (DD and CTMC among others) and a streamlined interface: easy and/or basic learning objectives should require an easy interaction with the tool. The more the student learns, the deeper the interaction with the tool can be. There are many things we want to add as a support for teaching, among others: some RG analysis algorithms are currently not accessible, rewards for SPN should make the definition of measures significantly easier, and we need to include an easy way to introduce the students to our stochastic model-checker for CSL^{TA}. In introducing these new features we shall first define the level L_i of the learning objective, and then design the solver according to the chosen architectural level. This approach, we believe, is a contribution of the paper that goes beyond the specific tool considered.

The learning objectives of the previous section, and therefore the choice of the properties that are analyzable at level A0 and A1 are strongly dependent on the courses that we teach at the University of Torino. Different courses may need a different characterization. The newest release of GreatSPN is available at github.com/greatspn/SOURCES: having the source code available on Github should allow other research group to develop their own customization based on a specific choice of learning objective. Distribution to students of a frozen version is also important when working in lab classes, to ensure that all students work with exactly the same tool. A Virtualbox image of GreatTeach is available at http://www.di.unito.it/~greatspn/VBox/GreatTeach.ova.

The attentive reader may have noticed that the name of the editor on top of the GUI in Figs. 1 and 2 is GreatSPN on both sides. This is not an error: the architectural choices of GreatTeach led to a GUI that is more structured and has a number of additional features and solvers specifically developed for it. We believe that GreatTeach can work well also for expert researchers, therefore we have decided to use GreatTeach as the next release of GreatSPN, which also avoids the creation of a branch in the development process. In the paper we retain the two separate names for clarity.

References

1. Ajmone-Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley, Hoboken (1995)
2. Amparore, E.G.: Reengineering the editor of the GreatSPN framework. In: PNSE@ Petri Nets, pp. 153–170 (2015)
3. Amparore, E.G., Balbo, G., Beccuti, M., Donatelli, S., Franceschinis, G.: 30 years of GreatSPN. In: Fiondella, L., Puliafito, A. (eds.) Principles of Performance and Reliability Modeling and Evaluation. SSRE, pp. 227–254. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30599-8_9

4. Amparore, E.G., Beccuti, M., Donatelli, S.: (Stochastic) model checking in GreatSPN. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 354–363. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07734-5_19
5. Amparore, E.G., Beccuti, M., Donatelli, S.: Gradient-based variable ordering of decision diagrams for systems with structural units. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 184–200. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_13
6. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30080-9_7
7. Bernardi, S., Donatelli, S., Horváth, A.: Implementing compositionality for stochastic Petri nets. Softw. Tools Technol. Transf. J. **3**(4), 417–430 (2001)
8. Cerone, A., Roggenbach, M., Schlingloff, H., Schneider, G., Shaikh, S.: Teaching formal methods for software engineering - ten principles. In: Informatica Didactica. University of Potsdam, Germany (2015)
9. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: Stochastic well-formed colored nets and symmetric modeling applications. IEEE Trans. Comput. **42**(11), 1343–1360 (1993)
10. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new symbolic model verifier. In: Halbwachs, N., Peled, D. (eds.) CAV 1999. LNCS, vol. 1633, pp. 495–499. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48683-6_44
11. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy – a unifying Petri net tool. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 398–407. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31131-4_22
12. Heiner, M., Schwarick, M., Wegener, J.-T.: Charlie – an extensible Petri net analysis tool. In: Devillers, R., Valmari, A. (eds.) PETRI NETS 2015. LNCS, vol. 9115, pp. 200–211. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19488-2_10
13. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer, Heidelberg (2009). https://doi.org/10.1007/b95112
14. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri nets and CPN tools for modelling and validation of concurrent systems. Softw. Tools Technol. Trans. J. **9**(3), 213–254 (2007)
15. Mei, C., Zhang, X., Zhao, W., Periyasamy, K., Headington, M.: A tool for teaching Petri nets. J. Comput. Sci. Coll. **26**(5), 181–188 (2011)
16. Schwarick, M., Heiner, M., Rohr, C.: MARCIE - model checking and reachability analysis done efficiently. In: International Conference on Quantitative Evaluation of Systems, pp. 91–100 (2011)
17. Thong, W.J., Ameedeen, M.A.: A survey of Petri net tools. In: Sulaiman, H.A., Othman, M.A., Othman, M.F.I., Rahim, Y.A., Pee, N.C. (eds.) Advanced Computer and Communication Engineering Technology. LNEE, vol. 315, pp. 537–551. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-07674-4_51
18. University of Torino: the GreatSPN tool homepage. http://www.di.unito.it/~greatspn/index.html
19. Vörös, A., Darvas, D., Molnár, V., Klenik, A., Hajdu, Á., Jámbor, A., Bartha, T., Majzik, I.: PetriDotNet 1.5: extensible Petri net editor and analyser for education and research. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016. LNCS, vol. 9698, pp. 123–132. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39086-4_9
20. Zimmermann, A.: Modelling and performance evaluation with TimeNET 4.4. In: Bertrand, N., Bortolussi, L. (eds.) QEST 2017. LNCS, vol. 10503, pp. 300–303. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66335-7_19

Author Index