Acta Informatica (2018) 55:575–611
https://doi.org/10.1007/s00236-017-0310-9

ORIGINAL ARTICLE

Bounded choice-free Petri net synthesis: algorithmic issues

Eike Best[1] · **Raymond Devillers**[2] · **Uli Schlachter**[1]

Received: 13 February 2017 / Accepted: 9 November 2017 / Published online: 20 November 2017

Abstract This paper describes a synthesis procedure dedicated to the construction of choice-free Petri nets from finite persistent transition systems, whenever possible. Taking advantage of the properties of choice-free Petri nets, a two-step approach is proposed. A pre-synthesis step checks necessary structural properties of the transition system and constructs some data structures needed for the second step. Then, a minimised set of simplified systems of linear inequalities is distilled from a general region-theoretic approach. This leads to a substantial narrowing of the sets of states for which linear inequalities must be solved, and allows an early detection of failures, supported by constructive error messages. The performance of the resulting algorithm is measured and compared numerically with existing synthesis tools.

Contents

Supported by DFG (German Research Foundation) through Grant Be 1267/15-1 ARS (Algorithms for Reengineering and Synthesis).

✉ Uli Schlachter
uli.schlachter@informatik.uni-oldenburg.de

Eike Best
eike.best@informatik.uni-oldenburg.de

Raymond Devillers
rdevil@ulb.ac.be

[1] Department of Computing Science, Carl von Ossietzky Universität, 26111 Oldenburg, Germany

[2] Université Libre de Bruxelles, Boulevard du Triomphe, C.P. 212, 1050 Bruxelles, Belgium

1 Introduction

1.1 Context of the paper

In synthesis, the task is to construct—preferably automatically, when possible—an implementing system from a given behavioural specification. A successfully synthesised system is correct by design. There is no need to re-examine its behavioural properties, because they are known to hold by construction. If synthesis is impossible for an input specification, then it is desirable to delineate—when possible—the reasons of the failure. This paves the way for adjustments of the given behaviour, allowing for a more successful subsequent synthesis.

In this paper, we consider the synthesis of Petri nets from labelled transition systems, as pioneered by Ehrenfeucht and Rozenberg [29] and comprehensively presented by Badouel, Bernardinello and Darondeau in [1]. In this approach, a system is presumed to be specified by a labelled transition system TS, and an implementation of it is sought in the shape of an unlabelled Petri net N with a reachability graph isomorphic to TS. The basic synthesis algorithm described in [1] works by associating a large number of linear-algebraic inequality systems to any given transition system. If all of them are solvable, a Petri net is then constructed from their solutions.

Petri net synthesis is interesting for several reasons. For instance, it is used in a variety of application areas, such as data mining [26] and digital design [22,24,34]. In the latter context, it is particularly desirable to synthesise choice-free Petri nets.[1] Moreover, a Petri net implementation tends to be much smaller than the transition system it originates from. It can also display useful information about the concurrency and the distributability inherent in a

[1] Where a net will be called choice-free if every place has at most one outgoing transition (not to be confused with free-choice nets [27]).

system [5,37]. As it happens, choice-free Petri nets are precisely the class of nets allowing an arbitrarily distributed implementation [7].

1.2 Goal of the paper

We investigate the case that Petri net synthesis is targeted towards the class of choice-free Petri nets. A naive (brute-force) approach, extending the classic algorithm described in [1], would be to add a separate layer of linear constraints describing—as much as possible—the class in question. However, such an approach is likely to create heavier and slower procedures. We shall argue that, on the contrary, very efficient synthesis procedures can be obtained by exploiting the structural properties of choice-free nets in a sophisticated way, aiming to reduce substantially (rather than to enlarge) the size and number of inequality systems that need to be solved. As sketched in Fig. 1, we therefore propose to prepend a pre-synthesis stage in front of the proper synthesis stage. Pre-synthesis serves to check some structural properties which are known (from Petri net structure theory) to hold for choice-free nets [6,35,40].

Any transition system rejected during pre-synthesis can obviously not be synthesised into a choice-free net. Since a characterisation of choice-free Petri net state spaces does not exist, it is too optimistic to assume that this will filter out *exactly* all unsolvable inputs. Instead, a surviving transition system may already, or may still not, be solvable by a choice-free net. However, knowing that our inputs now satisfy a set of restrictive properties, we can expect (and will indeed show) that fewer inequalities need to be solved than if the inputs were not restricted, making the resulting procedure competitive in terms of efficiency. Apart from this, pre-synthesis has additional benefits: first, in many cases, inputs for which synthesis is not possible may be detected at an early stage, leading to run-time savings in failing cases; secondly, meaningful error messages may then be issued; thirdly, data structures facilitating the subsequent synthesis may already be built during pre-synthesis.

1.3 Structure of the paper

After the basic notations used in this paper, and some examples, in Sect. 2, we turn our attention to the two-stage method proposed in Fig. 1. It involves at least three intricate problems:

- Choosing the properties to be checked during pre-synthesis.
  A proposal will be made in Sect. 3 below.
- Checking the chosen properties optimally.

input a labelled transition system TS;
Stage 1 (pre-synthesis):
for every property Π of (the state spaces of) choice-free nets **do**
  if TS does not satisfy Π **then** synthesis failure due to $\neg\Pi$; **stop**
At this point, TS is known to satisfy several properties; consequently, some linear inequality systems needed in the basic algorithm [1] can be eliminated or reduced.
Stage 2 (synthesis):
for every remaining linear inequality system needed in the basic algorithm **do**
  if unsolvable **then** synthesis failure due to an unsolvable linear system; **stop**
output a choice-free Petri net synthesised from TS

Fig. 1 Splitting synthesis into two separate stages

This problem has been addressed in [15], the results of which will be recapitulated (without proofs) in Sect. 4 below. In Sect. 5, a pre-synthesis algorithm based on these results will then be described and proved correct.

- Reducing the number of linear inequalities to be solved during synthesis.
  This may involve judicious (mathematical) reasoning, as spelt out in [13]. Section 6 describes a synthesis algorithm which is based on this reasoning, as well as its proof of correctness.

In Sect. 7, we sketch various special cases and algorithmic characteristics of the combined algorithm. Experimental results are presented and analysed in Sects. 8, and 9 concludes. Some of the more technical proofs can be found in Sects. A and B of "Appendix" section.

2 Basic definitions

Definition 1 (*Labelled transition systems*) A *labelled transition system with initial state*, lts for short, is a quadruple $TS = (S, T, \rightarrow, \iota)$ where S is a set of *states*, T is a set of *labels*, $\rightarrow \subseteq (S \times T \times S)$ is the *transition relation*, and $\iota \in S$ is an *initial state*. TS is *finite* if S and T (hence also $\rightarrow$) are finite sets.

A label t is *enabled* at $s \in S$, written formally as $s[t\rangle$, if $\exists s' \in S\colon (s, t, s') \in \rightarrow$, and *backward enabled* at s, written as $[t\rangle s$, if $\exists s' \in S\colon (s', t, s) \in \rightarrow$. For $t \in T$, $s[t\rangle s'$ if $(s, t, s') \in \rightarrow$, meaning that s' is *reachable* from s through the execution of t. For *sequences* $\sigma \in T^*$, $s[\varepsilon\rangle$ and $s[\varepsilon\rangle s$ are always true; and $s[\sigma t\rangle$ ($s[\sigma t\rangle s'$) if there is some s'' with $s[\sigma\rangle s''$ and $s''[t\rangle$ ($s''[t\rangle s'$, respectively). A state s' is *reachable* from state s if $\exists \sigma \in T^*\colon s[\sigma\rangle s'$. The set of states reachable from s is denoted by $[s\rangle$. A sequence $s[\sigma\rangle s'$ is called a *cycle*, or more precisely a *cycle at (or around) state* s, if $s = s'$.

The *language* of TS is the set $L(TS) = \{\sigma \in T^* \mid \iota[\sigma\rangle\}$. Two lts with the same label set $TS = (S, T, \rightarrow, \iota)$ and $TS' = (S', T, \rightarrow', \iota')$ are *language-equivalent* if $L(TS) = L(TS')$, and *isomorphic* if there is a bijection $\zeta\colon S \rightarrow S'$ with $\zeta(\iota) = \iota'$ and $(r, t, s) \in \rightarrow \Leftrightarrow (\zeta(r), t, \zeta(s)) \in \rightarrow'$, for all $r, s \in S$ and $t \in T$. □

Definition 2 (*Petri nets*) A (finite, initially marked, place-transition, arc-weighted) *Petri net* is a tuple $N = (P, T, F, M_0)$ such that P is a finite set of *places*, T is a finite set of *transitions*, with $P \cap T = \emptyset$, F is a *flow* function $F\colon ((P \times T) \cup (T \times P)) \rightarrow \mathbb{N}$, M_0 is the *initial marking* (where a *marking* is a mapping $M\colon P \rightarrow \mathbb{N}$, indicating the number of tokens in each place).

N is *plain* if no arc weight exceeds 1; *pure* if $\forall p \in P\colon (p^\bullet \cap {}^\bullet p) = \emptyset$, where $p^\bullet = \{t \in T \mid F(p, t) > 0\}$ and ${}^\bullet p = \{t \in T \mid F(t, p) > 0\}$; *choice-free* [25,40] or place-output-nonbranching [10] if $\forall p \in P\colon |p^\bullet| \leq 1$; a *marked graph* [20] if N is plain and $|p^\bullet| = 1$ and $|{}^\bullet p| = 1$ for all places $p \in P$; and a *T-system* [27] if N is plain and $|p^\bullet| \leq 1$ and $|{}^\bullet p| \leq 1$ for all places $p \in P$.

A transition $t \in T$ is *enabled by* a marking M, denoted by $M[t\rangle$, if for all places $p \in P$, $M(p) \geq F(p, t)$. If t is enabled at M, then t can *occur* (or *fire*) in M, leading to the marking M' defined by $M'(p) = M(p) - F(p, t) + F(t, p)$ (denoted by $M[t\rangle M'$). A marking M' is *reachable* from M if there is a sequence of firings leading from M to M'. The set of markings reachable from M is denoted by $[M\rangle$. The *reachability graph of* N is the labelled transition system $RG(N)$ with the set of vertices $[M_0\rangle$, initial state M_0 and transitions $\{(M, t, M') \mid M, M' \in [M_0\rangle \wedge M[t\rangle M'\}$. A Petri net $N = (P, T, F, M_0)$ is *bounded* if $[M_0\rangle$ is finite, i.e., the number of tokens in each place is bounded. Moreover, all notions defined for labelled transition systems apply to Petri nets through their reachability graphs. □

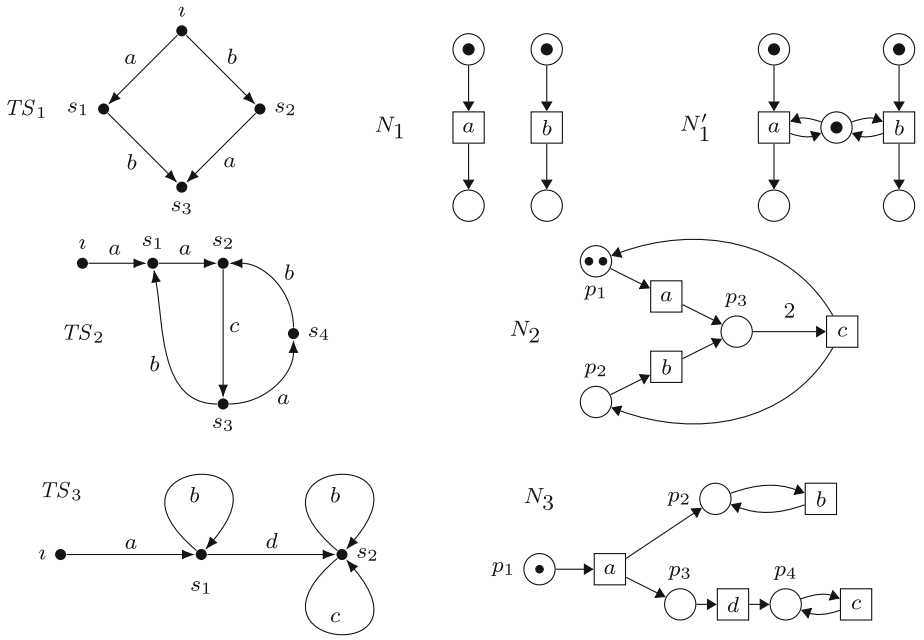

Fig. 2 N_1 solves TS_1 choice-freely. N_1' also solves TS_1 (but not choice-freely). N_2 solves TS_2 choice-freely, and N_3 solves TS_3 choice-freely. No plain solution of TS_2 exists, and no pure solution of TS_3 exists (we shall not prove this in the present paper. It is not difficult to verify)

The next definition connects the previous two. It delineates the class of transition systems which are (isomorphic to) Petri net reachability graphs.

Definition 3 (*Solvable transition systems*) If an lts TS is isomorphic to the reachability graph of a Petri net N, we say that N *solves* TS or that TS is *synthesisable* into N. □

The examples shown in Fig. 2 illustrate various basic properties of the structures we consider: solvability; plainness; and pureness. Note that we only consider unlabelled Petri nets, so that we do not have silent transitions; nor two different transitions with the same label; nor transition carrying a sequence of labels. This requirement is a basic assumption in [1] as well as in this paper.[2] It supports the precise analysis of physical distributability [5,7,31,41]. For instance, the transition system TS_1 may be synthesised into N_1 or into N_1', also shown in this figure (or into other nets, not shown here). N_1 is choice-free, and a and b are independent of each other so that they can be executed concurrently (and also be distributed physically) in order to generate TS_1. By contrast, N_1' is not choice-free, and its structure does not reveal any explicit concurrency or distributivity information. TS_2 can be solved by the Petri net N_2; the latter is not plain, since it contains an arc having weight greater than 1 from p_3 to c. Similarly, TS_3 can be solved by N_3, which is not pure, because it contains (twice) a transition and a place which are connected in both directions by arcs.

Definition 4 (*Parikh vectors*) Let TS be a labelled transition system $TS = (S, T, \rightarrow, \iota)$. A T*-vector* is a function $\Phi : T \rightarrow \mathbb{N}$, and its *support* is $supp(\Phi) = \{t \in T \mid \Phi(t) > 0\}$. For a T-vector $\Phi : T \rightarrow \mathbb{N}$, the *greatest common divisor* $\gcd(\Phi)$ is defined as $\gcd(\Phi) =$

[2] Which could be relaxed in different contexts [22] tolerating label splitting [17].

$\gcd\{\Phi(t) \mid t \in T\}$. Two T-vectors $\Phi_1, \Phi_2 : T \to \mathbb{N}$ are *label-disjoint* if their supports are disjoint. The *Parikh vector* of a sequence $\sigma \in T^*$, denoted $\Psi(\sigma)$, is the T-vector counting for each label $t \in T$ the number of occurrences of t in σ. The support of a sequence σ is defined as $supp(\sigma) = supp(\Psi(\sigma))$, the greatest common divisor as $\gcd(\sigma) = \gcd(\Psi(\sigma))$. Two sequences $\sigma_1, \sigma_2 \in T^*$ are *label-disjoint* if $\Psi(\sigma_1)$ are $\Psi(\sigma_2)$ are label-disjoint, and *Parikh-equivalent* if they have the same Parikh vector, i.e., if $\Psi(\sigma_1) = \Psi(\sigma_2)$. □

3 Necessary properties of bounded choice-free Petri nets

We first define some properties which an lts must satisfy in order to be solvable by a *bounded* Petri net (Definitions 5, 6 and Theorem 1). Then, a set of properties which an lts must additionally satisfy in order to be solvable by a bounded *choice-free* Petri net will be described (Definitions 7–9 and Theorem 2). The section ends with some remarks about checking these properties.

Definition 5 (*Total reachability, weak and full determinism*) A labelled transition system $(S, T, \to, \iota)$ is called

- *totally reachable* if $[\iota\rangle = S$ (i.e., every state is reachable from ι);
- *(weakly forward) deterministic* if, for all states $s, s', s'' \in S$, and for any label $t \in T$, $s[t\rangle s'$ and $s[t\rangle s''$ imply $s' = s''$ (i.e., an executable label uniquely determines the successor state);
- *(weakly) backward deterministic* if, for all states $s, s', s'' \in S$, and for any label $t \in T$, $s'[t\rangle s$ and $s''[t\rangle s$ imply $s' = s''$;
- *fully forward deterministic* if, for all states $s, s', s'' \in S$ and for all sequences $\alpha, \alpha' \in T^*$, $(s[\alpha\rangle s' \wedge s[\alpha'\rangle s'' \wedge \Psi(\alpha) = \Psi(\alpha'))$ entails $s' = s''$ (i.e., the Parikh vector of an executable sequence uniquely determines the target state);
- *fully backward deterministic* if, for all states $s, s', s'' \in S$ and sequences $\alpha, \alpha' \in T^*$, $(s'[\alpha\rangle s \wedge s''[\alpha'\rangle s \wedge \Psi(\alpha) = \Psi(\alpha'))$ entails $s' = s''$. □

Clearly, full forward (backward) determinism implies weak forward (backward, respectively) determinism, but not conversely.

Definition 6 (*Weak periodicity*) A labelled transition system $(S, T, \to, \iota)$ is called *weakly periodic* if for every $s_1 \in S$, label sequence $\sigma \in T^*$, and infinite sequence $s_1[\sigma\rangle s_2[\sigma\rangle s_3[\sigma\rangle s_4[\sigma\rangle \cdots$, either $s_i = s_j$ for all $i, j \geq 1$, or $s_i \neq s_j$ for all $i, j \geq 1$ with $i \neq j$. □

In a finite and weakly periodic lts, the second condition ($s_i \neq s_j$) in the definition cannot hold. Thus, a finite lts $(S, T, \to, \iota)$ is weakly periodic iff, for every infinite sequence $s_1[\sigma\rangle s_2[\sigma\rangle s_3[\sigma\rangle \cdots$, $s_i = s_j$ for all $i, j \geq 1$.

For example, in Fig. 3, TS_4 is fully deterministic in both directions, but not weakly periodic (for instance, $\iota[aaa \ldots\rangle$, and not all visited states are equal or different). By contrast, TS_5 is not just fully deterministic but also weakly periodic (for instance, from ι, the only possible periods are $\sigma = aabbcc$ or its multiples, and they always lead back to ι).

Theorem 1 (Petri net reachability graphs) *The reachability graph $RG(N)$ of a Petri net N is totally reachable, fully forward and backward deterministic, and weakly periodic. Moreover, it is finite iff N is bounded.*

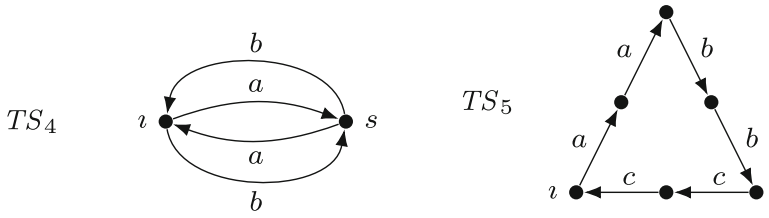

Fig. 3 TS_4 is fully deterministic, but not weakly periodic. TS_5 is fully deterministic as well as weakly periodic

Proof Easy, as well as basic, in Petri net theory. Total reachability arises by the definition of $RG(N)$. All forms of determinism, as well as weak periodicity, follow directly from the Petri net state equation [36,38]. Finiteness is immediate from the definition of Petri net boundedness. □

Definition 7 (*Persistence*) A labelled transition system $TS = (S, T, \rightarrow, \iota)$ is called *persistent* [35] if for all states $s, s', s'' \in S$, and labels $t \neq u$, if $s[t\rangle s'$ and $s[u\rangle s''$, then there is some state $r \in S$ such that both $s'[u\rangle r$ and $s''[t\rangle r$ (i.e., once two different labels are both enabled, neither can disable the other, and this leads to the same state, forming a characteristic diamond shape). □

Definition 8 (*Small and prime cycles*)
Let TS be a labelled transition system $TS = (S, T, \rightarrow, \iota)$.

- A cycle $s[\sigma\rangle s$ is *nontrivial* if $\sigma \neq \varepsilon$.
- A cycle $s[\sigma\rangle s$ is *small* if it is nontrivial and there is no nontrivial cycle $s'[\sigma'\rangle s'$ with $s' \in S$ and $\Psi(\sigma') \lneqq \Psi(\sigma)$ (where, by definition, $\lneqq$ equals $(\leq \cap \neq)$, and $\leq$ is understood componentwise).
- A cycle $s[\sigma\rangle s$ is *prime* if $\gcd(\sigma) = 1$.

TS has the *prime cycle property* if all of its small cycles are prime. □

For example, TS_1, TS_2 and TS_3 (cf. Fig. 2) are persistent and satisfy the prime cycle property. TS_4 and TS_5 (Fig. 3) are persistent but do not satisfy the prime cycle property. In TS_4, there are non-prime small cycles $\iota[aa\rangle\iota$ and $\iota[bb\rangle\iota$ around state ι. TS_5 contains a non-prime small cycle $\iota[aabbcc\rangle\iota$.[3]

Definition 9 (*Short paths, Parikh-minimal paths*, cycle-reduction) Let TS be a labelled transition system $TS = (S, T, \rightarrow, \iota)$.
A path $r[\sigma\rangle s$ (for $r, s \in S$ and $\sigma \in T^*$) is

- *short* (or *short from r to s*, to be precise), if no path $r[\sigma'\rangle s$ satisfies $|\sigma'| < |\sigma|$;
- *Parikh-minimal* if no path $r[\sigma'\rangle s$ satisfies $\Psi(\sigma') \lneqq \Psi(\sigma)$;
- *cycle-reduced* if there is no $s' \in S$ and small cycle $s'[\alpha\rangle s'$ with $\Psi(\alpha) \leq \Psi(\sigma)$.

TS has the *cycle-reduction property* if for any pair $r, s \in S$ of states with $s \in [r\rangle$, there is a cycle-reduced path $r[\sigma\rangle s$. □

[3] We stress that small cycles are different from elementary cycles in graph theory, i.e., cycles which visit every state at most once. While every small cycle is also elementary, the converse is not true. In fact, there exist finite, totally reachable, deterministic, persistent, transition systems in which every small cycle has a prime Parikh vector, and which contain non-prime elementary cycles.

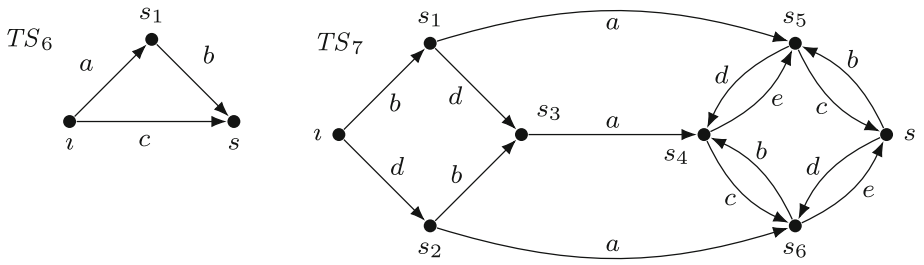

Fig. 4 TS_6 contains a Parikh-minimal but non-short path $\iota[ab\rangle s$. TS_7 contains small cycles with Parikh vectors (0 1 1 0 0) and (0 0 0 1 1) (for the label ordering $(a\ b\ c\ d\ e)$)

Clearly, every short path is Parikh-minimal. TS_6, depicted on the left-hand side of Fig. 4, shows that the converse implication does not hold in general. If $r \in S$ and $s \in [r\rangle$, there is always a short path (hence also a Parikh-minimal one) from r to s; however, it may happen that there is no cycle-reduced path from r to s. For instance, in TS_7, depicted on the right-hand side of Fig. 4, the paths $\iota[bac\rangle s$ and $\iota[dae\rangle s$ are both short and Parikh-minimal. However, they are not cycle-reduced, since, for instance, there is a small cycle $s[bc\rangle s$ whose Parikh vector is dominated by the Parikh vector of the path $\iota[bac\rangle s$, i.e., satisfies $\Psi(bc) \leq \Psi(bac)$. Similarly, there is a small cycle $s[de\rangle s$ with $\Psi(de) \leq \Psi(dae)$. Moreover, there is no cycle-reduced path from ι to s, so that TS_7 does not have the cycle-reduction property.

Theorem 2 (Bounded choice-free Petri net reachability graphs) *The reachability graph of a choice-free Petri net is persistent. If the net is bounded, it also satisfies the prime cycle property and the cycle-reduction property.*

Proof Choice-free nets are readily seen to be structurally persistent, i.e., persistent for any initial marking. For a proof of the prime cycle property, see Lemma 3.6 of [13]. The cycle-reduction property arises from Lemma 16 in [40], which implies that if $s[\sigma\rangle s'$ with $\Psi(\sigma) \geq \Psi(\alpha)$ and $s''[\alpha\rangle s''$, then σ may be rearranged in such a way that $s[\sigma'\rangle s'[\sigma''\rangle s'$ with $\Psi(\sigma) = \Psi(\sigma') + \Psi(\sigma'')$ and $\Psi(\sigma'') = \Psi(\alpha)$. □

The scheme shown in Fig. 1 stipulates that a pre-synthesis phase for bounded and choice-free Petri nets should ideally check all properties specified in Theorems 1 and 2. TS_5 is rejected by such a test because it does not enjoy the prime cycle property, and TS_7 is eliminated because it does not satisfy the cycle-reduction property. In TS_4, choice-free synthesis fails for at least two reasons: it is not weakly periodic; and it has a non-prime cycle.

Checking the properties ought to be organised in an efficient manner. For example, total reachability is easy to check, by visiting each edge once. Checking weak determinism is also easy, since it is a "local" property, referring only to the immediate surroundings of states. For persistence, each pair of enabled labels has to be checked in every state, which is quadratic in the number of labels; but we may expect the number of labels to be a lot smaller than the number of states, so that we could get an approximately linear complexity in the size of the lts. By contrast, checking weak periodicity or full determinism is more expensive, since these are "global" properties pertaining to pairs (or triples) of states, and (possibly long) paths between them. It is therefore desirable that a bunch of local (or at least, not too difficult to check) properties implies one or more global ones mathematically, because checking the latter can then be skipped if the former are true. For a start, we have the following result, whose proof can be found in Appendix A.

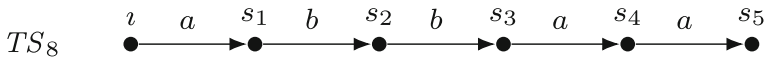

Fig. 5 An lts which satisfies all properties defined so far but is not Petri net solvable

Proposition 1 (Deriving full forward determinism) *Let $TS = (S, T, \rightarrow, \iota)$ be a deterministic, persistent lts. Then TS is fully forward deterministic.* □

Theorems 1 and 2 cannot be reversed, as shown by the example in Fig. 5. Indeed, TS_8 survives all of the above pre-synthesis tests, since all properties are satisfied. It will not survive proper synthesis, however; for its non-solvability, the reader is referred to a more general analysis of 2-label lts (e.g., [8]).

4 Structure of cycles and paths in bounded choice-free nets

Throughout this section, we shall concentrate our investigation on labelled transition systems which are

finite, totally reachable, deterministic, persistent (1)

Finiteness is usually guaranteed by the way the lts is given. Total reachability and determinism are mandatory for Petri net solvability in general, and persistence is mandatory for choice-free solvability. In Sects. 4.1 and 4.2, we continue to investigate the structure of the cycles and paths of bounded choice-free Petri net reachability graphs, obtaining further necessary conditions.

4.1 The cyclic structure of bounded choice-free Petri nets

Observe that the transition system TS_4 shown above has four small cycles, $\iota[aa\rangle\iota$, $\iota[ab\rangle\iota$, $\iota[ba\rangle\iota$, and $\iota[bb\rangle\iota$, whose Parikh vectors partially overlap. The next definition forbids this.

Definition 10 (*Disjoint small cycles property* $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$) A labelled transition system $TS = (S, T, \rightarrow, \iota)$ has the *disjoint small cycles property* if there exist an integer $n \leq |T|$ and a finite set of mutually label-disjoint T-vectors $\Upsilon_1, \ldots, \Upsilon_n : T \rightarrow \mathbb{N}$ such that

$$\{\, \Psi(\beta) \mid TS \text{ contains a state } s \text{ and a small cycle } s[\beta\rangle s \,\} = \{\, \Upsilon_1, \ldots, \Upsilon_n \,\}$$

If this property is satisfied, we shall abbreviate it by $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ (for disjoint **P**arikh vectors of small cycles). □

In particular, this implies that the Parikh vectors of small cycles are either equal or disjoint. For example, in Fig. 4, TS_7 satisfies $\mathbf{P}\{\Psi(bc), \Psi(de)\}$.

It is shown in [6] that the reachability graph of every bounded choice-free Petri net satisfies the disjoint small cycles property. However, not every lts satisfying (1) does so; for instance, TS_4 does not. Nevertheless, $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ is implied with a single additional premise, as shown by the next two theorems.

Theorem 3 (Obtaining $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ using weak periodicity [6,15]) *Let $TS = (S, T, \rightarrow, \iota)$ be finite, totally reachable, deterministic, and persistent. Also, assume that TS is weakly periodic. Then TS satisfies $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ for some set $\{\Upsilon_1, \ldots, \Upsilon_n\}$.* □

Theorem 4 (Obtaining $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ using prime cycles [15]) *Let $TS = (S, T, \rightarrow, \iota)$ be finite, totally reachable, deterministic, and persistent. Also, assume that TS satisfies the prime cycle property. Then TS satisfies $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ for some set $\{\Upsilon_1, \ldots, \Upsilon_n\}$.*

We state a corollary, which can be proved using home states.

Definition 11 (*Home states*) Let $TS = (S, T, \rightarrow, \iota)$ be a labelled transition system. A state $\widetilde{s} \in S$ is a *home state* if $\forall s \in S : \widetilde{s} \in [s\rangle$ (i.e., $\widetilde{s}$ is reachable from any state). □

It is not hard to prove [6] that in a finite, totally reachable, deterministic, and persistent lts, home states always exist.

Lemma 1 (Parikh-equivalent cycle transportation along a path) *Let $TS = (S, T, \rightarrow, \iota)$ be a deterministic and persistent lts. Suppose that $r[\kappa\rangle r$ and $r[\alpha\rangle s$. Then there is a sequence $s[\kappa'\rangle s$ with $\Psi(\kappa') = \Psi(\kappa)$.* □

By Lemma 1 (the proof of which can be found in Appendix A), every cycle has a Parikh-equivalent counterpart around every home state. Using this, it is also easy to prove that, once $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ is established, the Parikh vector of any cycle is a linear combination of the Υ_i's:

Corollary 1 (Cycle decomposition (Theorem 2 in [6])) *Let $TS = (S, T, \rightarrow, \iota)$ be finite, totally reachable, deterministic and persistent. Suppose it also satisfies $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$. Then the Parikh vector of any cycle $s[\rho\rangle s$ satisfies $\Psi(\rho) = \sum_{i=1}^{n} k_i \cdot \Psi(\Upsilon_i)$ with some (unique) $k_i \in \mathbb{N}$.* □

Due to Theorems 3 and 4, the last premise can also be replaced by weak periodicity or by the prime cycle property. By Lemma 1, the decomposition given by Corollary 1 can be realised by the Parikh vectors of small cycles around any home state s.

4.2 The path structure of bounded choice-free Petri nets

Even if a labelled transition system satisfies (1) and enjoys, in addition, weak periodicity, the prime cycle property, and the disjoint small cycles property, its path structure may still prevent it from being choice-freely Petri net synthesisable. This is the case for TS_7 (shown in Fig. 4) where the missing cycle-reduction property is instrumental in preventing choice-free solvability. We shall base our investigations on the notions of *modulo vectors* and *distances between states* (Definition 13 below).

Definition 12 (*Modulo vectors*) Suppose that $TS = (S, T, \rightarrow, \iota)$ satisfies (1) and $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$. For a T-vector $\Psi : T \rightarrow \mathbb{N}$, its modulo vector $\Psi \bmod \{\Upsilon_1, \ldots, \Upsilon_n\}$ is the smallest, non-negative vector $\Psi - \sum_{i \in \{1,\ldots,n\}} k_i \cdot \Upsilon_i$ for $k_1, \ldots, k_n \in \mathbb{N}$. Since the Υ_i's are mutually label-disjoint, this minimal vector is well-defined.

For example, TS_7 satisfies $\mathbf{P}\{\Upsilon_1, \Upsilon_2\}$ where $\Upsilon_1 = \Psi(bc)$ and $\Upsilon_2 = \Psi(de)$. For the two paths $\iota[bac\rangle s$ and $\iota[dae\rangle s$, we get

$$\Psi(bac) \bmod \{\Upsilon_1, \Upsilon_2\} = \Psi(a) \quad \text{as well as} \quad \Psi(dae) \bmod \{\Upsilon_1, \Upsilon_2\} = \Psi(a)$$

subtracting Υ_1 in the first case and Υ_2 in the second case.

For Parikh vectors of paths between two given states, the modulo vector is an invariant, as shown in Proposition 18 of [15]:

Proposition 2 (Modulo vectors are independent of paths [15]) *Let $TS = (S, T, \to, \imath)$ be finite, totally reachable, deterministic, and persistent. Suppose it also satisfies* $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$.

Let $r[\sigma\rangle s$ and $r[\sigma'\rangle s$ be two paths between the same states $r, s \in S$.

Then $\Psi(\sigma)$ mod $\{\Upsilon_1, \ldots, \Upsilon_n\} = \Psi(\sigma')$ mod $\{\Upsilon_1, \ldots, \Upsilon_n\}$. □

Proposition 2 ensures that the next definition is sound.

Definition 13 (*Distances*) Let $TS = (S, T, \to, \imath)$ be finite, totally reachable, deterministic, and persistent. Suppose it also satisfies $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$.

Let $r, s \in S$, and let $r[\alpha\rangle s$ be a path of TS. Then $\Delta_{r,s} = \Psi(\alpha)$ mod $\{\Upsilon_1, \ldots, \Upsilon_n\}$ is called the *distance* from r to s.[4] For the sake of brevity, we shall denote $\Delta_{\imath,s}$ by Δ_s. □

In TS_7 (shown in Fig. 4), for instance, there are two Parikh-incomparable short paths $\imath[bac\rangle s$ and $\imath[dae\rangle s$. Yet the distance Δ_s is uniquely defined as the Parikh vector $\Delta_s = \Psi(a)$.

From Definition 13 and Proposition 2, one immediately gets:

Corollary 2 (Distance addition and equivalent distances) *Let $TS = (S, T, \to, \imath)$ be finite, totally reachable, weakly forward deterministic and persistent. Suppose it also satisfies* $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$. *For any states $s, s', s'' \in S$, if $s' \in [s\rangle$ and $s'' \in [s'\rangle$, then $\Delta_{s,s''} = (\Delta_{s,s'} + \Delta_{s',s''})$ mod $\{\Upsilon_1, \ldots, \Upsilon_n\}$.*

Moreover,

$$\Delta_{s_0,s_1} = \Delta_{s_0,s_2} \iff \Delta_{s_1,s_3} = \Delta_{s_2,s_3}$$

for all states $s_0, s_1, s_2, s_3 \in S$ with $s_1, s_2 \in [s_0\rangle$ and $s_3 \in [s_1\rangle \cap [s_2\rangle$. □

Even though, in TS_7, the distance from $\imath$ to s is defined as $\Psi(a)$, there is no path from $\imath$ to s which actually realises this distance, i.e., has Parikh vector $\Psi(a)$. The next definition expresses that each distance is realised.

Definition 14 (*Distance paths, and distance-path property*) Let $TS = (S, T, \to, \imath)$ be finite, totally reachable, deterministic, and persistent. Suppose it also satisfies $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$.

If there is a path $r[\sigma\rangle s$ between r and s such that $\Psi(\sigma) = \Delta_{r,s}$, we shall say it is a *distance-path*. TS satisfies the *distance-path property* if there is a distance-path $r[\sigma\rangle s$ between any pair of states $r, s \in S$ with $s \in [r\rangle$. □

If $r[\sigma\rangle s$ is a path from r to s and for no $\Theta \in \{\Upsilon_1, \ldots, \Upsilon_n\}$, $\Theta \leq \Psi(\sigma)$, then $r[\sigma\rangle s$ is necessarily a distance-path. The converse is also true; i.e.:

Proposition 3 (Distance paths are *cycle-reduced* paths) *Let $TS = (S, T, \to, \imath)$ be finite, totally reachable, deterministic, and persistent. Suppose it also satisfies* $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$.

For $r, s \in S$ and $\sigma \in T^*$, $r[\sigma\rangle s$ is a distance-path if it is a cycle-reduced path.

Proof

$$\begin{aligned} r[\sigma\rangle s \text{ is a distance-path} &\iff \Psi(\sigma) = \Delta_{r,s} \\ &\iff \neg\exists \Theta \in \{\Upsilon_1, \ldots, \Upsilon_n\} \colon \Psi(\sigma) \geq \Theta \\ &\iff r[\sigma\rangle s \text{ is cycle-reduced} \end{aligned}$$

where the central equivalence comes from the definition of Δ. □

[4] Note that, contrary to what happens in metric spaces where distances are usually scalar and symmetric, in (labelled) oriented graphs like here we may have (vectorial) non-symmetric distances.

Hence the structure of an arbitrary path in the class of transition systems under consideration can be understood as follows.

Corollary 3 ("Lasso structure" of the Parikh vectors of paths) *Let* $TS = (S, T, \rightarrow, \iota)$ *be finite, totally reachable, deterministic, and persistent. Suppose that it also satisfies* $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$. *Suppose that* $r, s \in S$ *with* $s \in [r\rangle$. *Every path* $r[\sigma\rangle s$ *satisfies* $\Psi(\sigma) = \Delta_{r,s} + \Psi(\kappa)$, *where* $\Psi(\kappa)$ *is cyclic (i.e., there is some* $s' \in S$ *with* $s'[\kappa\rangle s'$*). If, in addition, the distance-path property holds, then* $\Delta_{r,s}$ *can be realised by a short, cycle-reduced path from* r *to* s.

Proof The first claim follows directly from the well-definedness of distances (Proposition 2). The second claim follows from the definition of the distance-path property, together with Proposition 3. □

Thus, in transition systems which are boundedly and choice-freely solvable, all Parikh vectors of paths are composed of some Parikh vector of a small (as well as Parikh-minimal and cycle-reduced) path, plus that of a cycle. In TS_7, such a structure is absent. For example, the path $\iota[bac\rangle s$ satisfies $\Psi(bac) = \Psi(a) + \Psi(bc)$, and while there is a cycle with Parikh vector $\Psi(bc)$, there is no small path from ι to s with Parikh vector $\Psi(a)$.

Next, we show that not all paths between pairs of states need to be checked in order to verify the distance-path property; the ones starting from ι are enough. This allows the distance-path property (thus, equivalently, by Proposition 3, also the cycle-reduction property) to be tested efficiently.

Proposition 4 (An efficient test for the distance-path property) *Let* $TS = (S, T, \rightarrow, \iota)$ *be finite, totally reachable, deterministic, and persistent. Suppose it also satisfies* $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$. *Then TS satisfies the distance-path property iff for all* $s \in S$, *there is a path* $\iota[\alpha\rangle s$ *such that* $\Psi(\alpha) = \Delta_s$.

Proof See Appendix B. □

Finally in this section, we cite a result showing that, in the given context, the prime cycles and distance-path properties are strong enough to imply other global properties.

Theorem 5 (Power of prime cycle and distance-path properties [15]) *Let* $TS = (S, T, \rightarrow, \iota)$ *be finite, totally reachable, deterministic, and persistent. Suppose that it also satisfies the prime cycle property and the distance-path property. Then TS is fully forward and backward deterministic as well as weakly periodic.* □

5 Pre-synthesis of bounded choice-free nets

This section describes an algorithm which can be used in order to check, at once, the properties mentioned previously. Indeed, Corollary 3 and Theorem 5 reveal that, for finite, totally reachable, deterministic, and persistent transition systems, testing prime cyclicity and the existence of distance-paths through Proposition 4—or equivalently, by Proposition 3, the cycle-reduction property—obliterates the need to test full forward/backward determinism and weak periodicity. Henceforth, in this section, we will assume that a given transition system *TS* is

input a finite, deterministic, and persistent lts $TS = (S, T, \to, \iota)$;
initially $\mathcal{L}_1 := \mathcal{R} := \{\iota\}$, $\mathcal{L}_2 := \mathcal{P} := \emptyset$, $\mathcal{A} :=\to$, and $\Delta_\iota := \mathbf{0}$;
while $\mathcal{L}_1 \neq \emptyset$ **do**
  while there is some $(s, t, s') \in \mathcal{A}$ with $s \in \mathcal{L}_1$ and $s' \notin \mathcal{R}$ **do**
    remove (s, t, s') from $\mathcal{A}$; add s' to $\mathcal{R}$ and to $\mathcal{L}_2$; $\Delta_{s'} := \Delta_s + \delta_t$;
  end while;
  $\mathcal{L}_1 := \mathcal{L}_2$; $\mathcal{L}_2 := \emptyset$;
end while
%now we know the Parikh vectors of a short path from ι to s, for every $s \in S$;
if $\mathcal{R} \neq S$ **then** {**output** "TS is not totally reachable"; **fail**};
while there is some $(s, t, s') \in \mathcal{A}$ **do**
  remove (s, t, s') from $\mathcal{A}$;
  if $\neg(\Delta_{s'} \leq \Delta_s + \delta_t)$ **then** {**output** "some necessary properties are missing" 1 ; **fail**}
  $update\mathcal{P}((\Delta_s + \delta_t - \Delta_{s'}) \bmod \{\mathcal{P}\})$
end while
for $s \in S$, $\Theta \in \mathcal{P}$ **do**
  if $\Theta \leq \Delta_s$ **then** {**output** "some necessary properties are missing" 2 ; **fail**}
end for
output "TS is totally reachable, weakly periodic, satisfies $\mathbf{P}\mathcal{P}$,
presents the prime cycle property and the distance-path property,
and for each state s, Δ_s was computed correctly. "

procedure $update\mathcal{P}(v)$:
if $v = \mathbf{0}$ **then return** ;
if $supp(v) \not\subseteq \cup_{\Theta \in \mathcal{P}} supp(\Theta)$
then{let v' be the restriction of v to $supp(v) \setminus \cup_{\Theta \in \mathcal{P}} supp(\Theta)$; add $v'/\gcd(v')$ to $\mathcal{P}$; }
for $\Theta \in \mathcal{P}$
  if $supp(v) \cap supp(\Theta) \neq \emptyset$ **then**
    {let v' be v restricted to $supp(\Theta)$;
      if v' is a multiple of Θ **then continue**;
      suppress Θ from $\mathcal{P}$;
      for each rational q such that, for some $t \in supp(\Theta)$, $v(t) = q \cdot \Theta(t)$
      do let $TT = \{t \in supp(\Theta) | v(t) = q \cdot \Theta(t)\}$;
        let v'' be Θ restricted to TT;
        add $v''/\gcd(v'')$ to $\mathcal{P}$
      end for
    }
end for

Fig. 6 An algorithm checking the remaining preconditions (apart from finiteness, weak forward determinism, and persistence). Weak backward determinism might also be assumed. However, this is not essential for the rest of the algorithm, and it will be guaranteed by the rest of the checks by Theorem 5; but checking this property beforehand is easy, may speed up the pre-synthesis, and allows to produce a more specific error message in case of failure

finite, deterministic, persistent

since all three properties are necessary for the synthesis. As noticed earlier (in Sect. 4), they are also easy to test. Finiteness is not an issue if *TS* is given directly, by listing its states and labelled arcs. Weak forward determinism, as well as persistence, are local properties, hence not too difficult to assert or refute; then, if the check is passed, by Proposition 1, full forward determinism is also ascertained.

All other properties are checked by the algorithm shown in Fig. 6.

Effectively, this algorithm consists of three successive loops. In the first loop, a spanning tree is constructed, and for every state s, the Parikh vector Δ_s of a small path from ι to s is saved.[5]

The second loop scans all edges which are not in the spanning tree. By these edges, a cycle is created back to some state s' which has already been seen before. In the regular (choice-freely synthesisable) case, the saved short Parikh vector $\Delta_{s'}$ is the vector of a short path, of which all others are extensions by Corollary 3. Hence in this case, the test around $\boxed{1}$ must succeed, and if it does not, then we can already stop with an error message.

The third loop tests all Δ_s for cycle-reduction, because in the regular case, these are the true distances from ι to s and $\mathcal{P}$ contains the Parikh vectors of small cycles. The failure of any such test leads to an error exit $\boxed{2}$, which is sound by Proposition 4. The auxiliary procedure *update*$\mathcal{P}$ examines a newly obtained (candidate for a) cycle and adapts the members of $\mathcal{P}$ accordingly, preserving their primeness and mutual disjointness. A member of $\mathcal{P}$ may disappear at a further stage; but in the regular case, $\mathcal{P}$ eventually contains the small cycle Parikh vectors $\Upsilon_1, \ldots, \Upsilon_n$.

In case the algorithm produces errors by means of exits $\boxed{1}$ or $\boxed{2}$, the underlying reasons can already be narrowed down, because it is known at which states and paths the failure occurs. Further analysis might, however, be necessary in order to detect the true reason (or reasons) for the failure.

A full, detailed explanation of the algorithm will be given in the proof of the following theorem.

Theorem 6 (The algorithm shown in Fig. 6 is correct) *When applied to a finite, deterministic, persistent lts, the algorithm succeeds iff it is totally reachable and satisfies the prime cycle property as well as the cycle-reduction property. At the end, the lts satisfies* $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$, *and the computation yields* $\mathcal{P} = \{\Upsilon_1, \ldots, \Upsilon_n\}$, *as well as* Δ_s *for all* $s \in S$.

Proof Clearly, during the first while loop, $\mathcal{L}_1$ yields the set of states reachable from ι with exactly no arcs, then one arc, then two arcs, ..., and $\mathcal{L}_2$ constructs the next such set; whenever some Δ_s is initialised, it yields the Parikh vector of some path with minimal length; at the end $\mathcal{R}$ gathers all the reachable states, and $\mathcal{A}$ gathers the arcs not explored to construct $\mathcal{R}$. This amounts in fact to constructing a spanning tree of the lts rooted in ι.

($\Leftarrow$): **We assume** that the given lts is finite, deterministic, persistent, totally reachable, and satisfies the prime cycle property and the cycle-reduction property. **We show** that the algorithm succeeds and computes a suitable set $\{\Upsilon_1, \ldots, \Upsilon_n\}$ of Parikh vectors of mutually label-disjoint small cycles as well as all proper distances Δ_s.

From Proposition 1, we know that the lts is fully forward deterministic. From Theorem 4, the lts satisfies $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ for some mutually disjoint prime T-vectors $\Upsilon_1, \ldots, \Upsilon_n$. From Proposition 2, the distance notion is well defined, and from Propositions 3 and 4 we have the distance-path property.

Since the lts is totally reachable, at the end of the first while loop, $\mathcal{R} = S$ and the algorithm successfully passes the first check. Moreover, at that point, since the lts has the distance-path property and we constructed short paths from ι, each Δ_s equals the distance from ι to s.

We shall now show that, whenever one examines a remaining arc, $\mathcal{P}$ is composed of linear combinations of the Υ_i's, these members are prime and have disjoint supports, and each nontrivial cycle already discovered has a Parikh vector which is a linear combination of these

[5] It would have been possible to incorporate the tests for determinism and persistence in this loop, instead of doing them beforehand; but doing these tests separately is cheap, so that the gain seems negligible. See also Table 1 in Sect. 8.

members. Trivially, this is true at the beginning since $\mathcal{P}$ is initialised empty and no nontrivial cycle has been discovered yet.

When one examines a remaining arc, the states s and s' have already been visited. Then, $\Delta_s + \delta_t$ is the Parikh vector of some path from ι to s', and it must be greater or equal to $\Delta_{s'}$ since the latter is the distance from ι to s'; hence the next checks succeed. From the proof of Proposition 2 in [13], the Parikh vector of $(\Delta_s + \delta_t) - \Delta_{s'}$ is a linear combination of the Υ_i's, and we do not lose anything if we only consider it modulo $\mathcal{P}$: let us call it v. After refining $\mathcal{P}$ by v, $\mathcal{P}$ remains composed of prime linear combinations of the Υ_i's with disjoint supports, such that v is a linear combination of the members of the new $\mathcal{P}$, and each cycle formed by the arcs considered up to now has a Parikh vector which is a linear combination of the members of the new $\mathcal{P}$.

To see this, we may first observe that, if $supp(v) \not\subseteq \cup_{\Theta\in\mathcal{P}} supp(\Theta)$, the restriction v' of v (or its version before taking the modulo) to $supp(v)\backslash\cup_{\Theta\in\mathcal{P}} supp(\Theta)$ is a linear combination of some of the Υ_i's not occurring in $\mathcal{P}$, and if we add to the latter the prime version $(v'/\gcd(v'))$ of v', we have a prime vector disjoint from the other members of $\mathcal{P}$, and from the primality of the Υ_i's this vector still is a linear combination of the Υ_i's occurring in v'.

Now, let us assume v intersects the support of some member Θ of $\mathcal{P}$. We have both $v = \sum_i k_i \cdot \Upsilon_i$ and $\Theta = \sum_i h_i \cdot \Upsilon_i$, for some integer coefficients k_i and h_i. If $u \in supp(\Upsilon_i)$ with $k_i \neq 0 \neq h_i$, $k_i \cdot \Upsilon_i$ is proportional to $h_i \cdot \Upsilon_i$; as a consequence, by the way v'' is built in the ancillary procedure, it is composed of Υ_i's with a same proportionality factor, and the various prime versions of all those v''s satisfy the needed properties.

Finally, if a cycle has just been closed (the previous cycles have already been considered, and since the old members of $\mathcal{P}$ are linear combinations of the new ones, the Parikh vectors of those cycles remain linear combinations of the new members), it is not guaranteed that we shall consider two paths of the form $\iota[\alpha\rangle s$ and $\iota[\alpha\rangle s[\tau\rangle s$ before applying $update\mathcal{P}$ because other paths may have been considered for the intermediate state, but then, by construction, the Parikh vectors of the residues are divisible by the present members of $\mathcal{P}$, so that the Parikh vector of τ is also divisible by those present members.

Hence, when a small cycle is closed, its Parikh vector (which is one of the Υ_k's) is a linear combination of the members of $\mathcal{P}$, which are themselves linear combinations of the Υ_k's, and that is only possible if Υ_k is itself a member of $\mathcal{P}$. As a consequence, at the end, $\mathcal{P} = \{\Upsilon_1, \ldots, \Upsilon_n\}$.

But then the last checks also succeed since a distance may not be greater than any Υ_i.

($\Rightarrow$): **We assume** that the given lts is finite, deterministic, persistent, and that the algorithm succeeds. **We show** that the lts is totally reachable and satisfies the prime cycle property as well as the cycle-reduction property, and that, moreover, the results it computes are correct.

Since, at the end of the first while loop, $\mathcal{R}$ is the set of reachable states, the lts is totally reachable iff the first check succeeds.

During the first loop, one discovers short paths from the initial state to the various states; for any state s', if we discover another short path to it during the second loop, since the second check succeeds, that means that all the short paths from the initial state to some state have the same Parikh vector, and that those are the unique Parikh-minimal paths to s.

Let us consider an arc $s[t\rangle s'$. If it is discovered during the first loop, we have $\Delta_{s'} = \Delta_s + \delta_t$; otherwise, from the way it is handled, if the second checks succeeds, $\Delta_s + \delta_t - \Delta_{s'} = \sum_{\Theta\in\mathcal{P}} k_\Theta \cdot \Theta$ for some natural numbers k_Θ, and this remains true after each refinement. Adding those relations for all the arcs of any loop, all the δ_t will cancel out, and we shall end with the fact that the Parikh vector of the loop is a linear combination of members of (the present) $\mathcal{P}$. Similarly, if $s[\sigma\rangle s'$ for some $s, s' \in S$ and some $\sigma \in T^*$ and we add those

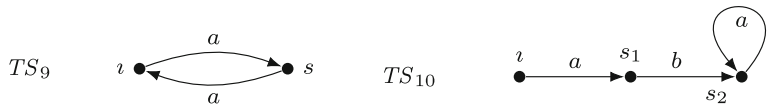

Fig. 7 Neither $TS_{aa.ts}$ nor TS_{10} can be solved by a Petri net

relations for all the arcs of σ, we shall get that $\Delta_s + \Psi(\sigma) = \Delta_{s'} + \sum_{\Theta\in\mathcal{P}} k_\Theta \cdot \Theta$, for some natural numbers k_Θ.

Let us now assume that $s[\sigma\rangle s'$ for some $s, s' \in S$ and some $\sigma \in T^*$ such that $\Psi(\sigma) = \sum_{\Theta\in\mathcal{P}} k_\Theta \cdot \Theta$. From the previous remark, since the check in the third loop always succeeds and all the members of $\mathcal{P}$ have disjoint supports, $\Delta_{s'} = \Delta_{s'} \bmod \mathcal{P} = \Delta_s \bmod \mathcal{P} = \Delta_s$, and since this is the Parikh vector of any short path from ι to s and to s', respectively, and the lts is fully forward deterministic, we have $s = s'$. The same is true if $\Psi(\sigma)$ is proportional to a linear combination of the members of $\mathcal{P}$ since, due to the primality and disjointness of the latter,[6] the same argument as the one used in the proof of Proposition 2 shows that a vector proportional to such a linear combination is a linear combination itself. Then, since any cycle has a Parikh vector which is a linear combination of the members of $\mathcal{P}$ and the latter are prime, from the persistence of the system, we get the disjoint small cycles property $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ from Theorem 4.

Let us now show that $\mathcal{P}$ is exactly the set of the Parikh vectors of all the small cycles of the lts. We have to prove that the Parikh vector of each small cycle is a member of $\mathcal{P}$, and that each member of $\mathcal{P}$ is the Parikh vector of a small cycle.

First, let us consider any small cycle. Since the relation $\leq$ is a partial order on T-vectors, let us choose among the states of the considered small cycle one with a maximal Δ: let us call it s and let $s[t\rangle s'[\alpha\rangle s$ be that cycle. At the end of the while loop, $\Psi(t\alpha) = \sum_{\tau\in\mathcal{P}} k_\tau \cdot \tau$, for some integers k_τ; now, $\Theta(t) > 0$ for some unique $\Theta \in \mathcal{P}$ since the members of $\mathcal{P}$ are disjoint, so that $k_\Theta > 0$. We have $\Delta_{s'} = (\Delta_s + \delta_t) \bmod \mathcal{P} = \Delta_s + \delta_t - \Theta$ since by construction s' may not be at a higher distance from ι than s and we may not subtract members of $\mathcal{P}$ from Δ_s: as a consequence we may only subtract Θ to cancel out δ_t, and $\Delta_s + \delta_t - \Theta \leq \Delta_s$ so that $(\Delta_s + \delta_t - \Theta) \bmod \mathcal{P} = \Delta_s + \delta_t - \Theta$. Hence, since Δ_s and $\Delta_{s'}$ are the Parikh vectors of paths from ι to s and s', respectively, and since $\Delta'_s = \Delta_s + \delta_t - \Theta \leq \Delta_s$, by Keller's theorem (Appendix A), there are a path $s'[\beta\rangle s$ with Parikh vector $\Psi(\beta) = \Theta - \delta_t$ and a cycle $s[t\beta\rangle s$ with Parikh vector Θ. This cannot be a smaller nontrivial cycle than the small one $t\alpha$, leading to $k_\Theta = 1$ and all the other k_τ being zero. Hence the claimed property.

Now, to prove that any $\Theta \in \mathcal{P}$ is the Parikh vector of some small cycle, we may observe that, from the construction of $\mathcal{P}$ and the disjoint small cycle property shown in the previous paragraph (since each small cycle has its Parikh vector in $\mathcal{P}$ and the latter is only composed of disjoint prime vectors), there is an integer $k > 0$ and integer coefficients h_i such that $k \cdot \Theta = \sum_i h_i \cdot \Upsilon_i$. Since we have shown that each Υ_i is equal to some element of $\mathcal{P}$ and $\mathcal{P}$ is composed of disjoint vectors, this is only possible if for some i we have $\Theta = \Upsilon_i$. As a consequence, we have the prime cycle property.

By construction, if all the tests are passed, we have distance-paths from ι to any reachable state. We may then use Proposition 4 to infer that we have the distance-path property, and Proposition 3 to infer that we have the cycle-reduction property. □

With this pre-synthesis algorithm, both TS_9 and TS_{10}, as depicted in Fig. 7, fail at exit 2. TS_9 fails because in the first loop, a short, non-cyclic path with Parikh vector $\Delta_s =$

[6] Seen as in the other part of the proof.

$\Psi(a)$ is computed. Then, in the second loop, a cycle with Parikh vector $\Psi(aa)$ is detected, which by *update*$\mathcal{P}$ becomes a T-vector $\Theta = \Psi(a)$ which must, in the regular (choice-freely synthesisable) case, belong to a small cycle with Parikh vector $\Psi(a)$. Then, in the third loop, $\Theta \leq \Delta_s$ holds, and $\boxed{2}$ is executed; in the regular case, this would indicate the existence of a small path which, in this case, is also non-cycle-reduced. TS_{10} fails similarly. The distance $\Delta_{s_2} = \Psi(ab)$ is computed, and then also the Parikh vector $\Theta = \Psi(a)$ of a small cycle. This yields $\Theta < \Delta_{s_2}$, and $\boxed{2}$ is entered in the third loop. By contrast, TS_4 (depicted in Fig. 3) already fails at $\boxed{1}$ in our implementation. This is because, at first, $\Delta_\iota = \Psi(\varepsilon)$ and $\Delta_s = \Psi(b)$ are computed due to the short paths $\iota[\varepsilon\rangle\iota$ and $\iota[b\rangle s$. Then, the chord $s[a\rangle\iota$ is detected, yielding $\delta_a = \Psi(a)$, and also leading to $\neg(\Delta_s \leq \Delta_\iota + \delta_a)$, and subsequently to error $\boxed{1}$.

6 Synthesis of bounded choice-free nets

In this section, we turn our attention to the synthesis of bounded choice-free Petri nets. Thus, let a labelled transition system $TS = (S, T, \rightarrow, \iota)$ be given. We are looking for a bounded choice-free net $N = (P, T, F, M_0)$ with a reachability graph isomorphic to TS. First of all, we shall assume that TS is

$$\boxed{\text{finite, totally reachable, fully deterministic, persistent, weakly periodic}} \tag{2}$$

As we have seen, transition systems not satisfying these properties can be rejected straight away.

The properties in (2) have been chosen because they are a small set of premises under which our synthesis algorithm can be shown to work correctly (possibly with a negative answer). Thus, any pre-synthesis method guaranteeing at least these properties cooperates correctly with the synthesis algorithm to be described in the present section. Note that by the previous theory (in particular, Theorem 5), the pre-synthesis algorithm described in Sect. 5 guarantees Properties (2) for any surviving transition system. Observe, however, that this algorithm establishes a much stronger set of properties.

In Sect. 6.1, we shall describe in which way the special structure of places in a choice-free Petri net allows to reduce the linear-algebraic burden of synthesis. In Sect. 6.2, we shall derive a concrete synthesis algorithm and prove its correctness.

6.1 Analysis of choice-free synthesis obligations

In general synthesis, two kinds of separation problems need—and suffice—to be solved, *state separation problems* and *event/state separation problems* [1]. We shall see in due course that state separation problems play no role in bounded choice-free synthesis. However, pre-synthesis is not capable of obliterating all event/state separation problems.

By Theorem 3, Properties (2) imply the small cycles property, i.e., the existence of a set of mutually label-disjoint Parikh vectors $\Upsilon_1, \ldots, \Upsilon_n$ such that $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ holds. Actually, such a set is readily constructed for TS if it survives the pre-synthesis procedure explained in Sect. 5. Note also that if pre-synthesis works as described there, we shall additionally have the prime cycle property and the distance-path property, even if Properties (2) are not strong enough to imply them, as shown by TS_5 and TS_7.[7]

[7] For any alternative pre-synthesis guaranteeing the properties in (2) but not necessarily the prime cycle property and the distance-path property, any lts not satisfying the latter will be rejected during synthesis.

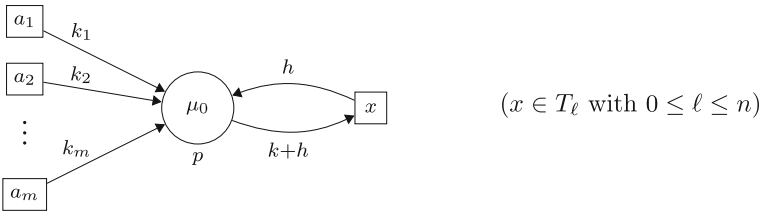

Fig. 8 A general pure ($h = 0$) or non-pure ($h > 0$) choice-free place p with initial marking μ_0. Place p has at most one outgoing transition named x. The set $\{a_1, \ldots, a_m\}$ comprises all other transitions, i.e., $T = \{x, a_1, \ldots, a_m\}$, and k_j denotes the weight of the arc from a_j to p (which could be zero). For every a_j, there is exactly one i such that $a_j \in T_i$, and ℓ is defined as the unique index such that $x \in T_\ell$

Without loss of generality, we may assume that no label is superfluous in TS. Indeed, if t does not occur in $\rightarrow$, then TS is (choice-freely) PN-solvable if and only if $(S, T\backslash\{t\}, \rightarrow, \iota)$ is. Further, the assumptions about TS imply $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$; let the small cyclic Parikh vectors $\{\Upsilon_1, \ldots, \Upsilon_n\}$ be fixed from now on. The following notation provides a shorthand for the supports of Υ_i and the remaining transitions.

Notation 7 (THE SETS T_0 AND $T_1, \ldots, T_n$) T_i is the support of Υ_i, and T_0 is the set $T\backslash(T_1 \cup \ldots \cup T_n)$. □

As a consequence of $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$, and by the definition of T_0, all $n+1$ sets $T_0, T_1, \ldots, T_n$ are mutually disjoint. Since live transitions occur in some cycle, T_0 is precisely the set of non-live transitions. E.g., in TS_2 (Fig. 2), we have $T_0 = \emptyset$ and $T_1 = \{a, b, c\}$. In TS_3, we have $T_0 = \{a, d\}$, $T_1 = \{b\}$, and $T_2 = \{c\}$.

In our impending analysis, we will exploit the structure of the hoped-for Petri net $N = (P, T, F, M_0)$, namely, that every target place has at most one outgoing transition. This means that a place $p \in P$ has the general form shown in Fig. 8, with $x \in T$ as its only outgoing transition, and $\{a_1, \ldots, a_m\} = T\backslash\{x\}$. The arc weight parameters $k, h, k_1, k_2, \ldots, k_m$ and the initial marking μ_0 of p are the unknowns of the synthesis. They are required to be semipositive, which could include zero, e.g. $\mu_0 = 0$ if p has no tokens initially, or $k_i = 0$ if there is no arc from a_i to x.

Let p and its only output transition x, in the context of Fig. 8, be fixed for the rest of this section. We shall analyse which constraints such a place p must satisfy if it is newly introduced, and the conditions under which it has to be introduced for synthesis to succeed.

For $0 \le i \le n$, we denote by $I_i = \{j \mid a_j \in T_i\}$ the indices of transitions a_j for which $\Upsilon_i(a_j) > 0$. We shall also denote by ℓ the unique index such that $x \in T_\ell$, so that $\{a_j \mid j \in I_\ell\} = T_\ell\backslash\{x\}$, while for $i \neq \ell$ we have $\{a_j \mid j \in I_i\} = T_i$.

6.1.1 Ensuring that cycles are preserved

Since the net effect of firing x on p is $-k = (-(k+h)+h)$ and all Parikh vectors in $\{\Upsilon_1, \ldots, \Upsilon_n\}$ correspond to cycles, we must have

$$\forall i \in \{1, \ldots, n\} : \quad \sum_{j \in I_i} k_j \cdot \Upsilon_i(a_j) = k \cdot \Upsilon_i(x) \tag{3}$$

ensuring that if every transition t is fired $\Upsilon_i(t)$ times, the marking on p is reproduced. Note that this implies $k \ge 0$, and even $k > 0$ unless all the k_j's for $j \in I_i$ are null.

If $x \in T_0$, i.e., $\ell = 0$, all the right-hand sides of (3) are null, so that all k_j for $j \notin I_0$ must be null too; in other words, if $p^\bullet \subseteq T_0$, then ${}^\bullet p \subseteq T_0$. Thus, if x is non-live, then all transitions in ${}^\bullet p$ are also non-live. If $x \in T_\ell$ for $\ell \in \{1, \ldots, n\}$, the right-hand sides of (3) are null when $i \in \{1, \ldots, n\} \backslash \{\ell\}$, so that all k_j for $j \notin I_0 \cup I_\ell$ must be null too; in other words, if $p^\bullet \subseteq T_\ell$, then ${}^\bullet p \subseteq T_0 \cup T_\ell$. Thus, if x is live, hence part of some small cycle, then all transitions in ${}^\bullet p$ are either non-live, or live and part of the same small cycle.

6.1.2 Characterising the various markings

Let $\iota[\alpha\rangle r$ be any path to $r \in S$ (the lts is totally reachable). By the shape of the place shown in Fig. 8, by ${}^\bullet p \subseteq T_0 \cup T_\ell$, and by the firing rule, the marking $M_r(p)$ of place p at (the marking corresponding to) an arbitrary state $r \in [\iota\rangle$ is

$$M_r(p) = \mu_0 + \sum_{j \in I_0 \cup I_\ell} k_j \cdot \Psi(\alpha)(a_j) - k \cdot \Psi(\alpha)(x)$$

From the assumed properties of the system and Proposition 2, the distance notion is valid, and from Corollaries 1 and 3, $\Psi(\alpha) = \Delta_r + \sum_{\Phi \in \mathcal{P}} c_\Phi \cdot \Phi$ for some natural numbers c_Φ. Hence, from Eq. (3) we get

$$M_r(p) = \mu_0 + \sum_{j \in I_0 \cup I_\ell} k_j \cdot \Delta_r(a_j) - k \cdot \Delta_r(x) \tag{4}$$

which is independent of the specific path considered to reach r, as expected. We may in fact consider 3 distinct subcases of this formula:

- if $\ell = 0$, I_ℓ collapses with I_0 and we get

$$M_r(p) = \mu_0 + \sum_{j \in I_0} k_j \cdot \Delta_r(a_j) - k \cdot \Delta_r(x) \tag{5}$$

- if $\ell > 0$ but T_ℓ is a singleton, this corresponds to loops $s[x\rangle s$, I_ℓ is empty, $k = 0$ [see Eq. (3)] and x never occurs in Δ_r, so that we get

$$M_r(p) = \mu_0 + \sum_{j \in I_0} k_j \cdot \Delta_r(a_j) \tag{6}$$

- otherwise (if $\ell > 0$ and $|T_\ell| > 1$), using Eq. (3), we may eliminate k and get

$$M_r(p) = \mu_0 + \sum_{j \in I_0} k_j \cdot \Delta_r(a_j) + \sum_{j \in I_\ell} k_j \cdot \Big[\Delta_r(a_j) - \frac{\Upsilon_\ell(a_j)}{\Upsilon_\ell(x)} \Delta_r(x) \Big] \tag{7}$$

  (note that some coefficients may be rational, and not natural, if $\Psi_\ell(x) > 1$).

6.1.3 Ensuring that the markings on p do not prevent enabled transitions

Each marking, hence each $M_r(p)$ as given by Eq. (4), [or by Eqs. (5), (6) and (7)], must be nonnegative. Moreover, p may never prevent any enabled x.

Let us define $X(x) = \{r \in S | r[x\rangle\}$ and consider an edge $r[x\rangle r'$ for $r \in X(x)$. Then the marking $M_r(p)$ has to be at least $k + h$, and $M_{r'}(p)$ is at least h. More generally, if $l > 0$ and $r[x^l\rangle r'$, then we must have $M_r(p) \geq l \cdot k + h$, or equivalently, $M_{r'}(p) \geq h$. For this reason, when considering the marking of p at a state r with $r[x\rangle$, we first try to follow x-chains in forward direction as long as possible. If $r_0[x\rangle r_1[x\rangle r_r[x\rangle \ldots [x\rangle r_l$ with $r = r_0 \neq r_1$, then, by

weak periodicity, for any $l \geq 1$, r_l must be different from the previous r_i's; as a consequence, each x-chain starting with two different states can never hit an x-cycle, nor an x-branch (by determinism), and necessarily has a unique last state. We may have infinite x-paths, but only in case $r[x\rangle r$ (like those for b or for c in TS_3 shown in Fig. 2, see also Case 2 above), where state r can never be left by an x-edge.

Thus, we are interested in the following subset of states:

$$XNX(x) = \begin{cases} \{r \in S \mid [x\rangle r \wedge \neg r[x\rangle\} & \text{if } \ \ell = 0 \text{ or } \ell > 0 \text{ but } |T_\ell| > 1 \\ X(x) & \text{if } \ \ell > 0 \text{ and } T_\ell = \{x\} \end{cases}$$

which either are produced by x but do not enable x, or have an x-loop. The above considerations amount to a proof of the following corollary:

Corollary 4 (*XNX* and enabling condition) $\big(\forall r \in X(x)\colon M_r(p) \geq k + h\big) \iff \big(\forall r \in XNX(x)\colon M_r(p) \geq h\big)$ □

In other words, we only need to require $M_r(p) \geq h$ for states $r \in XNX(x)$ in order to guarantee that place p allows x whenever it is enabled. But we can do more.

For two states r and r', we can compare their distances Δ_r and $\Delta_{r'}$ from the initial state. This allows us to derive relations between their numbers of tokens for any place p with output x, as follows:

Definition 15 (*Marking order w.r.t. some label*) We shall first denote by $r \equiv_x r'$ the fact that

- if $\ell = 0$, $\Delta_r(x) = \Delta_{r'}(x)$ and $\forall j \in I_0 : \Delta_r(a_j) = \Delta_{r'}(a_j)$,
- if $\ell > 0$ but T_ℓ is a singleton, $\forall j \in I_0 : \Delta_r(a_j) = \Delta_{r'}(a_j)$,
- otherwise (if $\ell > 0$ and $|T_\ell| > 1$), $\forall j \in I_0 : \Delta_r(a_j) = \Delta_{r'}(a_j)$ and $\forall j \in I_\ell : \Delta_r(a_j) - \frac{\Psi_\ell(a_j)}{\Psi_\ell(x)}\Delta_r(x) = \Delta_{r'}(a_j) - \frac{\Psi_\ell(a_j)}{\Psi_\ell(x)}\Delta_{r'}(x)$;

by $r \leq_x r'$ the fact that

- if $\ell = 0$, $\Delta_r(x) \geq \Delta_{r'}(x)$ and $\forall j \in I_0 : \Delta_r(a_j) \leq \Delta_{r'}(a_j)$,
- if $\ell > 0$ but T_ℓ is a singleton, $\forall j \in I_0 : \Delta_r(a_j) \leq \Delta_{r'}(a_j)$,
- otherwise (if $\ell > 0$ and $|T_\ell| > 1$), $\forall j \in I_0 : \Delta_r(a_j) \leq \Delta_{r'}(a_j)$ and $\forall j \in I_\ell : \Delta_r(a_j) - \frac{\Psi_\ell(a_j)}{\Psi_\ell(x)}\Delta_r(x) \leq \Delta_{r'}(a_j) - \frac{\Psi_\ell(a_j)}{\Psi_\ell(x)}\Delta_{r'}(x)$;

and by $r \lneq_x r'$ the fact that $(r \leq_x r') \wedge (r \not\equiv_x r')$. □

It should be clear that $\equiv_x \ = \ \leq_x \cap \geq_x$ is an equivalence, $\leq_x$ is a weak order, and $\lneq_x$ is a strict order. Moreover, from the Eqs. (5), (6) and (7), $r \equiv_x r'$ implies $M_r(p) = M_{r'}(p)$, $r \leq_x r'$ implies $M_r(p) \leq M_{r'}(p)$, and $r \lneq_x r'$ implies $M_r(p) \leq M_{r'}(p)$ (it may happen that $M_r(p) = M_{r'}(p)$, if some k_j's are null).

Let us now define

$$MXNX(x) = \left\{\, r \in XNX(x) \;\middle|\; \nexists r' \in XNX(x)\colon r' \lneq_x r \right\}$$

i.e., we only consider states in $XNX(x)$ which are not "dominated" by other ones (in terms of the constraint on $M_r(p)$). This set may contain many $\equiv_x$-equivalent states, but we need only keep one of them. Let us therefore choose some set

$$\boxed{\begin{array}{l} mXNX(x) \ \subseteq \ MXNX(x) \\ \quad \text{with a single representative of each } \equiv_x\text{-equivalent class in it} \end{array}}$$

These considerations amount to a proof of

Corollary 5 (*mXNX* and enabling condition) $\big(\forall r \in S\colon r[x\rangle \Rightarrow M_r(p) \geq k + h\big) \iff \big(\forall r \in mXNX(x)\colon M_r(p) \geq h\big)$ □

In other words, we only need to require $M_r(p) \geq h$ for states $r \in mXNX(x)$ in order to guarantee that place p allows x whenever it is enabled. Combining Corollary 5 with the formula (4) relating any $M_r(p)$ to μ_0 yields the following set of constraints:

$$\forall r \in mXNX(x)\colon \quad \mu_0 \quad \geq \quad k \cdot \Delta_r(x) - \sum_{j \in I_0 \cup I_\ell} k_j \cdot \Delta_r(a_j) + h \tag{8}$$

Let us now re-examine the constraint that $\forall r \in [\iota\rangle\colon M_r(p) \geq 0$. By $r \in [\iota\rangle$, there is a path $\iota[\alpha\rangle r$. If α contains an x, let s be the visited state before the last x; from the previous constraints, $M_s(p) \geq k+h$ and $M_r(p) \geq h \geq 0$. If α contains no x, then it has a semipositive effect on p and thus, $\mu_0 \leq M_r(p)$. It is then enough to impose $\mu_0 \geq 0$ to get the desired property $M_r(p) \geq 0$. However, $\mu_0 \geq 0$ can always be ensured by adding, if necessary, an adequate shift to all markings of p, as well as to h.

To summarise the discussion, requiring the non-negative solvability of (8) suffices in order to find parameters $k, h, k_1, \ldots, k_m, \mu_0$ in such a way that the corresponding place allows all the paths that are possible in *TS*.

6.1.4 Ensuring that place p solves an event/state separation problem

For each state s not enabling x, there should be a place p which does not have enough tokens to allow to perform x when reaching the state corresponding to s. That is, in addition to the constraints derived above, p should satisfy $M_s(p) < k + h$, i.e., using (4) with $r = s$:

$$\mu_0 \quad < \quad k + h + k \cdot \Delta_s(x) - \sum_{j \in I_0 \cup I_\ell} k_j \cdot \Delta_s(a_j) \tag{9}$$

However, it may happen that the same place works for many such states, i.e., it is possible to reduce the number of such inequalities.

Let $NX(x) = \{s \in S | \neg s[x\rangle\}$. Since $s \lesssim_x s'$ implies $M_s(p) \leq M_{s'}(p)$, if $M_{s'}(p)$ prevents x at s', the same will be true for s. I.e., if (9) is ascertained for s', it is no longer necessary to bother about s. Hence, it makes sense to define

$$MNX(x) = \{s \in NX(x) \mid \nexists s' \in NX(x) : s \lesssim_x s'\}$$

and

$$\boxed{\begin{array}{l} mNX(x) \quad \subseteq \quad MNX(x) \\ \quad \text{with a single representative of each } \equiv_\ell\text{-equivalent class in it} \end{array}}$$

i.e., we consider states not allowing x such that no less favourable state already precludes to do it

Hence, for every $s \in mNX(x)$, we need to find a place satisfying (8) and (9).

Combining the constraints (8) and (9) allows to eliminate both μ_0 and h:

$$\boxed{\begin{array}{l} \forall r \in mXNX(x)\colon \\ 0 < k \cdot \big[\ 1 + \Delta_s(x) - \Delta_r(x)\ \big] + \sum_{j \in I_0 \cup I_\ell} k_j \cdot \big[\ \Delta_r(a_j) - \Delta_s(a_j)\ \big] \end{array}} \tag{10}$$

If the system (10) is solvable in the domain of natural numbers (with $k_j = 0$ if $j \notin I_0 \cup I_\ell$), let us define $\mu = \max\{k \cdot \Delta_r(x) - \sum_{j \in I_0 \cup I_\ell} k_j \cdot \Delta_r(a_j) \mid r \in mXNX(x)\}$. If $\mu \geq 0$, by

input an lts $TS = (S, T, \to, \imath)$;
Pre-synthesis: apply, for instance, the procedure described in Figure 6
Synthesis (at this point, we can compute, or we have immediately,
each Δ_s from $\Upsilon_1, \ldots, \Upsilon_n$):
initially T is the set of transitions, and P is empty;
for every $x \in T_\ell$ $(0 \leq \ell \leq n)$ and $s \in mNX(x)$ **do**
construct a set $mXNX(x)$ and the corresponding system (10/11);
if there is no solution in $\mathbb{N}$ to this system **then**
{**output** "TS is not choice-freely solvable, due to x, s, and system (10/11)"; **stop**};
choose a set of natural numbers $(k, k_1, \ldots, k_m)$ satisfying (10),
as well as (3) if $\ell \neq 0$, and compute h and μ_0;
add to P a place as in Figure 8, with weights $k_1, \ldots, k_m, k + h, h$,
and initial marking μ_0;
end for
output "The constructed net is bounded and solves TS choice-freely".

Fig. 9 An algorithm checking choice-free solvability and constructing a solution

choosing $h = 0$ and $\mu_0 = \mu$ we shall get a solution to the systems (8) and (9), with $\mu_0 \geq 0$. If $\mu < 0$, it is not possible to create a suitable pure place from this solution, but we may choose $h = -\mu$ and $\mu_0 = 0$ (realising the "adequate shift" referred to above), and we shall again get a solution to the systems (8) and (9), with $\mu_0 \geq 0$.

If $x \notin T_0$, the constraints (3) need to be fulfilled as well. Combining (3) and (10), we get (here keeping integer coefficients)

$$\begin{array}{l} \forall r \in mXNX(x): \\ 0 < \sum\limits_{j \in I_0 \cup I_\ell} k_j \cdot \big[\Upsilon_\ell(a_j) \cdot (1 + \Delta_s(x) - \Delta_r(x)) - \Upsilon_\ell(x) \cdot (\Delta_s(a_j) - \Delta_r(a_j))\big] \end{array} \tag{11}$$

If the system (11) is solvable in the domain $\mathbb{N}$, it is also possible to find a natural solution to both (3) and (10), by choosing a suitable value for k using (3), and, if necessary, multiplying the solution found by a common factor. Then we may choose h and μ_0 as described above.

6.2 Algorithm and proof of its correctness

Figure 9 summarises the resulting algorithm, where by "system (10/11)" we mean "system (10)" if $x \in T_0$ and "system (11)" if $x \in T_\ell$ for $1 \leq \ell \leq n$.

Theorem 8 (Correctness of the synthesis algorithm) *Let $TS = (S, T, \to, \imath)$ be finite, totally reachable, deterministic, weakly periodic, and persistent. Also suppose that TS satisfies $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$.*
If, for some $x \in T$ and $s \in mNX(x)$, the corresponding system (10/11) is not solvable, then TS has no choice-free solution. Otherwise, the constructed net is a choice-free solution of TS.

Proof If the system (10/11) associated with some $x \in T$ and $s \in mNX(x)$ is not solvable, then from the analysis conducted in Sect. 6.1, there is no place of a choice-free net both allowing all valid evolutions and precluding the invalid transition x from s.

Let us thus assume all those systems are solvable and let us consider the net constructed as above. We have seen that for any $s \in S$, if $s[x\rangle$, there is a state $r \in mXNX(x)$ such that $M_s(p) \geq M_r(p) \geq h$ if the synthesis succeeds. As a consequence, place p allows all valid

evolutions specified by the lts. If $\neg s'[x\rangle$, we know there is some $s \in mNX(x)$ such that $M_{s'}(p) \leq M_s(p)$. From the choice of $M_0(p)$ for the corresponding place p, and from (10), we have (9) whatever h, which precludes to perform x from s as well as from s'.

Therefore, the solvability of all systems (10/11) implies that all event/state separation problems [1] can be solved and from the general theory [1], the (choice-free) net thus obtained has the same language as TS.

The only way to have non-isomorphism is that some state separation problem cannot be solved, i.e., that two states s_1 and s_2 correspond to the same marking. In that case, let $s_1[\beta\rangle q_1$ be a path to a home state q_1 of TS. Since s_1 and s_2 correspond to the same marking and $L(N) = L(TS)$, $s_2[\beta\rangle q_2$ for some state q_2 corresponding to the same marking as q_1. Since q_1 is a home state, there is a path $q_2[\alpha\rangle q_1$. Since the language is the same and q_1, q_2 correspond to the same marking, we have $q_2[\alpha\rangle q_1[\alpha\rangle q_3[\alpha\rangle q_4 \ldots$, and from finiteness and weak periodicity, we must have $q_1 = q_2$. But then, by weak backward determinism, we also have $s_1 = s_2$.

As a consequence of this and the general theory [1], the constructed choice-free net solves the given lts TS. □

7 Some remarks on the synthesis algorithm

In Sect. 7.1, we examine the implementability of various sets of states identified in the previous analysis. Section 7.2 describes applications of the synthesis algorithm to some concrete transition systems defined in earlier parts of this paper. In Sect. 7.3, various special cases (and relationships to published literature) will be described. Finally, some complexity considerations can be found in Sect. 7.4.

7.1 Structure-based sets

It may be observed that, for each $x \in T$, the definition of the set $mXNX(x)$ is essentially computational, since it is based on arithmetic formulas about the distances. But any set M with $mXNX(x) \subseteq M \subseteq XNX(x)$ may be used instead in formula (8) to guarantee that no existing x-labelled arc is prevented. In particular we shall here consider a more structural definition, thus easing the reasoning about those notions, using another equivalence relation between states of S, based on the class of transitions x belongs to (let us recall that we assumed $x \in T_\ell$):

$$q \equiv^\ell q' \iff \exists \beta, \beta' \in (T \backslash (T_0 \cup T_\ell))^* : q[\beta\rangle q' \text{ and } q'[\beta'\rangle q$$

The equivalence classes w.r.t. $\equiv^\ell$ thus corresponds to the reachability classes of the system when arcs labelled in T_0 and T_ℓ are ignored. These classes can be computed quickly, for example with the well-known Tarjan algorithm. Since the cycle $s[\beta\beta'\rangle s$ satisfies $\Psi(\beta\beta') = \sum_i k_i \cdot \Upsilon_i$, labels in T_0 cannot occur in it, and the fact that there is no label from T_ℓ is the same as saying there is no x in the cycle.

This leads to the following definitions (where the "s" means "structural")

$$\begin{array}{l} MXNXs(x) = \{r \in XNX(x) \mid \nexists r' \in XNX(x) : r' \not\equiv^\ell r \\ \qquad\qquad\qquad\qquad \text{and } r'[\alpha\rangle r \text{ with } \alpha \in (T \backslash \{x\})^+\} \\ mXNXs(x) \subseteq MXNXs(x) \\ \quad \text{with a single representative of each } \equiv^\ell\text{-equivalent class in it} \end{array}$$

i.e., we only consider states in $XNX(x)$ which do not lie x-freely after another (non-$\equiv^{\ell}$-equivalent) one, and we keep a single representative of each class. To see that

$$MXNX(x) \subseteq MXNXs(x) \subseteq XNX(x) \text{ and } mXNX(x) \subseteq mXNXs(x) \subseteq XNX(x)$$

we may observe that, if $r[\beta\rangle r'$, $\Delta_{r'} = (\Delta_r + \Psi(\beta)) \bmod \{\Upsilon_1, \ldots, \Upsilon_n\}$; as a consequence, if no label from $T_0 \cup T_\ell$ occurs in β, then $\Delta_{r'}$ and Δ_r may only differ on $\{T_i | 0 < i \neq \ell\}$ and, from the definition of $\equiv_x$, $r \equiv^{\ell} r' \Rightarrow r \equiv_x r'$ (or $\equiv^{\ell} \subseteq \equiv_x$). Similarly, if no x occurs in β, $\Delta_{r'}(y) \geq \Delta_r(y)$ for $y \in T_0$, and $\Delta_{r'}(x) = \Delta_r(x)$ for $x \in T_0$; if $x \in T_\ell$ with $\ell > 0$ and $|T_\ell| > 1$, we may have $\Delta_{r'} = \Delta_r + \Psi(\beta) - k \cdot \Upsilon_\ell$ for some $k \geq 0$, but since $\Psi(\beta)(x) = 0$, we may deduce from the previous definitions that $r \leq_x r'$. The claimed inclusions immediately result.

Analogously, the set $mNX(x)$ is essentially computational, since it is based on arithmetic formulas about the distances, but again, in formula (11) we may use any set between $mNX(x)$ and $NX(x)$. In particular we may exploit the relation $\equiv^{\ell}$, and this leads to the following (more structural and more local) definitions:

$MNXs(x) = \{s \in S \mid \neg s[x\rangle \text{ and } \forall s[a\rangle r\colon (s \equiv^{\ell} r) \vee r[x\rangle\}$
$mNXs(x) \subseteq MNXs(x)$
with a single representative of each $\equiv^{\ell}$-equivalent class in it

i.e., we consider states not allowing x such that no non-equivalent successor also precludes performing x, and we keep one representative of each class. From the relationships we derived above between $\equiv_x$ and $\equiv^{\ell}$, as well as between $\leq_x$ and x-free paths, we get as expected the inclusions: $MNX(x) \subseteq MNXs(x) \subseteq NX(x)$ and $mNX(x) \subseteq mNXs(x) \subseteq NX(x)$.

Finally, it may be observed that a naive application of the definitions of $MNX(x)$ and $mNX(x)$ is essentially quadratic in the size of S. However, a direct computation of $mNX(x)$ may be rendered more efficient, as illustrated by the following algorithm:

input an lts $TS = (S, T, \rightarrow, \imath)$ and a label $x \in T$;
$mNX(x) := \emptyset$;
for every $s \in S$ **do**
  if $\neg s[x\rangle$ **continue**;
  for every $s' \in mNX(x)$ **do**
    if $s \leq_x s'$ **then continue** the for loop on s ;
    if $s' \leq_x s$ **then** drop s' from $mNX(x)$;
  add s to $mNX(x)$;

7.2 Four examples

We illustrate the constructions of Sect. 6 on four examples, TS_3, TS_7, TS_8, and TS_9, shown, respectively, in Figs. 2, 4, 5, and 7. Note that TS_3 has a choice-free solution while TS_7 does not, and that TS_8 and TS_9 have no solution at all.

- Consider TS_3. For $x = a$, we get $x \in T_0 = \{a, d\}$, $I_\ell = I_0 = \{1\}$ if $a_1 = d$, $mXNXs(a) = \{s_1\}$, $mNXs(a) = \{s_2\}$ and Eq. (10) reduces to

$$0 < k \cdot [1 + 1 - 1] + k_1 \cdot [0 - 1]$$

  which leads to the solution $k = 1, k_1 = 0, \mu_0 = 1$ and $h = 0$, corresponding to place p_1 in N_3. For $x = b$, we get $x \in T_1 = \{b\}$, $T_0 = \{a, d\}$, $I_0 = \{1, 2\}$ if $a_1 = a$ and $a_2 = d$,

$I_1 = \emptyset$, $mXNXs(b) = \{s_1\}$, $mNXs(b) = \{\imath\}$ and Eq. (11) reduces to

$$0 < k_1 \cdot [0 - 1 \cdot (0 - 1)] + k_2 \cdot [0 - 1 \cdot (0 - 0)]$$

which leads to the solution $k_1 = 1$, $k = k_2 = 0$, $\mu_0 = 0$ and $h = 1$, corresponding to place p_2 in N_3. The treatments of c and d are similar.

- Consider now TS_7. For $x = a$, we get $x \in T_0 = \{a\}$, $I_\ell = I_0 = \emptyset$, $MXNXs(a) = \{s_4, s_5, s_6\}$ (that is, all states are $\equiv^0$-equivalent), $MNXs(a) = \{\imath, s_4, s_5, s_6, s\}$, and, for example, $mXNXs(a) = \{s_4\}$ and $mNXs(a) = \{\imath, s\}$. For $x = \imath$, Eq. (10) reduces to

$$0 < k \cdot [1 + 0 - 1] + 0$$

which is unsolvable.
Note that TS_7 would be rejected by a pre-synthesis algorithm testing for the distance-path property, before this failure occurs. This shows that our proper synthesis algorithm is stable, in the sense that it needs only the premises listed at the beginning of Sect. 6. The other constraints, like the distance-path property, are not necessary to apply the proper synthesis procedure, at the price of less informative and later failures.
- As a third example, let us consider TS_8, a non-synthesisable transition system which survives our pre-synthesis algorithm. Here we have $T = T_0 = \{a, b\}$. For $x = a$, we get $XNXs(a) = \{s_1, s_5\} = MXNXs(a) = mXNXs(a)$ and $MNXs(a) = \{s_2, s_5\} = mNXs(a)$, since $\equiv^0$ is the identity. Equation (11) yield the set of constraints $k - k_1 > 0$ and $-k + k_1 > 0$, which is unsolvable.
- Finally, consider TS_9. TS_9 satisfies all properties required in (2), except weak periodicity; it can therefore be used in order to explain the role of weak periodicity. The set $XNX(a)$ is empty in this case, since neither terminating backward a-chains nor a-loops exist in TS_9, and so are all other derived sets. When applied to TS_9, the synthesis algorithm terminates and produces a single transition a with no surrounding places. The reachability graph of this very simple net is language-equivalent, but not isomorphic, to TS_9. In a nutshell, we can use the synthesis algorithm even without requiring weak periodicity, but we will get language-equivalent (not necessarily exact) solutions.

7.3 Some special cases

Let us consider what happens to the general procedure if TS satisfies some additional properties. Since computational sets ($mXNX$, …) are less easy to characterise in general than structural ones ($mXNXs$, …), we shall use the latter here. This will also make it more easy to compare our techniques to previous papers on the subject.

- All transitions are live. Then $T_0 = \emptyset$; $\equiv^0$ is identity; and any $a_j \in {}^\bullet p$ corresponds to the same small cycle as $x \in p^\bullet$. If synthesis succeeds, the resulting choice-free net is a disjoint composition of n individual choice-free nets, each one connected by itself and corresponding to one of the Parikh vectors Υ_i (for $1 \leq i \leq n$). In essence, therefore, this reduces to the next case.
- All transitions are live and there is a single small cyclic Parikh vector [12]. Then both $\equiv^0$ and $\equiv^1$ reduce to identity. All sets $mXNXs(x)$ and $mNXs(x)$ are unique and can be simplified as follows:

$$\begin{aligned} XNXs(x) &= \{s \in S \mid [x\rangle s \wedge \neg s[x\rangle\} \\ mXNXs(x) &= \{s \in XNXs(x) \mid \nexists s' \in XNXs(x) : s'[\alpha\rangle s \text{ with } \alpha \in (T \backslash \{x\})^+\} \\ mNXs(x) &= \{s \in S \mid \neg s[x\rangle \wedge \forall s' \in S, a \in T : s[a\rangle s' \Rightarrow s'[x\rangle\} \end{aligned}$$

Such transition systems are "almost reversible" (i.e., they consist of an initial acyclic part, followed by a single strongly connected component, such as TS_2 in Fig. 2).

- **Reversibility.** *TS* is reversible, i.e., ι is a home state [9,10]. All previous simplifications hold and, additionally, if there is a choice-free solution, then there is also a pure solution; indeed, in the discussion after Corollary 5, reversibility implies that, for any $r \in S$ and $x \in T$, when x indeed occurs on some arc, there is always a path $\iota[\alpha\rangle r$ containing an x and it is not necessary to add the constraint $\mu_0 \geq 0$: it is automatically satisfied. Nevertheless (despite the simpler setting), it is possible to construct reversible persistent transition systems which can be solved by a plain and pure Petri net, but not by a choice-free net [10].
- **The marked graph case.** *TS* belongs to a marked graph or to a T-system. Such transition systems have full characterisations for bounded as well as for unbounded Petri nets [11,14].
- **Acyclicity.** *TS* is acyclic, so that $T = T_0$. In that case, for each label $x \in T$, $mXNXs(x) = MXNXs(x)$, $mNXs(x) = MNXs(x)$, and we only have to solve for each x and $s \in mNXs(x)$, the systems

$$\forall r \in mXNXs(x) : 0 < k \cdot [1 + \Delta_s(x) - \Delta_r(x)] + \sum_{j \in I_0} k_j \cdot [\Delta_r(a_j) - \Delta_s(a_j)]$$

It is not known how this could be avoided completely, but better solvability conditions are known in special cases [8,30].

7.4 Complexity considerations

In the algorithm shown in Fig. 9, we have to solve, for each label $x \in T$, $|mNX(x)|$ systems, each with $|mXNX(x)|$ inequalities (or $|mNXs(x)|$ systems with $|mXNXs(x)|$ inequalities in the variant described in Sect. 7.1), and we should estimate the sizes of those two sets in order to quantify the complexity of the synthesis. In general, those sets are expected to have a moderate size, but they may in some circumstances be linear in the size of the lts.

Since solving linear inequality systems is possible in polynomial time [33], this already shows that our algorithm has polynomial time complexity. This is also the case for Petri net synthesis not targeted to a specific class [2], while other targeted classes might even be NP-complete [3]. In the remainder of this section, we estimate the complexity bounds of our algorithm more precisely.

Let us consider an arc $s[a\rangle s'$ with $s \neq s'$: we shall show that s and s' may not both belong to $mNX(x)$ nor to $mXNX(x)$.

If $a = x$, by definition, s may not belong to $NX(x)$ nor to $XNX(x)$. Let us now assume $a \neq x$. If $s \equiv^{\ell} s'$ (hence $s \equiv_x s'$), since only one representative is kept in $MNX(x)$ to get $mNX(x)$, and in $MXNX(x)$ to get $mXNX(x)$, we may not have both s and s' in $mNX(x)$ nor in $mXNX(x)$. Otherwise, if $s \in mNX(x) \subseteq MNXs(x)$, $s'[x\rangle$ and s' does not belong to $MNX(x)$. And if $s \in mXNX(x) \subseteq mXNXs(x)$, there is an x-free path (of length 1) from s to s', and s' may not belong to $MXNXs(x)$, hence to $mXNX(x)$.

As a consequence, in any case, we may not have two successive members in $mNX(x)$ (or $mNXs(x)$), nor two successive members in $mXNX(x)$ (or $mXNXs(x)$), and the sizes of those sets are bounded by $\lceil |S|/2 \rceil$. To see that this bound may be reached, let us consider for example the solution of a word $(ab)^n$ for any $n \in \mathbb{N}$ and $a \neq b$, as illustrated in Fig. 10. The corresponding transition system has $2 \cdot n + 1$ states, $2 \cdot n$ arcs, and 2 alternating labels. Then, $mNXs(a) = \{s_{2 \cdot n}\} \cup \{s_{2 \cdot i - 1} | i = 1, \ldots, n - 1\}$, $mXNXs(a) = \{s_{2 \cdot i - 1} | i = 1, \ldots, n\}$,

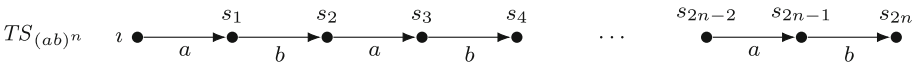

Fig. 10 Labelled transition system for a word $(ab)^n$

$mNXs(b) = \{s_{2\cdot i} | i = 0, \ldots, n\}$ and $mXNXs(b) = \{s_{2\cdot i} | i = 1, \ldots, n\}$. As a consequence, the sizes of those sets are either n ($= \lfloor |S|/2 \rfloor$) or $n + 1$ ($= \lceil |S|/2 \rceil$).

If we neglect the numbers of labels and arcs, we thus have in the worst case $O(|S|)$ systems of $O(|S|)$ inequalities to be solved. Since computing the solution of each such system has a complexity of $O(|S|^3)$ (see [33]), this leads to a complexity in $O(|S|^4)$ for our algorithm, to be compared to a complexity of about $O(|S|^6)$ for the general algorithm in [1] (the exact complexity of this algorithm is hard to obtain but its authors estimated it between $O(|S|^5)$ and $O(|S|^6)$). However, in practice and in particular in our experiments, some of which are presented in the next section, the number and size of the inequality systems to be solved are usually rather small and most of the execution time is spent in the construction of the sets $mNXs(x)$ and $mXNXs(x)$ (see Table 1), which is at most quadratic. When the pre-synthesis yields a negative result, we only have to consider the complexity of the latter since there is no proper synthesis. Again, in the worst case the complexity is then quadratic, but it may be much better if the culprit is found rapidly, hence it highly depends on the order in which checks are performed.

8 Experimental evaluation

The proposed algorithm was implemented and its performance was compared with existing implementations. Several classes of Petri nets were used. The reachability graph of such a Petri net was calculated and re-synthesised into a—possibly different—Petri net. The running time of the synthesis step was measured.

This section first introduces the Petri nets and the synthesis implementations that participated in the evaluation. Then, the results are presented and interpreted.

The size of a Petri net is controlled by a parameter n (and a parameter k for the marking on circles). The reader is referred to Fig. 11 for some examples.

- Connected bit nets: A bit can be in one of two states, *set* and *unset*. Initially, all bits are unset. A connected n-bit net has n such bits put next to each other. The lowest bit has a transition to set it. 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 $1 \leq i < n$. Such a Petri net is a connected marked graph and its reachability graph has 2^n reachable states (it is one of the many possible implementations of an n-bit buffer).
- Circles: A circle of size n has n places and n transitions forming a circle in which tokens are transported. An arbitrary place contains k tokens. It is a connected marked graph, hence also a choice-free net. Each distribution of the k tokens on the n places is a reachable marking, thus by standard combinatorics there are $\binom{n+k-1}{k} = \frac{(n+k-1)!}{(n-1)!k!}$ reachable markings. Since k will be fixed, this is in $\mathcal{O}(n^k)$.
- Pre-cubes: A (hyper-)cube of dimension n has n transitions t_i ($i = 1, \ldots, n$) that are concurrently enabled (once). In an n-dimensional pre-cube (inspired from Figure 16 in [4]), there is another transition tr which gets enabled after at least $n - 1$ transitions t_i fire. When all these $n + 1$ transitions fired, the initial marking is reached again. This kind of lts cannot be solved by a marked-graph, because the initial state is not backward

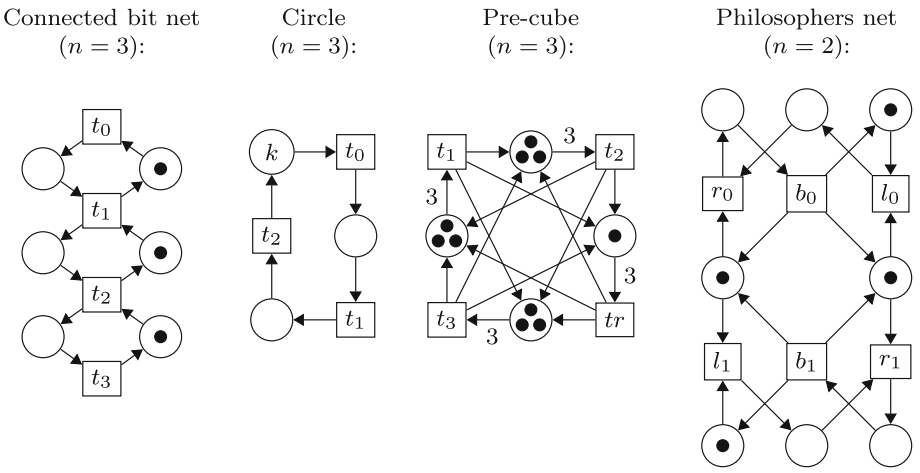

Fig. 11 Instances of the net classes. In the philosophers net, transitions l and r describe the taking of a left fork and of a right fork, respectively, while b describes putting two forks back simultaneously

persistent [14]. There are $2^n + n$ reachable markings: the n-dimensional cube produces 2^n markings and there are n additional markings reachable via tr when one of the n transitions t_i has not fired yet.
A possible choice-free Petri net generating this behaviour has $n + 1$ places, one for each transition. Each transition consumes n tokens from its place and produces one token on every other place. The places for transitions t_i each initially have n tokens, so they are enabled, while the place for transition tr has one token initially, so needs $n - 1$ of the t_i to fire before enabling tr.

- Philosophers net: This is Dijkstra's dining philosophers example [28] modelled as a Petri net. Each philosopher can be in one of three states: Thinking, waiting, or eating. When transiting to the waiting state, a philosopher grabs his left fork. To reach the eating state, he also needs his right fork. These forks are shared resources which each philosopher competes for with his neighbours. In the lts this leads to non-persistence, meaning that this behaviour cannot be generated by a choice-free Petri net (cf. Theorem 2). It also can deadlock. The state space has between 2^n and 3^n reachable markings. For the upper bound, assume that each philosopher can freely transition to one of his three states. For the lower bound, each philosopher can be either thinking or waiting and all such combinations are reachable.

Several algorithms for Petri net synthesis were considered in this benchmark.

- NEW: The algorithm introduced in the present paper, combining a pre-synthesis based on the algorithm in Fig. 6 and a proper synthesis based on the algorithm in Fig. 9, with the changes outlined in Sect. 7.1, because they lead to an overall faster algorithm in various experiments. Inequality systems are solved via the SMTInterpol library [21].
- Pre-synthesis: The runtime for the pre-synthesis algorithm in Fig. 6 was measured separately, so that its runtime can be compared with the full NEW algorithm.
- APT-CF: APT is a general toolbox for analysing Petri nets and transition systems [39]. It contains several Petri net synthesis implementations and tries to choose the best one for a given problem. APT-CF is based on a generic algorithm for the synthesis of P/T nets

from [1] with additional linear inequality systems to provide choice-free solutions. No optimisations are employed for the target class, contrary to NEW.

- APT-MG: The characterisation from [14] allows to synthesise connected marked graph Petri nets efficiently. This algorithm is implemented in APT. Every marked graph Petri net is also choice-free, so this is a more specialised algorithm compared to the algorithm introduced in the present paper.
- Synet [16] is a Petri net synthesis tool. For this paper, synet was always used with parameter `-r` telling it to use its improved algorithm. Synet synthesises optimised (in terms of size) general P/T Petri nets by default.
- Synet-CF: Synet supports *locations* on labels [5]. By giving each label a distinct location, a choice-free solution can be requested. Since this works as an extra layer on Synet, we may not expect better performances, but it compares more faithfully to NEW when a choice-free net is definitely sought for.
- Petrify [22–24] synthesises 1-bounded Petri nets, i.e., Petri nets where no place ever has more than one token on it. In case no 1-bounded solution exists, *label splitting* [17] is applied. This means that for inputs that are not solvable by 1-bounded Petri nets, Petrify actually solves a slightly different problem than the other tools in this benchmark. Petrify was always given the `-uc` parameter. This asks for a unique-choice [23] solution. In the presented examples, Petrify never becomes slower when this option is given. Also, this is the strongest restriction on choices that Petrify supports, which allows a better comparison with choice-free Petri nets.
- Genet [18,19] is a general tool supporting the synthesis of bounded Petri nets. Its algorithm is a generalisation of the Petrify approach that will produce a solution with the lowest possible bound. The algorithm is based on multisets of states that are used to compute places.

Only the NEW, APT-CF and Synet-CF implementations actually guarantee that a choice-free solution is produced whenever possible (and APT-MG, when a marked graph solution, which is also choice-free, is available). The other implementations compute general place-transition nets, or elementary nets in the case of Petrify. This means that we are comparing tools that target different classes of nets. The hope is to show that targeted synthesis, particularly the NEW algorithm for choice-free synthesis, is more efficient than general synthesis. Note, however, that, even if there is a choice-free (or marked graph) solution, non-targeted tools may produce completely different solutions.

The benchmarks were performed in Java. For the interaction with tools provided as external programs, a new process was spawned when needed. The NEW, APT-MG, and APT-CF implementations are written in Java and can be called directly. This means that for Synet, Synet-CF, Petrify, and Genet, (a small) additional overhead was incurred compared to the other implementations.

Figures 12, 13, 14, 15 and 16 show experimental results using semi-logarithmic plots.

8.1 Connected bit nets

Figure 12 presents the results for n-connected bit nets. It can be seen that the synet tool performs a lot worse than the other implementations. The proposed NEW algorithm performs better than all other implementations, except for the marked graph algorithm, which wins this benchmark (which is not too surprising since the latter is tailored for this case). However, for larger values the NEW algorithm is the fastest implementation, and it is about 20 times faster than the triple Genet/Petrify/APT-CF.

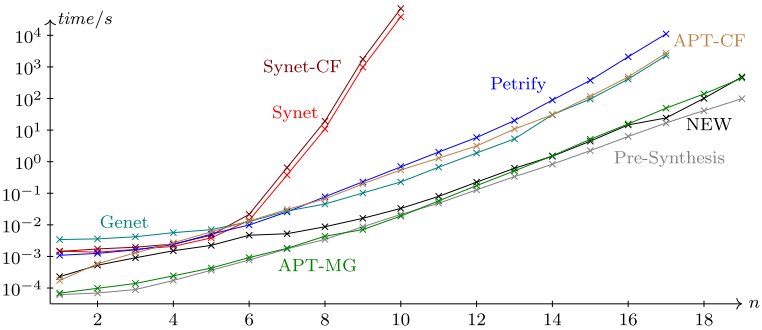

Fig. 12 Runtime for the synthesis of connected bit net with n bits

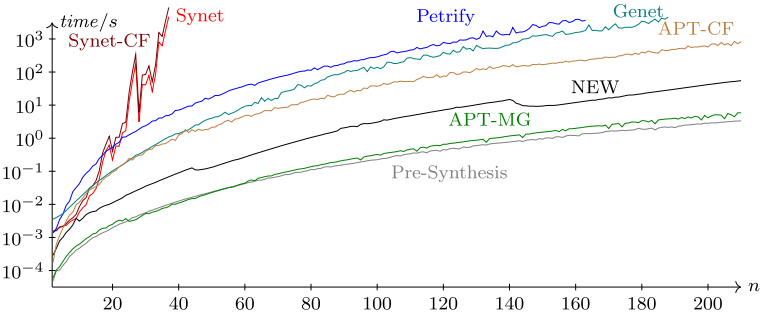

Fig. 13 Runtime for the synthesis of circles of size n with two tokens

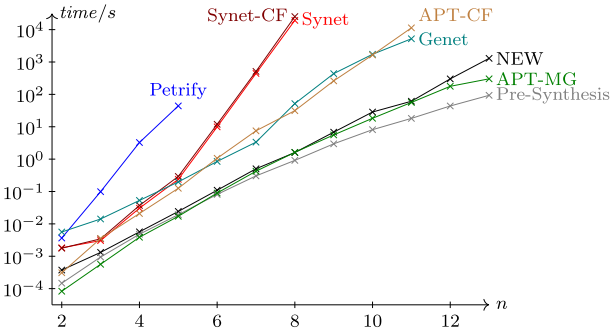

Fig. 14 Runtime for the synthesis of circles of size n with ten tokens

A closer examination of the behaviour of the NEW algorithm indicates that the pre-synthesis phase takes about half of the total execution time, but this number varies from 20% to 70% across the graph. This shows that even when this does not allow to stop the process before the proper-synthesis, the overhead is rather small (it can even be a lot smaller: see Table 1). Anyway, let us recall it computes the distances needed by the proper-synthesis.

8.2 Circles

The runtime for the synthesis of circle Petri nets with $k = 2$ tokens is shown in Figs. 13, and 14 shows the results for $k = 10$ tokens. Circles with $k = 1$ were not measured, because

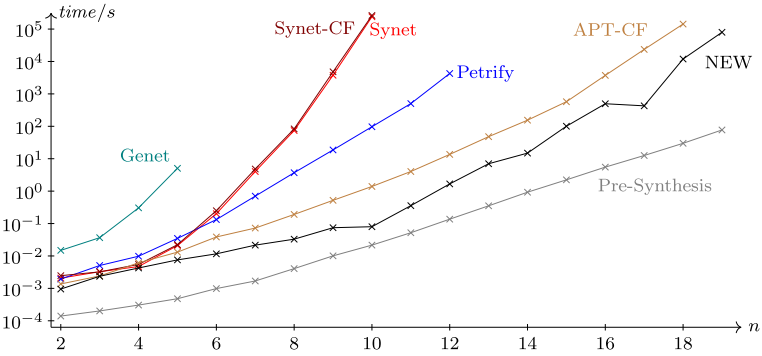

Fig. 15 Runtime for the synthesis of pre-cubes of size n

the time for their synthesis is dominated by input and output operations and not by the actual synthesis algorithm. This is less so for $k > 1$.

Petrify becomes slower with larger k, which is not surprising since Petrify specialises on 1-bounded Petri nets. Synet loses this benchmark by showing poor performance, but its performance comes closer to the other algorithms for larger k. The NEW synthesis shows a surprisingly non-monotonic behaviour for which we have no explanation for the moment. This behaviour is reproducible and seems not to be caused by measurement errors. Its runtime is smaller than the times of Genet and APT-CF. For $k = 2$, the marked graph algorithm clearly wins this benchmark. For $k = 10$, the NEW synthesis comes very close to it.

We can see that the pre-synthesis is very fast on this benchmark and only takes a fraction of the time of the complete NEW synthesis. In fact, for $k = 2$ pre-synthesis takes about 20% of the runtime of the NEW algorithm, but this value goes down to about 5% for larger values of n. For $k = 10$, pre-synthesis is initially responsible for about 80% of the time, but this goes down to about 15% for $n = 12$.

8.3 Pre-cube nets

Figure 15 shows the runtime for synthesising the pre-cube example. As already mentioned, APT-MG is not applicable on this example. We can see that all other implementations are slower than the NEW algorithm, which again shows some non-monotonic behaviour. The closest contester is APT-CF, which is however an order of magnitude slower most of the time. Also pre-synthesis is again very fast and is another order of magnitude faster than the complete NEW synthesis algorithm.

8.4 Philosophers nets

The philosophers nets produce the results from Fig. 16. Because each philosophers example produces a non-persistent reachability graph, by Theorem 2 this behaviour cannot be generated by a choice-free Petri net. This means that only implementations limited to choice-free Petri nets can fairly be compared, since otherwise we would compare the time until a failure is returned for a choice-free synthesis with the time for a successful general synthesis, which is unfair.

The graph shows that the proposed algorithm, which first checks determinism and persistence during the pre-synthesis and returns a negative result, is a lot faster (by about a factor 1000) than the existing algorithms, which actually attempt synthesis. Note however that it is

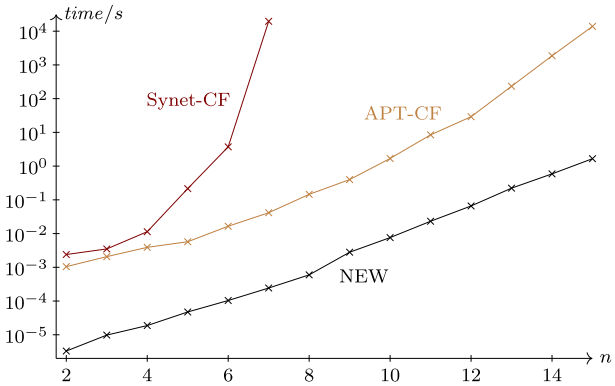

Fig. 16 Runtime for the (failing) synthesis of n philosophers

Table 1 Profiling the NEW synthesis of a connected bit net of size $n = 15$ and of a 2-token circle of size $n = 100$

Activity	Bit net (%)	Circle (%)
Reading the input	22.1	0.3
Checking determinism and persistence	3.5	0.0
Performing pre-synthesis	16.1	0.2
Computing the sets *mXNXs* etc.	54.7	98.9
Solving the remaining (indispensable) inequalities	2.1	0.5

difficult to grasp the performance for a failing synthesis, since this highly relies on the order in which the inequality systems or pre-conditions are considered.

8.5 A closer look at NEW

Finally, we profiled the NEW algorithm on a connected bit net of size $n = 15$ and a circle of size $n = 100$ containing two tokens. Table 1 shows the results. The bottleneck is clearly the computation of sets such as *mXNXs*, this being the price for reducing the number and size of linear systems to consider in the proper synthesis. Since this computation involves a lot of accesses to the data structures, there might be more potential in optimising the latter. However, this is beyond the scope of the present paper.

9 Concluding remarks

In this paper (which substantially extends [13] and complements [15]), we have described an efficient procedure for the synthesis of bounded, choice-free Petri nets from labelled transition systems. Its key properties and benefits are as follows.

- Structural knowledge about the target systems is embodied in an essential way in the algorithm.
- Splitting synthesis into two stages allows not only the application of such knowledge, but also the early pruning of unsuitable inputs, accompanied by meaningful error messages.

- Experiments show that the algorithm embodied in our new procedure is substantially faster than known ones.

The structural, two-phase, method we propose seems to be generalisable. For some subclasses, viz. for T-systems and marked graphs, we have a full state space characterisation, which means that a solution can be built during pre-synthesis, rendering a separate synthesis phase unnecessary. For other classes, Petri net theory offers a plenitude of necessary conditions for synthesisability, affording partial state space characterisations. All such necessary conditions are suitable candidates for being incorporated in pre-synthesis. It may be noted that *exact* state space characterisations are presently out of reach, even for the solvability of (linearly ordered) acyclic persistent labelled transition systems.

Our experimental evaluation, in fact, demonstrates that specialised Petri net synthesis algorithms can be a lot faster than general algorithms. Stronger restrictions on the Petri nets tend to allow more efficient algorithms, as is evident from the marked graph algorithm APT-MG winning on all benchmarks when there is a MG solution, followed by the proposed choice-free synthesis algorithm NEW. Besides the presented benchmarks, we also performed many other ones. Some of these examples were not solvable by marked graphs; then APT-MG was not applicable and the NEW algorithm from this paper was consistently faster than all other implementations. In the cases in which APT-MG was applicable, it was always the fastest implementation, closely followed by the NEW algorithm. Genet and APT-CF both came next. It was also shown that the pre-synthesis presented here allows to see very quickly that a full synthesis procedure will fail, in a fraction of the time needed by a full synthesis algorithm.

In future work, it would be interesting

- to refine the worst case complexity analysis of the algorithm described in this paper,[8] and to also consider the average case complexity, theoretically and/or experimentally;
- to explore whether the slack between the relatively small set of properties (2), needed for synthesis, and the full set of additional properties yielded by the pre-synthesis algorithm, can be exploited in order to further optimise its run-time;
- to investigate extensions of the method described in this paper to other target classes, as well as to other (not linear-algebra based) synthesis algorithms;
- to generalise the method to unbounded Petri nets, i.e., to infinite transition systems which may have been specified indirectly, e.g., by giving a Petri net for which we search for an equivalent (with an isomorphic reachability graph) choice-free net, or by giving a grammar for the language of the desired net;
- and to investigate other equivalences besides graph isomorphism, for example bisimilarity or language equivalence.

Acknowledgements We are indebted to the reviewers for valuable comments.

A Keller's theorem

The notion of a residual sequence, defined next, captures the subtraction of one sequence from another one "as much as possible".

[8] Note that this is likely to be very difficult; not even for the basic algorithm [1], the exact complexity seems to be fully investigated.

Definition 16 (*Residues*) Suppose that $\tau, \sigma \in T^*$. The *(left) residue of* τ *with respect to* σ, denoted by $\tau \stackrel{\bullet}{-} \sigma$, arises from cancelling successively in τ the leftmost occurrences of all symbols from σ, read from left to right. Inductively: $\tau \stackrel{\bullet}{-} \varepsilon = \tau$; $\tau \stackrel{\bullet}{-} t = \tau$ if $t \notin supp(\tau)$; $\tau \stackrel{\bullet}{-} t =$ the sequence obtained by erasing the leftmost t in τ if $t \in supp(\tau)$; and $\tau \stackrel{\bullet}{-} (t\sigma) = (\tau \stackrel{\bullet}{-} t) \stackrel{\bullet}{-} \sigma$.

For example, $acbcacbc \stackrel{\bullet}{-} abbcb = cacc$ and $abbcb \stackrel{\bullet}{-} acbcacbc = b$. In terms of Parikh vectors, Definition 16 directly yields the following consequence (where function min is extended componentwise to vectors):

Corollary 6 (Parikh vectors of residues) *For* $\tau, \sigma \in T^*$, $\Psi(\sigma) - \Psi(\sigma \stackrel{\bullet}{-} \tau) = \Psi(\tau) - \Psi(\tau \stackrel{\bullet}{-} \sigma) = \min(\Psi(\sigma), \Psi(\tau))$.

E.g., $\Psi(acbcacbc) - \Psi(acbcacbc \stackrel{\bullet}{-} abbcb) = \Psi(abbcb) - \Psi(abbcb \stackrel{\bullet}{-} acbcacbc) = \Psi(abbc)$.

Theorem 9 (Keller [32]) *Let* $(S, T, \rightarrow, \iota)$ *be a deterministic, persistent lts. Let* τ *and* σ *be two label sequences enabled at some state* s. *Then* $\tau(\sigma \stackrel{\bullet}{-} \tau)$ *and* $\sigma(\tau \stackrel{\bullet}{-} \sigma)$ *are also enabled at* s. *Furthermore, the state reached after* $\tau(\sigma \stackrel{\bullet}{-} \tau)$ *equals the state reached after* $\sigma(\tau \stackrel{\bullet}{-} \sigma)$.

Two nice little applications of this result are the following proofs.

Proof (of Proposition 1) Let $s, s', s'' \in S$ and $s[\alpha\rangle s' \wedge s[\alpha'\rangle s'' \wedge \Psi(\alpha) = \Psi(\alpha')$; we need to prove $s' = s''$ (see Definition 5). Theorem 9 (applicable by determinism and persistence) yields $s[\alpha\rangle s'[\alpha' \stackrel{\bullet}{-} \alpha\rangle q \wedge s[\alpha'\rangle s''[\alpha \stackrel{\bullet}{-} \alpha'\rangle q$. By $\Psi(\alpha) = \Psi(\alpha')$, both $\alpha \stackrel{\bullet}{-} \alpha'$ and $\alpha' \stackrel{\bullet}{-} \alpha$ equal the empty sequence ε, entailing $s' = q = s''$. □

Proof (of Lemma 1) By the premise, we have $r[\kappa\alpha\rangle s$ and $r[\alpha\rangle s$. Theorem 9 yields both $s[\alpha \stackrel{\bullet}{-} (\kappa\alpha)\rangle q$ and $s[(\kappa\alpha) \stackrel{\bullet}{-} \alpha\rangle q$, for some state q. By $\Psi(\alpha) \leq \Psi(\kappa\alpha)$, the sequence $\alpha \stackrel{\bullet}{-} (\kappa\alpha)$ is empty, and thus, $s = q$. Therefore, defining $\kappa' = (\kappa\alpha) \stackrel{\bullet}{-} \alpha$, we get $s[\kappa'\rangle s$. Clearly, $\Psi(\kappa') = \Psi(\kappa)$, ending the proof. □

B Proof of Proposition 4

($\Rightarrow$): Obvious since the distance-path property requires that we have distance-paths for any pair of states $s' \in [s\rangle$, hence also for $s' \in [\iota\rangle$.

($\Leftarrow$): We assume that for all $s \in S$, there is a path $\iota[\alpha\rangle s$ such that $\Psi(\alpha) = \Delta_s$, and we need to show that we have distance-paths for any pair of states $s' \in [s\rangle$, and not only when $s = \iota$.

Let $\mathcal{P} = \{\Upsilon_1, \ldots, \Upsilon_n\}$. We shall proceed by contradiction and assume that there are two states $s, s' \in S$ such that $s' \in [s\rangle$ and the distance between them is not realised. By $s' \in [s\rangle$, there is a short path $s[\gamma\rangle s'$, and because this path does not realise the distance, $\Psi(\gamma) \geq \Theta$ for at least one Parikh vector $\Theta \in \mathcal{P}$. We consider such a counterexample with the smallest possible γ over the entire lts (see Fig. 17).

Since a subpath of a short path is itself short, without loss of generality we may assume that for some labels $u, v \in T$, $\gamma = u\delta v$ with $\Psi(\gamma)(u) = \Theta(u)$, $\Psi(\gamma)(v) = \Theta(v)$ and, for no $\Theta' \in \mathcal{P}$, $\Psi(\gamma) \geq \Theta + \Theta'$ (otherwise, we can find a smaller counterexample by suppressing a prefix and/or a suffix).

We may assume that $\iota[\alpha\rangle s$ with $\Delta_s = \Psi(\alpha)$ and $\iota[\beta\rangle s'$ with $\Delta_{s'} = \Psi(\beta)$, and we know that $\Phi \not\leq \Delta_s$ and $\Phi \not\leq \Delta_{s'}$ for all $\Phi \in \mathcal{P}$, as well as that

$$\Delta_{s'} = \Psi(\alpha\gamma) \bmod \mathcal{P} = \Delta_s + \Psi(\gamma) - \Theta - \sum_{\Phi \in \mathcal{P}} k_\Phi \cdot \Phi \tag{12}$$

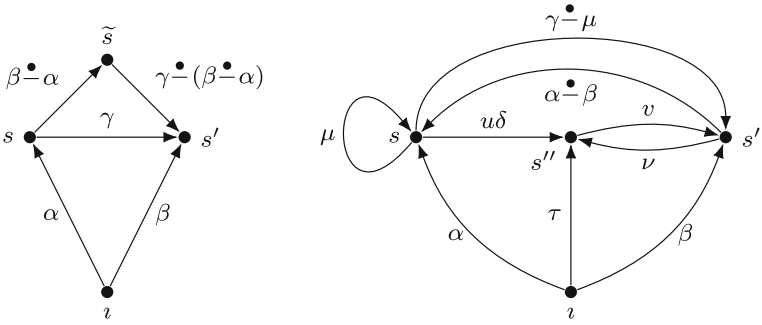

Fig. 17 We may have neither $\Delta_{s'} \not\leq \Delta_s$ (l.h.s.) nor $\Delta_{s'} \leq \Delta_s$ (r.h.s.)

for some natural numbers k_Φ. By $\Delta_s = \Psi(\alpha)$ and $\Delta_{s'} = \Psi(\beta)$, and by Corollary 6, this yields $\Psi(\beta \stackrel{\bullet}{-} \alpha) = \Psi(\alpha \stackrel{\bullet}{-} \beta) + \Psi(\gamma) - \Theta - \sum_{\Phi \in \mathcal{P}} k_\Phi \cdot \Phi$.

If $\Delta_{s'} \not\leq \Delta_s$, by Keller's theorem we have $s[\beta \stackrel{\bullet}{-} \alpha\rangle \tilde{s}$ for some state $\tilde{s} \in S$, $\beta \stackrel{\bullet}{-} \alpha$ being nonempty. Since $\Psi(\beta) \leq \Psi(\alpha) + \Psi(\gamma)$, we have that $\Psi(\beta \stackrel{\bullet}{-} \alpha) \leq \Psi(\gamma)$ and, by Keller's theorem again, $\tilde{s}[\gamma \stackrel{\bullet}{-} (\beta \stackrel{\bullet}{-} \alpha)\rangle s'$, where $\Psi((\beta \stackrel{\bullet}{-} \alpha)(\gamma \stackrel{\bullet}{-} (\beta \stackrel{\bullet}{-} \alpha))) = \Psi(\gamma)$, so that the path $s[\beta \stackrel{\bullet}{-} \alpha\rangle \tilde{s}[\gamma \stackrel{\bullet}{-} (\beta \stackrel{\bullet}{-} \alpha)\rangle s'$ is as short as γ, but it starts with labels from β not needed to get Θ, since $\Psi(\gamma \stackrel{\bullet}{-} (\beta \stackrel{\bullet}{-} \alpha)) = \Psi(\gamma) - \Psi(\beta \stackrel{\bullet}{-} \alpha) \geq \Theta$. As a consequence, we can construct a shorter counterexample by dropping $\beta \stackrel{\bullet}{-} \alpha$ in front of $\gamma \stackrel{\bullet}{-} (\beta \stackrel{\bullet}{-} \alpha)$ [see Fig. 17(l.h.s.)].

Thus we may assume that $\Delta_{s'} \leq \Delta_s$, and still by Keller's theorem we have that $s'[\alpha \stackrel{\bullet}{-} \beta\rangle s$. Now, let us consider the path $s[u\delta\rangle s''[v\rangle s'$. There is a path $\iota[\tau\rangle s''$ with

$$\Psi(\tau) = \Delta_{s''} = \Psi(\alpha u \delta) \bmod \mathcal{P} = \Delta_{s'} + \Theta - \Psi(v) \geq \Delta_s$$

The first and second equality come from the fact that τ realises the distance between ι and s''. The third equality is justified by first observing [from (12) by subtracting $\Psi(v)$ on both sides] that

$$\Psi(\alpha u \delta) = \Psi(\beta) + \Theta + \sum_{\Phi \in \mathcal{P}} k_\Phi \cdot \Phi - \Psi(v)$$

and then taking the modulo on both sides. Moreover, $\Delta_{s'} = \Psi(\tau v) \bmod \mathcal{P} \leq \Delta_{s''} + \Psi(v)$. Hence, with Keller again, there is a path $s'[v\rangle s''$ with $\Psi(v) = \Theta - \Psi(v)$, so that there is a loop around s' with Parikh vector Θ. Now, since loops can be transported Parikh-equivalently by Lemma 1, there is a loop around $s[\mu\rangle s$ with Parikh vector Θ too. And again by Keller's theorem, there is a path from $s[\gamma \stackrel{\bullet}{-} \mu\rangle s'$ from s to s' with Parikh vector $\Psi(\gamma) - \Theta$, contradicting the shortness of γ [see Fig. 17(r.h.s.)].

We can thus conclude that there is no counterexample to our claim, that each short path is a distance-path and that the distance-path property is valid for any pair of reachable states.

References

1. Badouel, É., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Texts in Theoretical Computer Science, p. 339. Springer, Berlin (2015). ISBN 978-3-662-47967-4
2. Badouel, É., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Mosses, P., Nielsen, M., Schwartzbach, M. (eds.) TAPSOFT 1995, Aarhus (Denmark). Lecture Notes in Computer Science, vol. 915, pp. 364–378. Springer, Berlin (1995)
3. Badouel, É., 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, É., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models. Lecture Notes in Computer Science, vol. 1491, pp. 529–586. Springer, Berlin (1999)
5. Badouel, É., Caillaud, B., Darondeau, P.: Distributing finite automata through Petri net synthesis. J. Form. Asp. Comput. **13**, 447–470 (2002)
6. Best, E., Darondeau, P.: A decomposition theorem for finite persistent transition systems. Acta Inf. **46**, 237–254 (2009)
7. Best, E., Darondeau, P.: Petri net distributability. In: Virbitskaite, I., Voronkov, A. (eds.) PSI'11, Novosibirsk, LNCS, vol. 7162, pp. 1–18. Springer, Berlin (2011)
8. Best, E., Erofeev, E., Schlachter, U., Wimmel, H.: Characterising Petri net solvable binary words. In: Moldt, D., Kordon, F. (eds.) Proc. 37th International Conference on Applications and Theory of Petri Nets and Concurrency, Toruń (Poland), Lecture Notes in Computer Science, vol. 9698, pp. 39–58. Springer, Berlin (2016)
9. Best, E., Devillers, R.: Synthesis of persistent systems. In: 35th International Conference on Application and Theory of Petri Nets and Concurrency (ICATPN 2014), pp. 111–129 (2014)
10. Best, E., Devillers, R.: Synthesis and reengineering of persistent systems. Acta Inf. **52**(1), 35–60 (2015)
11. Best, E., Devillers, R.: State space axioms for T-systems. Acta Inf. **52**(2–3), 133–152 (2015)
12. Best, E., Devillers, R.: Synthesis of live and bounded persistent systems. Fund. Inf. **140**, 39–59 (2015)
13. Best, E., Devillers, R.: Synthesis of bounded choice-free Petri nets. In: Aceto, L., Frutos Escrig, D. (eds.) Proc. 26th International Conference on Concurrency Theory (CONCUR 2015), LIPICS, pp. 128–141. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, Dagstuhl. https://doi.org/10.4230/LIPIcs.CONCUR.2015.128 (2015)
14. Best, E., Devillers, R.: Characterisation of the state spaces of marked graph Petri nets. Inf. Comput. **253**(Pt. 3), 399–410 (2017)
15. Best, E., Devillers, R.: Petri net pre-synthesis based on prime cycles and distance paths. To appear in Science of Computer Programming (2018). Also: Informatik-Bericht Nr. 3/16, Univ. Oldenburg, 26 pages (2016)
16. Caillaud, B.: Synet: un outil de synthèse de résaux de Petri bornés, applications. Research Report RR 3155, INRIA (1997). See also: https://hal.inria.fr/inria-00073534. http://www.irisa.fr/s4/tools/synet/
17. Carmona, J.: The label splitting problem. In: Jensen, K., Aalst, W.M.V.D., Ajmone-Marsan, M., Franceschinis, G., Kleijn, J., Kristensen, L.M. (eds.) Transactions on Petri Nets and Other Models of Concurrency VI, Lecture Notes in Computer Science, vol. 7400, pp. 1–23. Springer, Berlin (2012)
18. Carmona, J., Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: A symbolic algorithm for the synthesis of bounded Petri nets. In: van Hee, K., Valk, R. (eds.) Applications and Theory of Petri Nets 2008, LNCS, vol. 5062, pp. 92–111. Springer, Berlin (2008)
19. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded Petri nets. IEEE Trans. Comput. **59**(3), 371–384 (2010)
20. Commoner, F., Holt, A.W., Even, S., Pnueli, A.: Marked directed graphs. J. Comput. Syst. Sci. **5**(5), 511–523 (1971)
21. Christ, J., Hoenicke, J., Nutz, A.: SMTInterpol: an interpolating SMT solver. In: Donaldson, A., Parker, D. (eds.) Proc. of Model Checking Software, Oxford, LNCS, vol. 7385, pp. 248–254. Springer, Berlin (2012). See also: https://ultimate.informatik.uni-freiburg.de/smtinterpol/
22. 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)
23. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets for finite transition systems. IEEE Trans. Comput. **47**(8), 859–882 (1998)
24. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Logic Synthesis for Asynchronous Controllers and Interfaces, Volume 8 of Advanced Microelectronics. Springer Science & Business Media, Berlin (2012)
25. Crespi-Reghizzi, S., Mandrioli, D.: A decidability theorem for a class of vector-addition systems. Inf. Process. Lett. **3**(3), 78–80 (1975)
26. de San Pedro, J., Cortadella, J.: Mining structured Petri nets for the visualization of process behavior. In: 31st ACM Symposium on Applied Computing, pp. 839–846, Pisa (2016)
27. Desel, J., Esparza, J.: Free Choice Petri Nets, vol. 40, p. 242. Cambridge Tracts in Theoretical Computer Science, Cambridge (1995)
28. Dijkstra, E.W.: Hierarchical ordering of sequential processes. Acta Inf. **1**(2), 115–138 (1971)
29. Ehrenfeucht, A., Rozenberg, G.: Partial 2-structures, part I: basic notions and the representation problem, and part II: state spaces of concurrent systems. Acta Inf. **27**(4), 315–368 (1990)

30. Erofeev, E., Barylska, K., Mikulski, Ł., Piątkowski, M.: Generating all minimal Petri net unsolvable binary words. In: Proceedings of the Prague Stringology Conference, pp. 33–46 (2016). See http://www.stringology.org/event/
31. Hopkins, R.P.: Distributable nets. Applications and theory of Petri nets 1990. In: Rozenberg, G. (ed.) Advances of Petri Nets 1991, LNCS, vol. 524, pp. 161–187. Springer, Berlin (1991)
32. Keller, R.M.: A fundamental theorem of asynchronous parallel computation. In: Parallel Processing, LNCS, vol. 24, pp. 102–112. Springer, Berlin (1975)
33. Khachiyan, L.: Selected works. Moscow Center for Mathematical Continuous Education. ISBN 978-5-94057-509-2, 519 pages (2009) (**in Russian**)
34. Kondratyev, A., Cortadella, J., Kishinevsky, M., Pastor, E., Roig, O., Yakovlev, A.: Checking signal transition graph implementability by symbolic BDD traversal. In: Proc. European Design and Test Conference, pp. 325–332, Paris (1995)
35. Landweber, L.H., Robertson, E.L.: Properties of conflict-free and persistent Petri nets. JACM **25**(3), 352–364 (1978)
36. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**, 541–580 (1989)
37. Petri, C.A.: Concurrency. In: Brauer, W. (ed.) Proc. of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, LNCS, vol. 84, pp. 251–260. Springer, Berlin (1980)
38. Reisig, W.: Petri Nets. EATCS Monographs on Theoretical Computer Science, vol. 4. Springer, Berlin (1985)
39. Schlachter, U. et al.: https://github.com/CvO-Theory/apt (2013–2017)
40. 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)
41. van Glabbeek, R.J., Goltz, U., Schicke-Uffmann, J.-W.: On distributability of Petri nets—(extended abstract). In: Birkedal, L. (ed.) Proc. FoSSaCS 2012 (Held as Part of ETAPS), LNCS, vol. 7213, pp. 331–345. Springer, Berlin (2012)