Farid Ablayev
Ernst W. Mayr (Eds.)

LNCS 6072

Computer Science – Theory and Applications

5th International Computer Science Symposium in Russia, CSR 2010
Kazan, Russia, June 2010, Proceedings

Springer

Lecture Notes in Computer Science 6072

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

Farid Ablayev Ernst W. Mayr (Eds.)

Computer Science – Theory and Applications

5th International Computer Science Symposium
in Russia, CSR 2010
Kazan, Russia, June 16-20, 2010
Proceedings

Springer

Volume Editors

Farid Ablayev
Kazan State University
Dept. of Theoretical Cybernetics
420008 Kazan, Russia
E-mail: ablayev@ksu.ru

Ernst W. Mayr
Technische Universität München
Institut für Informatik
85748 Garching, Germany
E-mail: mayr@in.tum.de

Library of Congress Control Number: 2010927600

CR Subject Classification (1998): F.2, F.3, E.3, G.2, F.1, F.4

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-13181-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-13181-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The 5th International Computer Science Symposium in Russia (CSR 2010) was held June 16–20, 2010 in Kazan, Russia, hosted by the Institute of Informatics of the Tatarstan Academy of Sciences and the Kazan State University. It was the fifth event in the series of regular international meetings, following CSR 2006 in St. Petersburg, CSR 2007 in Ekaterinburg, CSR 2008 in Moscow, and CSR 2009 in Novosibirsk.

The opening lecture was given by Alexander Razborov, and seven more invited plenary lectures were given by Susanne Albers, Fedor Fomin, Juraj Hromkovič, Richard Jozsa, Prabhakar Raghavan, Miklos Santha, and Uwe Schöning.

This volume contains all the accepted papers and, at varying detail, the abstracts or extended abstracts of the invited talks. The scope of the proposed topics for the symposium was quite broad and covered basically all areas of the foundations of (meaning: theoretical) computer science. Unlike in previous years, no special application track was scheduled. We received 62 valid submissions in total, and out of these the Program Committee selected 30 for acceptance.

As in previous years, Yandex provided the Best Paper Awards; the recipients of these awards were, as selected by the Program Committee:

- Best Paper Award:
  Dmitry Itsykson: "Lower Bound on Average-Case Complexity of Inversion of Goldreich's Function by Drunken Backtracking Algorithms"
- Best Student Paper Award:
  Yu Junhua: "Prehistoric Phenomena and Self-Referentiality"

The reviewing process was organized using the EasyChair conference system, created by Andrei Voronkov, and we would like to acknowledge that this system very much helped to improve the efficiency of the committee work (and, of course, made a physical meeting of the PC unnecessary, a fact certainly advantageous in a number of respects, and at the same time not in others).

The following satellite events were collocated with CSR 2010:

1. Workshop on Program Semantics, Specification and Verification: Theory and Applications (PSSV 2010)
2. International Workshop on High Productivity Computations (HPC 2010)

We are very grateful to our sponsors:

- Russian Foundation for Basic Research
- Yandex (the largest Russian Internet portal providing key Web services)

We would also like to thank all the local organizers, among them in particular Alexander Vasiliev, Dina Mingazova, Rauf Ahtyamov, Ramil Garaev, Mikhail Abramsky, Mansur Ziyatdinov, and Regina Kozlova.

June 2010

Farid Ablayev
Ernst W. Mayr

Organization

CSR 2010 was organized jointly by the Institute of Informatics of the Tatarstan Academy of Sciences, and by Kazan State University.

Program Committee Chair

Ernst W. Mayr	Munich

Program Committee

Sergei N. Artemov	City University of New York, USA
Lev Beklemishev	Steklov Mathematical Institute, Moscow, Russia
Michael Ben-Or	Hebrew University, Israel
Harry Buhrman	University of Amsterdam, The Netherlands
Edith Cohen	AT&T Research, USA
Samir Datta	Chennai Mathematical Institute, India
Andrew V. Goldberg	Microsoft Research, USA
Dima Grigoriev	Université de Lille, France
Martin Hofmann	LMU München, Germany
Stasys Jukna	Universität Frankfurt, Germany
Yuri Matiyasevich	Steklov Institute of Mathematics, St. Petersburg, Russia
Peter Bro Miltersen	Aarhus University, Denmark
Georg Moser	University of Innsbruck, Austria
Madhavan Mukund	Chennai Mathematical Institute, India
Harald Räcke	University of Warwick, UK
Uwe Schöning	Universität Ulm, Germany
Jeffrey Shallit	University of Waterloo, Canada
Alexander Shen	LIF Marseille, France
Alberto Marchetti-Spaccamela	Università di Roma "La Sapienza", Italy
Michael Tautschnig	TU Darmstadt, Germany
Pascal Tesson	Université Laval, Canada
Berthold Vöcking	RWTH Aachen, Germany
Sergey Yekhanin	Microsoft Research, USA
Alexander Zelikovsky	Georgia State University, USA

Symposium Chair

Farid Ablayev	Kazan State University

CSR Steering Committee

Volker Diekert
Anna Frid
Edward A. Hirsch
Juhani Karhumäki
Mikhail Volkov

External Reviewers

Martin Avanzini
Haris Aziz
Maxim Babenko
Ulrich Berger
Lennart Beringer
Sam Buss
Albertas Caplinskas
Vincenzo Ciancia
Tristan Crolard
Bireswar Das
Eugeny Dashkov
Martin Dietzfelbinger
Alexander Dikovsky
Melvin Fitting
Enrico Formenti
Tim Furche
Evan Goris
Nick Gravin
Andreas Holzer
Rosalie Iemhoff
Dmitry Karpov
Joost-Pieter Katoen
Johannes Kinder
Alexander Knapp
Roman Kontchakov
Nagarajan Krishnamurthy
Vladimir Krupski
Andrey Kudinov
Petr Kurka
Roman Kuznets
Christof Löding
Martin Lange
Arjen Lenstra
Ming Li
Yury Lifshits
Stefan Lucks
Frank McSherry
Ilya Mezhirov
Ilya Mironov
Daniil Musatov
K. Narayan Kumar
Prajakta Nimbhorkar
Michael Nüsken
Dmitrii Pasechnik
Alexei Pastor
Ruzica Piskac
Vladimir Podolskii
Ilia Ponomarenko
Alexander Razborov
Andrei Romashchenko
Sasanka Roy
Martin Sauerhoff
Andreas Schnabl
Igor Shparlinski
San Skulrattanakulchai
Arne Storjohann
Thomas Streicher
S.P. Suresh
Alexei Talambutsa
Michael Ummels
H. Venkateswaran
Nikolai Vorobjov
Johannes Waldmann
Vladimir Zakharov
Harald Zankl
Florian Zuleger

Table of Contents

Algorithms for Energy Management (Invited Talk) 1
Susanne Albers

Sofic and Almost of Finite Type Tree-Shifts 12
Nathalie Aubrun and Marie-Pierre Béal

Proof-Based Design of Security Protocols 25
Nazim Benaissa and Dominique Méry

Approximating the Minimum Length of Synchronizing Words is Hard 37
Mikhail V. Berlinkov

Realizability of Dynamic MSC Languages 48
Benedikt Bollig and Loïc Hélouët

The MAX QUASI-INDEPENDENT SET Problem 60
N. Bourgeois, A. Giannakos, G. Lucarelli, I. Milis, V. Th. Paschos, and O. Pottié

Equilibria in Quantitative Reachability Games 72
Thomas Brihaye, Véronique Bruyère, and Julie De Pril

Quotient Complexity of Closed Languages 84
Janusz Brzozowski, Galina Jirásková, and Chenglong Zou

Right-Sequential Functions on Infinite Words 96
Olivier Carton

Kernelization (Invited Talk) 107
Fedor V. Fomin

Zigzags in Turing Machines 109
Anahí Gajardo and Pierre Guillon

Frameworks for Logically Classifying Polynomial-Time Optimisation Problems 120
James Gate and Iain A. Stewart

Validating the Knuth-Morris-Pratt Failure Function, Fast and Online 132
Paweł Gawrychowski, Artur Jeż, and Łukasz Jeż

Identical Relations in Symmetric Groups and Separating Words with Reversible Automata 144
R.A. Gimadeev and M.N. Vyalyi

Time Optimal d-List Colouring of a Graph 156
Nick Gravin

The Cantor Space as a Generic Model of Topologically Presented Knowledge 169
Bernhard Heinemann

Algorithmics – Is There Hope for a Unified Theory? (Invited Talk) 181
Juraj Hromkovič

Classifying Rankwidth k-DH-Graphs 195
Ling-Ju Hung and Ton Kloks

Lower Bound on Average-Case Complexity of Inversion of Goldreich's Function by Drunken Backtracking Algorithms 204
Dmitry Itsykson

A SAT Based Effective Algorithm for the Directed Hamiltonian Cycle Problem 216
Gerold Jäger and Weixiong Zhang

Balancing Bounded Treewidth Circuits 228
Maurice Jansen and Jayalal Sarma M.N.

Obtaining Online Ecological Colourings by Generalizing First-Fit 240
Matthew Johnson, Viresh Patel, Daniël Paulusma, and Théophile Trunck

Classical Simulation and Complexity of Quantum Computations (Invited Talk) 252
Richard Jozsa

Prefix-Free and Prefix-Correct Complexities with Compound Conditions 259
Elena Kalinina

Monotone Complexity of a Pair 266
Pavel Karpovich

Symbolic Models for Single-Conclusion Proof Logics 276
Vladimir N. Krupski

Complexity of Problems Concerning Carefully Synchronizing Words for PFA and Directing Words for NFA 288
P.V. Martyugin

Advancing Matrix Computations with Randomized Preprocessing 303
Victor Y. Pan, Guoliang Qian, and Ai-Long Zheng

Transfinite Sequences of Constructive Predicate Logics 315
Valery Plisko

The Quantitative Analysis of User Behavior Online — Data, Models and Algorithms (Abstract of Invited Talk) 327
Prabhakar Raghavan

A Faster Exact Algorithm for the Directed Maximum Leaf Spanning Tree Problem 328
Daniel Binkele-Raible and Henning Fernau

Complexity of Propositional Proofs (Invited Talk) 340
Alexander Razborov

Quantization of Random Walks: Search Algorithms and Hitting Time (Extended Abstract of Invited Talk) 343
Miklos Santha

Comparing Two Stochastic Local Search Algorithms for Constraint Satisfaction Problems (Invited Talk) 344
Uwe Schöning

Growth of Power-Free Languages over Large Alphabets 350
Arseny M. Shur

A Partially Synchronizing Coloring 362
Avraham N. Trahtman

An Encoding Invariant Version of Polynomial Time Computable Distributions 371
Nikolay Vereshchagin

Prehistoric Phenomena and Self-referentiality 384
Junhua Yu

Author Index 397

Algorithms for Energy Management*
(Invited Talk)

Susanne Albers

Department of Computer Science, Humboldt University Berlin
Unter den Linden 6, 10099 Berlin, Germany
albers@informatik.hu-berlin.de

Abstract. This article surveys algorithmic techniques to save energy in computing environments. More specifically, we address power-down mechanisms as well as dynamic speed scaling in modern microprocessors. We study basic problems and present important results developed over the past years.

1 Introduction

Energy has become a scarce and expensive resource. This also holds true for many computing environments. In data and computer centers energy costs often emerge as the second highest operating cost behind labor. Portable devices, such as laptops or mobile phones, operate on batteries of limited capacity and rely on effective energy management. Autonomous distributed systems such as sensor networks are even more sensitive in this respect as the charging of batteries is difficult or even impossible.

As a result energy has become a leading design constraint for computing devices. At the same time, the past years have witnessed considerable research interest in the design and analysis of algorithmic techniques to save energy. *Energy-efficient algorithms* reduce energy consumption while minimizing compromise to service. The studies focus mostly on the system and device level: How can we save energy in a single computational device? Two fundamental and effective techniques have been investigated extensively.

1. *Power-down mechanisms:* Whenever a system is idle, it can be transitioned to lower power stand-by or sleep modes. While a power-down operation is usually inexpensive, a subsequent wake-up operation incurs a significant amount of energy. The goal is to find state transition schedules minimizing energy consumption over a given time horizon.
2. *Dynamic speed scaling:* Many modern microprocessors are able to operate at variable speed/frequency. High speed levels result in high performance but also high energy consumption. The goal is to use the full speed/frequency spectrum of a processor and to apply low speeds whenever possible.

* Work supported by a Gottfried Wilhelm Leibniz Award of the German Research Foundation.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 1–11, 2010.

In this survey we study both techniques and review important results that have been developed in the algorithms community. A key aspect is that the corresponding solutions achieve a provably good performance. We remark that both of the above techniques have also been investigated in the systems community. The studies present algorithmic approaches but usually do not prove performance guarantees.

2 Power-Down Mechanisms

Power-down mechanisms are well-known and effective techniques to save energy. We encounter them on an every day basis: The desktop of a desktop turns off after some period of inactivity. A laptop transitions to a standby or hibernate mode if it has been idle for some time.

2.1 Problem Setting

We study a very general scenario where a given device can reside in one of several states. In addition to the active state, the device may be equipped with several low-power states such as stand-by, suspend, sleep and full-off modes. Specifications of such systems are given, for instance, in the Advanced Configuration and Power Management Interface (ACPI) that establishes industry-standard interfaces enabling power management and thermal management of mobile, desktop and server platforms; see [1] for more details. The various states have individual power consumption rates. Moreover state transitions incur cost. While the energy needed to move from a higher-power to a lower-power state is usually negligible, a reverse power-up operation consumes a significant amount of energy.

Over time, the device experiences an alternating sequence of active and idle periods. During an active period the device must reside in the active mode to execute jobs or provide a certain service. During an idle period the device may be transitioned to lower-power states. An algorithm has to decide when to perform transitions and to which states to move. Obviously, at the end of an idle period, the device must be transitioned back to the active state. The goal is to minimize the total energy consumption. As the energy consumption during the active periods is fixed, assuming that prescribed tasks have to be performed, we concentrate on energy optimization during the idle periods. In fact, we wish to design algorithms that, for any given idle period, minimize the energy consumption.

In practical applications, the power management problem described above typically is an *online problem*, i.e. the length T of an idle period is not known in advance. An online algorithm only learns about T when the period ends. Despite the handicap of not knowing the future, an online algorithm should achieve a provably good performance. Here one resorts to *competitive analysis* [21] that compares an online algorithm A to an optimal offline algorithm OPT. An optimal offline algorithm knows the length T in advance and can compute an optimal

state transition schedule minimizing the total energy consumption. Online algorithm A is called c-competitive if, for any idle period, the energy consumed by A is at most c times that of OPT.

Formally, suppose that we are given a device with ℓ states $s_1, \ldots, s_\ell$. Let r_i denote the power consumption rate, measured in energy units per time unit, of s_i. We number the states such that $r_1 > \ldots > r_\ell$. Hence s_1 is the active state and s_ℓ represents the state with lowest energy consumption. Moreover, let β_{ij}, $1 \leq i, j \leq \ell$, be the energy required to transition the system from state s_i to state s_j. Obviously, $\beta_{ii} = 0$, for any $1 \leq i \leq \ell$. We consider arbitrary state transition matrices where also power-down operations may incur energy.

We remark that if $\ell = 2$, i.e. the system has an active mode and a sleep mode only, the power management problem can be reduced to the *ski rental problem*, a basic problem in the theory of online algorithms, see e.g. [12]. As a result we can derive a 2-competitive deterministic online algorithm. The factor of 2 is the smallest competitive ratio that can be achieved by deterministic strategies. The best competitiveness of randomized online algorithms is equal to $e/(e-1) \approx 1.58$, where e is the Eulerian number, see [14]. Karlin et al. [14] also presented results for the case that the input, i.e. the length T of an idle period, is generated by a probability distribution.

In the following we concentrate on the general scenario that the given device is equipped with an arbitrary number ℓ of states. We present results that were developed by Augustine et al. [3] and Irani et al. [13].

2.2 Online and Offline Algorithms

Given a device with ℓ states, Augustine et al. [3] first develop an optimal offline algorithm that gives insight how to design good online strategies. They start with an observation that simplifies the state transition matrix $(\beta_{ij})_{1 \leq i,j \leq \ell}$. More specifically, one may assume without loss of generality that the power-up transition energies are zero. If this is not the case, one can define a new system in which the cost of transitioning from state s_i to s_j, where $i < j$, is $\beta_{ij} + d_{j0} - \beta_{i0}$ and the energy of moving from s_j to s_i is 0. A second observation is that during an idle period a system never powers up to an intermediate state s_i with $i > 1$. If the system powers up, then it does so at the end of the idle period to transition back to the active mode s_1. In the sequel, let $\beta_i = \beta_{1i}$ be the transition energy between s_1 and s_i.

Using the above facts, Augustine et al. presented an elegant formulation of an optimal offline strategy OPT. The total energy incurred by OPT in an idle period of length T is given by

$$OPT(T) = \min_{1 \leq i \leq \ell} \{\beta_i + r_i T\}.$$

Hence, OPT chooses the state that yields the smallest total cost consisting of state transition energy and energy consumption. Interestingly, the optimal cost has a simple graphical representation, see Figure 1. If we consider all linear

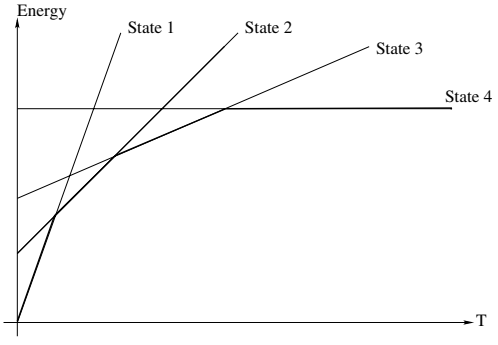


Fig. 1. The optimum cost in a four-state system

functions $f_i(t) = \beta_i + r_i t$, then the optimum energy consumption is given by the lower envelope of the arrangement of lines.

One can use this lower envelope to guide an online algorithm which state to use at any time. Let $S_{OPT}(t)$ denote the state used by OPT in an idle period of total length t, i.e. $S_{OPT}(t)$ is the state $\arg\min_{1 \le i \le \ell}\{\beta_i + r_i t\}$. Augustine et al. [13] propose an online algorithm that traverses the state sequence as suggested by the optimum offline algorithm, i.e. at any time t the algorithm uses state $S_{OPT}(t)$. This strategy works well if the state transition energies are well separated and satisfy $\beta_i \ge \gamma \beta_{i-1}$, for $1 < i \le \ell$ and $\gamma = 1 + 1/\sqrt{2}$. In this case the strategy achieves a competitive ratio of $2 + \sqrt{2} \approx 3.41$.

If the state transition energies are not well separated, one can work with a modified algorithm *OPT'* that restricts itself to states satisfying the desired constraint. The state set $\mathcal{S}'$ used by *OPT'* can be constructed as follows. Initially, $\mathcal{S}' = \{s_\ell\}$. Consider the system states in order of decreasing index and suppose that s_i was last added to $\mathcal{S}'$. Determine the largest j, where $1 \le j < i$, such that $\gamma \beta_j \le \beta_i$ and add s_j to $\mathcal{S}'$. Note that s_1 is finally added to $\mathcal{S}'$ because $0 = \gamma \beta_1 \le \beta_i$, for any i.

The algorithm by Augustine et al. traverses the state sequence of *OPT'*.

Algorithm Lower Envelope: In an idle period, at any time t, use state $S_{OPT'}(t)$.

Theorem 1. [3] *Lower-Envelope achieves a competitive ratio of* $3+2\sqrt{2} \approx 5.82$.

It is worth noting that *Lower Envelope* is 2-competitive if the state transition matrix is *additive*, i.e. if $\beta_{ik} = \beta_{ij} + \beta_{jk}$ for any $i < j < k$, see [13].

The above competitiveness of $3+2\sqrt{2}$ holds for any system. Augustine et al. [3] showed that, interestingly, better competitive ratios can be obtained for specific systems. More precisely, they gave a deterministic algorithm that achieves a competitive ratio of $c^* + \epsilon$, where c^* is best ratio possible for the given system. The key ingredient of the algorithm is a strategy that decides, for a fixed c, if a c-competitive algorithm exists. To this end the strategy enumerates all possible subsets of states and checks, for a given subset, if there exists a state transition

schedule whose energy consumption is always upper bounded by c time the optimum consumption. The best choice of c can then be approximated using binary search in the interval $[1, 3 + 2\sqrt{2}]$. The scheme described so far has a running time that is exponential in ℓ. Augustine et al. [3] showed that the time can be reduced to $O(\ell^2 \log \ell \log(1/\epsilon))$.

2.3 A Probabilistic Setting

The online algorithms presented in the last section achieve a provably good performance for any idle period whose length might even be generated by an adversary. From a practical point of view, this worst-case scenario is a bit pessimistic and it is reasonable to also study a stochastic setting where the length of idle periods is governed by probability distributions. In concrete applications, short periods might occur more frequently. Probability distributions can also model specific situations where either very short or very long idle periods are more likely to occur, compared to periods of medium length. Of course, such a probability distribution may not be known in advance but can be learned over time.

Let $Q = (q(T))_{0 \le T < \infty}$ be a fixed probability distribution on the length T of idle periods. Inspired by work of Karlin et al. [14], Irani et al. [13] and Augustine et al. [3] gave algorithms for this probabilistic setting. For simplicity we assume here that the state transition matrix is additive and use β_i to denote the transition energy between s_1 and s_i. The results can be extended to arbitrary state transition matrices [3].

Irani et al. [13] determined the time t_i when an online strategy should move from state s_{i-1} to s_i, $2 \le i \le \ell$. Let t_i be the time t that minimizes

$$\int_0^t r_{i-1} T q(T) dT + \int_t^\infty (r_{i-1} t + (T - t) r_i + \beta_i - \beta_{i-1}) q(T) dT.$$

Intuitively, the above expression is the expected cost of a deterministic algorithm ALG_t that powers down after t time units, assuming that only states s_{i-1} and s_i are available.

Algorithm ALG-P(ℓ): Change states at the transition times $t_2, \ldots, t_\ell$ defined above.

Theorem 2. [13] *For any fixed probability distribution Q, the expected energy consumption of ALG-P(ℓ) is at most $\frac{e}{e-1}$ times the expected optimum consumption.*

Irani et al. [13] presented an approach how to learn an initially unknown distribution Q. They combined the approach with *ALG-P(ℓ)* and performed experimental tests for an IBM mobile hard drive with four power states. It shows that the combined scheme achieves low energy consumption close to the optimum consumption.

Augustine et al. [3] extended the results to arbitrary state transition matrices. They determined optimal transition times t_{ij} when to move from s_i to s_j, assuming that only these two states are available. Based on these times the authors derived an optimal state sequence to be visited by the system.

3 Dynamic Speed Scaling

Many modern microprocessors allow the frequency/speed of the processor to be changed dynamically. Examples are the Intel SpeedStep or the AMD processor PowerNow. High speed levels yield high performance but consume significant amounts of energy. Low speed levels save energy but provide low performance only. The well-known cube-root rule for CMOS devices states that the speed s of a device is proportional to the cube-root of the power or, equivalently, that the power is proportional to s^3. The algorithms literature considers a generalization of this rule. If a processor runs at speed s, then the required power is s^α, where $\alpha > 1$ is a constant. Obviously, energy consumption is power integrated over time.

Dynamic speed scaling gives rise to many new challenging optimization problems. Most of them turn out to be scheduling problems. A scheduler, at any time, has to decide not only which job the execute on the processor but also which speed to use. The goal is to construct schedules that minimize energy consumption and satisfy additional constraints: The schedule has to be feasible and/or guarantee a certain quality of service. The past years have witnessed considerable research interest in dynamic speed scaling problems. In a seminal paper, initiating the algorithmic study of speed scaling, Yao, Demers and Shenker [22] investigated a scheduling problem with hard job deadlines. It is by far the most extensively studied speed scaling problem and we will concentrate on it in this survey.

Consider n jobs $J_1, \ldots J_n$ that have to be processed on a variable speed processor. Each job J_i is specified by an release time r_i, a deadline d_i and a processing volume w_i. The release time and the deadline specify the time interval $[r_i, d_i]$ during which the job must be executed. The job may not be started before r_i and must be finished until d_i. The processing volume w_i is the amount of work that must be completed to finish the job. Intuitively w_i can be viewed as the total number of CPU cycles required by the job. The processing time of the job depends on the processor speed. If J_i is executed at speed s, then it takes w_i/s time units to finish the task. Preemption of jobs is allowed, i.e. the processing of a job may be stopped and resumed later. The goal is to construct a feasible schedule minimizing the total energy consumption.

Yao, Demers and Shenker [22] make two simplifying assumptions. (1) There is no upper bound on the allowed processor speed. Hence a feasible schedule always exists. (2) The processor has a continuous spectrum of speed. In the following we will present algorithm for this enhanced model. Then we will discuss how to relax the assumptions.

3.1 Online and Offline Algorithms

Yao et al. [22] developed elegant online and offline algorithms. We first present the offline strategy, which knows all the jobs along with their characteristics in advance. The algorithm is known as *YDS*, referring to the initials of the authors. Algorithm *YDS* computes a minimum energy schedule for a given job set in a

series of rounds. In each round the algorithm identifies an interval of maximum density and computes a corresponding partial schedule for that interval. The *density* Δ_I of a time interval $I = [t, t']$ is the total processing volume to be completed in I divided by the length of I. More formally, let S_I be the set of jobs J_i that must be processed in I, i.e. that satisfy $[r_i, d_i] \subseteq I$. Then

$$\Delta_I = \frac{1}{|I|} \sum_{J_i \in S_I} w_i.$$

Intuitively, Δ_I is the minimum average speed necessary to complete all jobs that must be scheduled in I.

In each round, *YDS* determines the interval I of maximum density. In I the algorithm schedules the jobs of S_I at speed Δ_I according to *Earliest Deadline First (EDF)*. The *EDF* policy always processes the job having the earliest deadline among the available unfinished jobs. Then *YDS* removes the set S_I as well as the time interval I from the problem instance. More specifically, for any unscheduled job J_i with $d_i \in I$, the new deadline time is set to $d_i := t$. For any unscheduled J_i with $r_i \in I$, the new release time is $r_i := t'$. Time interval I is discarded. A summary of *YDS* in pseudo-code is given below.

Algorithm YDS: Initially $\mathcal{J} := \{J_1, \ldots, J_n\}$. While $\mathcal{J} \neq \emptyset$, execute the following two steps. (1) Determine the interval I of maximum density. In I process the jobs of S_I at speed Δ_I according to *EDF*. (2) Set $\mathcal{J} := \mathcal{J} \setminus S_I$. Remove I from the time horizon and update the release times and deadlines of unscheduled jobs accordingly.

The algorithm computes optimal schedules.

Theorem 3. [22] *For any job instance, YDS computes an optimal schedule minimizing the total energy consumption.*

Obviously, the running time of *YDS* is polynomial. When identifying intervals of maximum density, the algorithm only has to consider intervals whose boundaries are equal to the release times and deadlines of the jobs. Hence, a straightforward implementation of the algorithm has a running time of $O(n^3)$. Li et al. [18] showed that the time can be reduced to $O(n^2 \log n)$. Further improvements are possible if the job execution intervals form a tree structure [16].

In the online version of the problem, the jobs $J_1, \ldots, J_n$ arrive over time. A job J_i becomes known only at its arrival time r_i. At that time the deadline d_i and the processing volume w_i are also revealed. An online algorithm A is called c-competitive if, for any job sequence, the total energy consumption of A is at most c times that of an optimal offline algorithm OPT.

Yao et al. [22] devised two online algorithms, called *Average Rate* and *Optimal Available*. For any incoming job J_i, *Average Rate* considers the *density* $\delta_i = w_i/(d_i - r_i)$, which is the minimum average speed necessary to complete the job in time if no other jobs were present. At any time t the speed $s(t)$ is set to the accumulated density of jobs active at time t. A job J_i is *active at time* t if $t \in [r_i, d_i]$. Available jobs are scheduled according to the *EDF* policy.

Algorithm Average Rate: At any time t the processor uses a speed of $s(t) = \sum_{J_i:t\in[r_i,d_i]} \delta_i$. Available unfinished jobs are scheduled using *EDF*.

Yao et al. [22] proved an upper bound on the competitiveness.

Theorem 4. [22] *The competitive ratio of Average Rate is at most* $2^{\alpha-1}\alpha^\alpha$*, for any* $\alpha \geq 2$.

Bansal et al. [4] demonstrated that the analysis is essentially tight by giving a nearly matching lower bound.

Theorem 5. [4] *The competitive ratio of Average Rate is at least* $((2-\delta)\alpha)^\alpha/2$*, where* δ *is a function of* α *that approaches zero as* α *tends to infinity.*

The second strategy *Optimal Available* is computationally more expensive than *Average Rate*. It always computes an optimal schedule for the currently available work load. This can be done using *YDS*.

Algorithm Optimal Available: Whenever a new job arrives, compute an optimal schedule for the currently available unfinished jobs.

Bansal, Kimbrel and Pruhs [7] analyzed the above algorithm and proved the following result.

Theorem 6. [7] *The competitive ratio of Optimal Available is exactly* α^α.

The above theorem implies that in terms of competitiveness, *Optimal Available* is better than *Average Rate*. Bansal et al. [7] also developed a new online algorithm, called *BKP* according to the initials of the authors, that approximates the optimal speeds of *YDS* by considering interval densities. For times t, t_1 and t_2 with $t_1 < t \leq t_2$, let $w(t, t_1, t_2)$ be the total processing volume of jobs that are active at time t, have a release time of at least t_1 and a deadline of at most t_2.

Algorithm BKP: At any time t use a speed of

$$s(t) = \max_{t'>t} \frac{w(t, et-(e-1)t', t')}{t'-t}.$$

Available unfinished jobs are processed using *EDF*.

Theorem 7. [7] *Algorithm BKP achieves a competitive ratio of* $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$.

For large values of α, the competitiveness of *BKP* is better than that of *Optimal Available*.

All the above online algorithms attain constant competitive ratios that depend on α but no other other problem parameter. The dependence on α is exponential. For small values of α, which occur in practice, the competitive ratios are reasonably small. Moreover, a result by Bansal et al. [7] implies that the exponential dependence on α is inherent to the problem.

Theorem 8. [7] *Any randomized online algorithm has a competitiveness of at least* $\Omega((4/3)^\alpha)$.

An interesting open problem is to determine the best competitiveness that can be achieved by online algorithms.

3.2 Bounded Speed

The algorithms presented in the last section are designed for processors having available a continuous, unbounded spectrum of speeds. However, in practice a processor is equipped with only a finite set of discrete speed levels $s_1 < s_2 < \ldots < s_d$. The offline algorithm *YDS* can be modified easily to handle feasible job instances, i.e. inputs for which feasible schedules exist using the restricted set of speeds. Feasibility can be checked easily by always using the maximum speed s_d and scheduling available jobs according to the *EDF* policy. Given a feasible job instance the modification of *YDS* is as follows. We first construct the schedule according to *YDS*. For each identified interval I of maximum density we approximate the desired speed Δ_I by the two adjacent speed levels s_k and s_{k+1}, such that $s_k < \Delta_I < s_{k+1}$. Speed s_{k+1} is used first for some δ time units and s_k is used for the last $|I| - \delta$ time units in I, where δ is chosen such that the total work completed in I is equal to the original amount of $|I|\Delta_I$. An algorithm with an improved running time of $O(dn \log n)$ was presented by Li and Yao [17].

If the given job instance is not feasible, it is impossible to complete all the jobs. Here the goal is to design algorithms that achieve good *throughput*, which is the total processing volume of jobs finished by their deadline, and at the same time optimize energy consumption. Papers [5,11] present algorithms that even work online. At any time the strategies maintain a pool of jobs they intend to complete. Newly arriving jobs may be admitted to this pool. If the pool contains too large a processing volume, jobs are expelled such that the throughput is not diminished significantly. The algorithm with the best competitiveness currently known is due to Bansal et al. [5]. The algorithm, called *Slow-D*, is 4-competitive in terms of throughput and constant competitive with respect to energy consumption. We describe the strategy.

Slow-D assumes that the processor has a continuous speed spectrum that is upper bounded by a maximum speed $s_{\max}$. The algorithm always keeps track of the speeds that *Optimal Available* would use for the workload currently available. At any time t *Slow-D* uses the speed that *Optimal Available* would set at time t provided that this speed does not exceed $s_{\max}$; otherwise *Slow-D* uses $s_{\max}$. The algorithm also considers scheduling times that are critical in terms of speed. For any t, down-time(t) is the latest time $t' \geq t$ in the future schedule such that the speed of *Optimal Available* is at least $s_{\max}$. If no such time exists, down-time(t) is set to the most recent time when $s_{\max}$ was used or to 0 if this has never been the case. Using this definition, jobs are labeled as *urgent* or *slack*. These labels may change over time. A job J_i is called t-urgent if $d_i \leq$ down-time(t); otherwise it is called t-slack. Additionally, *Slow-D* maintains two queues Q_{work} and Q_{wait} of jobs it intends to process. The status of Q_{work} defines *urgent periods*. An urgent period starts at the release time r_i of a job J_i if Q_{work} contained no urgent job right before r_i and J_i is an urgent job admitted to Q_{work} at time r_i. An urgent period ends at time t if Q_{work} contains no more t-urgent jobs. *Slow-D* works as follows.

Algorithm Slow-D:
JOB ARRIVAL: A job J_i arriving at time r_i is admitted to Q_{work} if it is r_i-slack or if J_i and all the remaining work of r_i-urgent jobs in Q_{work} can be completed using $s_{\max}$. Otherwise J_i is appended to Q_{wait}.
JOB INTERRUPT: Whenever a job J_i in Q_{wait} reaches its last starting time $t = d_i - w_i/s_{\max}$, it raises an interrupt. At this time the algorithm is in an urgent period. Let J_k be the last job transfered from Q_{wait} to Q_{work} in the current period. If no such job exists, let J_k be a dummy job of processing volume zero transfered just before the current period started. Let W be the total orginal work of jobs ever admitted to Q_{work} that have become urgent after J_k was transfered to Q_{work}. If $w_i > 2(w_k + W)$, then remove all t-urgent jobs from Q_{work} and admit J_i; otherwise discard J_i.
JOB COMPLETION: Whenever a job is completed, it is removed from Q_{work}.

Bansal et al. [5] analyzed the above algorithm and proved the following result.

Theorem 9. [5] *Slow-D is 4-competitive with respect to throughput and* $(\alpha^\alpha + \alpha^2 4^\alpha)$*-competitive with respect to energy.*

Interestingly, the competitiveness of 4 is best possible, even if energy is ignored. More precisely, Baruah et al. [8] showed that no online algorithm can be better than 4-competitive considering only throughput maximization in overloaded systems where the deadlines cannot be met for all jobs.

4 Conclusions

This paper has surveyed algorithmic solutions to save energy in computing devices. As the field has attracted considerable research interest recently, this survey is not exhaustive. In particular, as for dynamic speed scaling, many more results have been developed. A classical objective function in scheduling is the minimization of response times as it models user satisfaction and quality of service. Unfortunately, energy minimization and response time minimization are contradicting objectives. The studies in [2,5,6,9,15,20] integrate both measures and develop offline as well as online algorithms. Another basic objective function in scheduling theory is makespan minimization, i.e. the minimization of the point in time when all jobs are finished. This measure, in the presence of energy minimization, has been considered in [10,19]. In summary we expect that the design and analysis of energy-efficient algorithm will continue to be an active area of research during the next years.

References

1. http://www.microsoft.com/whdc/system/pnppwr/ powermgmt/default.mspx
2. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Transactions on Algorithms 3 (2007)
3. Augustine, J., Irani, S., Swamy, C.: Optimal power-down strategies. SIAM Journal on Computing 37, 1499–1516 (2008)

4. Bansal, N., Bunde, D.P., Chan, H.-L., Pruhs, K.: Average rate speed scaling. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 240–251. Springer, Heidelberg (2008)
5. Bansal, N., Chan, H.-L., Lam, T.-W., Lee, K.-L.: Scheduling for speed bounded processors. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 409–420. Springer, Heidelberg (2008)
6. Bansal, N., Chan, H.-L., Pruhs, K.: Speed scaling with an arbitrary power function. In: Proc. 20th ACM-SIAM Symposium on Discrete Algorithm, pp. 693–701 (2009)
7. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. Journal of the ACM 54 (2007)
8. Baruah, S.K., Koren, G., Mishra, B., Raghunathan, A., Rosier, L.E., Shasha, D.: On-line scheduling in the presence of overload. In: Proc. 32nd Annual Symposium on Foundations of Computer Science, pp. 100–110 (1991)
9. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 805–813 (2007)
10. Bunde, D.P.: Power-aware scheduling for makespan and flow. In: Proc. 18th Annual ACM Symposiun on Parallel Algorithms and Architectures, pp. 190–196 (2006)
11. Chan, H.-L., Chan, W.-T., Lam, T.-W., Lee, K.-L., Mak, K.-S., Wong, P.W.H.: Energy efficient online deadline scheduling. In: Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 795–804 (2007)
12. Irani, S., Karlin, A.R.: Online computation. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-Hard Problems, pp. 521–564. PWS Publishing Company (1997)
13. Irani, S., Shukla, S.K., Gupta, R.K.: Online strategies for dynamic power management in systems with multiple power-saving states. ACM Transaction in Embedded Computing Systems 2, 325–346 (2003)
14. Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S.S.: Competitive randomized algorithms for nonuniform problems. Algorithmica 11, 542–571 (1994)
15. Lam, T.-W., Lee, L.-K., To, I.K.-K., Wong, P.W.H.: Speed scaling functions for flow time scheduling based on active job count. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 647–659. Springer, Heidelberg (2008)
16. Li, M., Liu, B.J., Yao, F.F.: Min-energy voltage allocation for tree-structured tasks. Journal on Combintorial Optimization 11, 305–319 (2006)
17. Li, M., Yao, F.F.: An efficient algorithm for computing optimal discrete voltage schedules. SIAM Journal on Computing 35, 658–671 (2005)
18. Li, M., Yao, A.C., Yao, F.F.: Discrete and continuous min-energy schedules for variable voltage processors. Proc. National Academy of Sciences USA 103, 3983–3987 (2006)
19. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. Theory of Computing Systems 43, 67–80 (2008)
20. Pruhs, K., Uthaisombut, P., Woeginger, G.J.: Getting the best response for your erg. ACM Transactions on Algorithms 4 (2008)
21. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communcations of the ACM 28, 202–208 (1985)
22. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. 36th IEEE Symposium on Foundations of Computer Science, pp. 374–382 (1995)

Sofic and Almost of Finite Type Tree-Shifts

Nathalie Aubrun and Marie-Pierre Béal

Université Paris-Est
Laboratoire d'informatique Gaspard-Monge, CNRS

Abstract. We introduce the notion of sofic tree-shifts which corresponds to symbolic dynamical systems of infinite trees accepted by finite tree automata. We show that, contrary to shifts of infinite sequences, there is no unique minimal deterministic irreducible tree automaton accepting an irreducible sofic tree-shift, but that there is a unique synchronized one, called the Shannon cover of the tree-shift. We define the notion of almost finite type tree-shift which is a meaningful intermediate dynamical class in between irreducible finite type tree-shifts and irreducible sofic tree-shifts. We characterize the Shannon cover of an almost finite type tree-shift and we design an algorithm to check whether a sofic tree-shift is almost of finite type.

1 Introduction

In a previous article [1], we introduced the notion of tree-shifts of finite type defined as sets of infinite trees avoiding a finite number of forbidden patterns. Infinite trees have a natural structure of one-sided symbolic systems equipped with several shift transformations. The ith shift transformation applied to a tree gives the subtree rooted at the child number i of the tree. Tree-shifts are highly interesting to study as they constitute an intermediate class between one-sided shifts of infinite sequences and multidimensional shifts.

The conjugacy of multidimensional shifts of finite type, also called textile systems or tiling systems (see for instance [11],[14],[8],[6]), is undecidable. However, the conjugacy of (one-sided) shift spaces of finite type of infinite sequences is decidable ([19], see also [12]). In [1], we extended William's result to trees, showing that the conjugacy of irreducible tree-shifts of finite type is decidable.

In this paper, we focus on sofic tree-shifts, which are shifts of infinite trees accepted by finite (bottom-up) tree automata, and thus whose set of patterns is a recognizable set of finite trees. The goal is to extend to trees some results of sofic shifts of infinite sequences, to define a hierarchy a sofic tree-shifts and characterize each level of this hierarchy.

We introduce the notion of irreducible sofic tree-shifts. We show, that, unlike for sofic shifts of sequences, an irreducible sofic tree-shift may be accepted by several minimal deterministic irreducible tree automata. This is due to the lack of a synchronizing block, even in minimal irreducible automata. We introduce the notion of synchronized tree automaton and the notion of Shannon cover of an irreducible sofic tree-shift. We prove that the Shannon cover is the unique minimal synchronized tree automaton accepting an irreducible sofic tree-shift.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 12–24, 2010.

The existence of the Shannon cover allows us to introduce the class of almost finite type tree-shifts, which extends the known notion of almost of finite type shifts of sequences. The almost of finite type concept was introduced by Marcus in [13] for coding purposes. The class of almost of finite type shifts is a meaningful class of shifts between irreducible shifts of finite type and irreducible sofic shifts (see [4], [20], [9], [3], [2]). The class contains strictly the class of irreducible shifts of finite type and is strictly contained in the class of sofic shifts. The class is stable by conjugacy and it is also invariant by a flow equivalence [7]. We characterize the Shannon cover of an almost of finite type tree-shift and design an algorithm to check whether a sofic tree-shift is almost of finite type.

The paper is organized as follows. In Section 2.1 and Section 2.2, we give basic definitions about tree-shifts and conjugacies. In Section 3, we define the notion of automaton accepting a tree-shift. We refer to [5], [18], [15] for more general trees and automata on finite and infinite trees. The notion of Shannon cover is introduced in Section 3.4. The characterization of almost of finite type tree-shifts is done in Section 4. In the algorithmic issue, Section 5, we give a construction of the Shannon cover of a sofic tree-shift. We design a polynomial-time algorithm to check whether a sofic tree-shift given by its Shannon cover is almost of finite type. Some proofs are omitted in this version of the paper.

2 Definitions

2.1 Tree-Shifts

We first recall some basic definitions of symbolic dynamics on infinite trees (see [1] for more details). We consider infinite trees whose nodes have a fixed number of children and are labeled in a finite alphabet. We restrict to binary trees, but all result extend to the case of trees with d children for all $d \geq 1$.

Let $\Sigma = \{0, 1\}$. An *infinite tree* t over a finite alphabet A is a complete function from Σ^* to A. Unless otherwise stated, a tree is an infinite tree. A node of a tree is a word of Σ^*. The empty word, that corresponds to the root of the tree, is denoted by ϵ. If x is a node, its children are xi with $i \in \Sigma$. Let t be a tree and let x be a node, we shall denote $t(x)$ by t_x. When Σ is fixed, we denote by $\mathcal{T}(A)$ the set of all infinite trees on A, hence the set A^{Σ^*}.

We define the shift transformations σ_i for $i \in \Sigma$ from $\mathcal{T}(A)$ to itself as follows. If t is a tree, $\sigma_i(t)$ is the tree rooted at the i-th child of t, I.e. $\sigma_i(t)_x = t_{ix}$ for all $x \in \Sigma^*$. The set $\mathcal{T}(A)$ equipped with the shift transformations σ_i is called the *full shift* of infinite trees over A. A sequence of words $(x_k)_{k\geq 0}$ of Σ^* is called a *path* if for all $k, x_{k+1} = x_k i_k$ with $i_k \in \Sigma$.

A *pattern* is a function $p : L \to A$, where L is a finite prefix-closed[1] subset of Σ^*. The set L is called the *support of the pattern*. A *block of height* n is a pattern with support $\Sigma^{\leq n}$, where n is some nonnegative integer, and $\Sigma^{\leq n}$ denotes the words of length at most n of letters of Σ. The *height* of a block u is denoted by height(u). A *leaf* of a pattern is a node with no child.

[1] Each prefix of L belongs to L.

We say that a block u of support L is a *block of a tree* t if there is a word $x \in \Sigma^*$ such that $t_{xy} = u_y$ for all words $y \in \Sigma^*$. We say that u is a block of t rooted at the node x. If u is not a block of t, one says that t *avoids* p.

We define a *tree-shift space* (or *tree-shift*) X of $\mathcal{T}(A)$ as the set $\mathsf{X}_\mathcal{F}$ of all trees avoiding each element of a set of blocks $\mathcal{F}$. If the set of trees on A is equipped with the usual product topology, where the topology in A is the discrete one, a tree-shift space is closed and invariant for any shift transformation σ_i. A *tree-shift of finite type* (SFT) X of $\mathcal{T}(A)$ is a set $\mathsf{X}_\mathcal{F}$ of all trees avoiding each block of a *finite* set of blocks $\mathcal{F}$. The set $\mathcal{F}$ is called *a set of forbidden blocks* of X.

We denote by $\mathcal{L}(X)$ the set of patterns of all trees of the tree-shift X, by $\mathcal{B}(X)$ the set of all blocks of X and by $\mathcal{B}_n(X)$ the set of all blocks of height n of X. If u is a block of height n with $n \geq 1$, we denote by $\sigma_i(u)$ the block of height $n-1$ such that $\sigma_i(u)_x = b_{ix}$ for $x \in \Sigma^{\leq n-1}$. The block u is written $u = (u_\varepsilon, \sigma_0(u), \sigma_1(u))$.

Example 1. In Figure 1 is pictured an infinite tree of a tree-shift X on the alphabet $\{a, b\}$. The forbidden blocks are those containing an even number of a between two b on any path in the tree. This tree-shift is not of finite type.

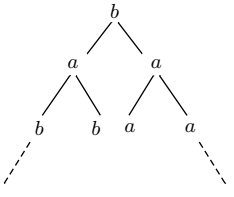


Fig. 1. An infinite tree of the tree-shift $\mathsf{X}_\mathcal{F}$, where $\mathcal{F}$ is the set of patterns containing an even number of a between two b on any path in the tree

2.2 Block Maps and Conjugacies

Let A, A' be two finite alphabets, X be a tree-shift of $\mathcal{T}(A)$ and m be a positive integer. A map $\Phi : X \subseteq \mathcal{T}(A) \to \mathcal{T}(A')$ is called an *m-local map* (or an *m-block map*) if there exists a function $\phi : \mathcal{B}_m(X) \to A'$ such that, for all $x \in \Sigma^*$, $\Phi(t)_x = \phi(t_{x\Sigma^{\leq m-1}})$, where $t_{x\Sigma^{\leq m-1}}$ is the block q such that $q_y = t_{xy}$ for all $y \in \Sigma^{\leq m-1}$. The smallest integer $m-1$ such that Φ is an m-block map, is called the *memory* of the block map. A *block map* is a map which is an m-block map for some nonnegative integer m.

The image of X by a block map is also a tree-shift, and is called a *factor* of X. A one-to-one and onto block map from a tree-shift X onto a tree-shift Y has an inverse which is also a block map, as for shifts of sequences. It is called a *conjugacy* from X onto Y. The tree-shifts X and Y are then *conjugate*. We call *sofic* a tree-shift which is a factor of a tree-shift of finite type.

Let X be a tree-shift and m a positive integer. We denote by $X^{(m)}$ the *higher block presentation of* X. It is a tree-shift on the alphabet $\mathcal{B}_m(X)$. For each tree t in $X^{(m)}$, there is tree t' in X such that, for each node x, t_x is the block of height m of t' rooted at x. The shifts X and $X^{(m)}$ are conjugate (see [1]).

3 Sofic Tree-Shifts

3.1 Tree Automata

In this section we consider bottom-up automata of infinite trees. Such an automaton starts its computation from the infinite branches and moves upward. A *tree automaton* is here a structure $\mathcal{A} = (V, A, \Delta)$ where V is a finite set of states, A is a finite set of input symbols, and Δ is a set of transitions of the form $(q_0, q_1), a \to q$, with $q, q_i \in V$, $a \in A$. A transition $(q_0, q_1), a \to q$ is called a transition *labeled by* a, *going out of* the pair of states (q_0, q_1) and *coming in* the state q. A transition $(q_0, q_1), a \to q$ will be pictured by

Note that no initial nor final states are specified. This means that all states are both initial and final.

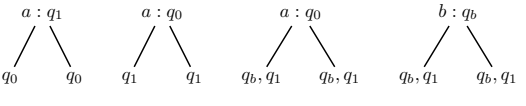
Such an automaton is *deterministic* if for each pair of states (q_0, q_1) and for each $a \in A$, there is at most one transition $(q_0, q_1), a \to q$. Then the set of transitions defines a partial function δ from $V^2 \times A$ to V.

A (bottom-up) *computation* of $\mathcal{A}$ on the infinite tree t is an infinite tree C on V such that, for each node x, there is a transition $(C_{x0}, C_{x1}), t_x \to C_x \in \Delta$. A tree t is *accepted* by $\mathcal{A}$ if there exists a computation of $\mathcal{A}$ on t. The set of infinite trees accepted by $\mathcal{A}$ is a tree-shift. Given a tree automaton $\mathcal{A}$, it is always possible to transform it into a deterministic tree automaton which accepts the same set of trees (this process is called determinization, see for instance [5] for details). In the sequel, we assume that all states of an automaton are *accessible*, I.e. each state ends some computation of the automaton.

A (bottom-up) *finite computation* of $\mathcal{A}$ on the complete finite tree t is a complete finite tree C on V such that, for each node x which is not a leaf, there is a transition $(C_{x0}, C_{x1}), t_x \to C_x \in \Delta$.

A tree automaton is called *an edge tree automaton* is all transitions have distinct labels. An *edge tree-shift* is a tree-shift (of finite type) accepted by an edge tree automaton.

Example 2. We define a tree automaton $\mathcal{A}$ with three state q_b, q_0 and q_1 which accepts the tree-shift X of Example 1. The two states q_0 and q_1 only label nodes with an a, and they control the parity of the number of a encountered from any last b below. The state q_b only labels nodes with a b. The transitions of the tree automaton $\mathcal{A}$ are



The proofs of the following proposition is similar to the one for shifts of infinite or bi-infinite sequences (see [12], [10]).

Proposition 1. *A tree-shift is sofic if and only if it is accepted by a tree automaton.*

3.2 Irreducible Tree-Shifts

In this section, we define a notion of irreducibility which is suitable for tree-shifts.

A *finite complete prefix code* of Σ^* is a prefix set [2] P of finite words in Σ^* such that each word of Σ^* longer than the words of P has a prefix in P.

A tree-shift X is *irreducible* if for each pair of blocks $u, v \in \mathcal{B}(X)$, there is a tree t in X and a finite complete prefix code $P \subset \Sigma^{\geq \text{height}(u)}$, such that u is a subtree of t rooted at ε, and v is a subtree of t rooted at x for all $x \in P$.

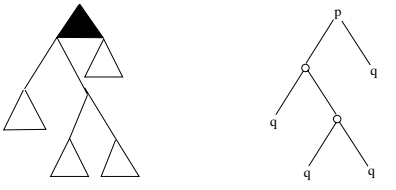


Fig. 2. (left) An irreducible tree-shift. Let t denotes the tree pictured. If u denotes the black block and v the white one, u is a subtree of t rooted at ε, and v is a subtree of t rooted at each $x \in P$, where P is the complete prefix code $\{00, 010, 011, 1\}$. (right) An hyperpath from q to p in a tree automaton.

A tree automaton is *irreducible* if for each pair of states p, q, there is a finite complete prefix code P of Σ^* and a finite computation C of the automaton on a pattern u such that $C_\varepsilon = p$ and $C_x = q$ for each $x \in P$. We say in this case that there is an *hyperpath* from q to p labeled by u. For two states p, q of a tree automaton, we say that p is *accessible* from q if there is a hyperpath from q to p.

Proposition 2. *An irreducible automaton accepts an irreducible sofic tree-shift. Conversely, for any irreducible sofic tree-shift, there is an irreducible automaton accepting it.*

Proposition 3. *Let S and T be two conjugate tree-shifts. Then S is irreducible if and only if T is irreducible.*

3.3 Synchronizing Blocks

We define below the notion of synchronizing block[3] of a deterministic tree automaton.

Let $\mathcal{A} = (V, A, \Delta)$ be a deterministic tree automaton accepting a sofic tree-shift X, and u be a block (resp. pattern). We say that u is a *synchronizing block (resp. pattern) of* $\mathcal{A}$ if all computations of $\mathcal{A}$ on u terminate at the same state $q \in Q$. We say that u *focuses* to the state q. A deterministic tree automaton which has a synchronizing block is called *synchronized*.

[2] I.e. no word is prefix of another one.

[3] Also called a homing pattern or a magic pattern.

3.4 Minimal Deterministic Tree Automata

Let X be a tree-shift. A *context* c is a finite pattern with a marked leaf. If u is a pattern, $c(u)$ is the pattern c where the marked leaf is replaced by u. If $c(u) \in \mathcal{L}(X)$, we say that c is a *context of* u *in* X. Given a block u, we denote by $\mathrm{cont}_X(u)$ the set of all the contexts of u in X.

Given a tree automaton $\mathcal{A} = (V, A, \Delta)$ accepting a sofic tree-shift X, the *context* of a state $q \in V$ in $\mathcal{A}$ is the set of patterns u with a marked leaf x on which there exists a finite computation C of $\mathcal{A}$ with $C_x = q$ and x is a leaf of C. We denote it by $\mathrm{cont}_{\mathcal{A}}(q)$. Note that the context of a pattern u of X is the union of the contexts of the states p such that there is a computation of $\mathcal{A}$ on u ending in p. As a consequence, a sofic tree-shift has only a finite number of distinct contexts.

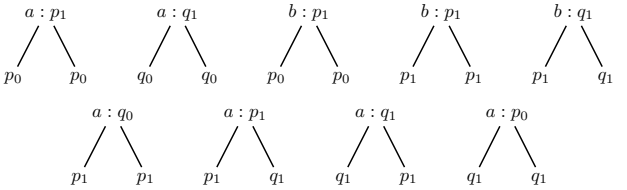
Let $\mathcal{A} = (V, A, \Delta)$ be a deterministic automaton accepting a sofic tree-shift X. We denote by $\delta(p, q, a)$ the unique state r such that $(p, q), a \to r \in \Delta$ when such a transition exists. We define a deterministic automaton $\mathcal{M}_{\mathcal{A}}$ accepting X called the *minimization of the automaton* $\mathcal{A}$ as follows. The states of $\mathcal{M}_{\mathcal{A}}$ are the classes of the coarsest partition of V such that if p, q belong to the same class, then for each letter a and each state r, $\delta(p, r, a)$ and $\delta(q, r, a)$ (resp. $\delta(r, p, a)$ and $\delta(r, q, a)$) belong to the same class, and $\delta(p, r, a)$ (resp. $\delta(r, p, a)$) is defined if and only if $\delta(q, r, a)$ (resp. $\delta(r, q, a)$) is defined. Let $[p]$ denotes the class of the state p. The transition $([p], [q]), a \to [\delta(p, q, a)]$ is a transition of $\mathcal{M}_{\mathcal{A}}$ if and only if $\delta(p, q, a)$ is defined. It can be shown that this definition is consistent, I.e. does not depend on the choice of the leader in each class. A deterministic automaton is *minimal* if it is equal to its minimization.

For any deterministic tree automaton, two states in a same class have the same context. If $\mathcal{A}$ is moreover irreducible and synchronized, it is minimal if and only if any two states have distinct contexts.

The minimization algorithm for deterministic tree automata accepting languages of finite trees, which is for instance described in [5, Section 1.5], can be applied to the tree automata accepting tree-shifts. Remember that all states in tree automata accepting tree-shifts are both initial and final.

In the framework of shifts of bi-infinite words, irreducible sofic shifts have a unique minimal deterministic automaton (I.e. all minimal deterministic automata accepting the shift are equal up to a renaming of the states), see [12]. The situation is quite different for trees since, as is shown below in Example 3, irreducible sofic tree-shifts may have several minimal deterministic tree automata. Indeed, contrary to the situation that we have for shifts of infinite or bi-infinite sequences, an irreducible minimal deterministic tree automaton may not have a synchronizing block.

Example 3. Let X be the full tree-shift on the alphabet $A = \{a, b\}$. It is accepted by a trivial one-state automaton. It is also accepted by the deterministic irreducible tree automaton $\mathcal{A} = (V, A, \delta)$ described by the following transitions and which is minimal.



One can overcome this difficulty with the notion of Shannon cover for irreducible tree-shifts.

Let X be a sofic tree-shift. The *context tree automaton* of X is the deterministic automaton $\mathcal{S} = (V, A, \Delta)$, where V is set of contexts of finite blocks of X. Since X is sofic, V is finite. The transitions of $\mathcal{S}$ are $(\mathrm{cont}_X(u), \mathrm{cont}_X(v)), a \to \mathrm{cont}_X(a, u, v)$, where $u, v \in \mathcal{B}(X)$.

Proposition 4. *The context tree automaton of a sofic tree-shift is synchronized.*

Proof. Let $\mathcal{S}$ be the context tree automaton of a sofic shift X. There is a finite computation C_1 in $\mathcal{S}$ on some block u_1 ending in $\mathrm{cont}_X(u_1)$. Let us assume that u_1 is not a synchronizing block of $\mathcal{S}$. Hence there is another computation C_2 of $\mathcal{S}$ on u_1 ending in $\mathrm{cont}_X(u_2)$ for some block $u_2 \neq u_1$. We get $\mathrm{cont}_X(u_2) \subsetneq \mathrm{cont}_X(u_1)$ since $\mathrm{cont}_X(u_2) \neq \mathrm{cont}_X(u_1)$. If u_2 is not a synchronizing block of $\mathcal{S}$, there is another computation C_2 of $\mathcal{S}$ on u_2 ending in $\mathrm{cont}_X(u_3)$. We get $\mathrm{cont}_X(u_3) \subsetneq \mathrm{cont}_X(u_2)$. By iterating this process, we either get a synchronizing block for $\mathcal{S}$ or an infinite strictly decreasing sequence of contexts. Hence, since the number of contexts is finite, there is a synchronizing block.

We define the *Shannon cover* of an irreducible sofic tree-shift X as the unique irreducible component $\mathcal{S}$ of its context tree automaton $\mathcal{C}$ obtained by keeping the states accessible from $\mathrm{cont}_{\mathcal{C}}(z)$, where z is a synchronizing block of $\mathcal{C}$. Since z is synchronizing and $\mathcal{C}$ is deterministic, each state in this component is the context of some synchronizing block.

Let us show that the states accessible from $\mathrm{cont}_{\mathcal{C}}(z)$ form an irreducible component and that it is the unique irreducible component of $\mathcal{S}$. It is enough to prove that there is a hyperpath from any state to $\mathrm{cont}_{\mathcal{C}}(z)$. Let $p = \mathrm{cont}_{\mathcal{C}}(u)$ be a state. Since X is irreducible, there is a pattern w of X and a finite complete prefix code P of $\Sigma^{\geq \mathrm{height}(z)}$, such that z is a subtree of w rooted at ε, and u is a subtree of w rooted at x for all $x \in P$. Let C be a computation of $\mathcal{C}$ on w. We have $C_\varepsilon = \mathrm{cont}_{\mathcal{C}}(z)$, and for all $x \in P$, $C_x = p$. Hence there is an hyperpath from p to $\mathrm{cont}_{\mathcal{C}}(z)$. Finally, it is easy to check that the automaton $\mathcal{S}$ accepts X by using the irreducibility of X.

We now prove that the Shannon cover is the unique minimal deterministic irreducible and synchronized tree automaton accepting an irreducible sofic tree-shift.

Proposition 5. *Two minimal deterministic irreducible and synchronized tree automata accepting the same irreducible sofic tree-shift are equal up to a renaming of the states.*

4 Almost of Finite Type Tree-Shifts

The following notion of almost of finite type tree-shift extends to trees the notion of almost of finite type shift of bi-infinite sequences. The class contains strictly the class of irreducible tree-shifts of finite type and is strictly contained in the class of irreducible sofic tree-shifts.

Let X, Y be two tree-shifts. A block map $\Phi : X \to Y$ is *left closing* if there are non negative integers m, a (m for memory and a for anticipation) such that, whenever $\Phi(s) = s'$, $\Phi(t) = t'$ with $s'_x = t'_x$ for all $x \in \Sigma^i$ with $0 \leq i \leq (a+1+m)$, and $s_x = t_x$ for all $x \in \Sigma^i$ with $0 \leq i \leq a$, then $s_x = t_x$ for any $x \in \Sigma^{(a+1)}$. The map Φ is *left resolving* if m can be chosen equal to 0.

A tree automaton $\mathcal{A} = (V, A, \Delta)$ is *left closing* if any two computations of $\mathcal{A}$ on a same tree and ending in a same state are equal. Equivalently, there is a nonnegative integer m such any two finite computations C, C' of $\mathcal{A}$ on a same block $u \in \mathcal{B}_{m+1}(X)$ such that $C_\varepsilon = C'_\varepsilon$ satisfy $C_x = C'_x$ for all $x \in \{0, 1\}$. Hence a deterministic and left closing tree automaton corresponds, for trees, to the notion of automata of words which are both deterministic and co-deterministic with a finite delay.

A block map $\Phi : X \to Y$ is *right closing* if there are non negative integers m, a such that, whenever $\Phi(s) = s'$, $\Phi(t) = t'$ with $s'_x = t'_x$ for all $x \in \Sigma^i$ with $0 \leq i \leq (a + 1 + m)$, and $s_x = t_x$ for all $x \in \Sigma^i$ with $a + 1 \leq i \leq (a + 1 + m)$, then $s_x = t_x$ for all $x \in \Sigma^a$. The map Φ is *right resolving* if a can be chosen equal to 0.

Let X be a tree-shift. Let u be a block in $\mathcal{B}_m(X)$. A *cylinder* $\mathcal{C}[u]$ of X with basis u is the set of trees t in X such that $t_x = u_x$ for all $x \in \Sigma^*$ of length at most m.

A sofic tree-shift is *almost of finite type* (AFT) if it is the image of an irreducible tree-shift of finite type via a block map which is right resolving, left closing, and is one-to-one on a cylinder (equivalently, the map is one-to-one on a non trivial open set).

Proposition 6. *Let S and T be two conjugate irreducible sofic tree-shifts. Then S is AFT if and only if T is AFT.*

Proof. (sketch) It is easy to check that the composition of a right resolving map with a conjugacy is still right resolving. The composition of a left closing map with a conjugacy is still left closing. A conjugacy also keeps the property of being one-to-one on a cylinder.

We say that an automaton is an *almost of finite type automaton* (AFT automaton) if it is deterministic, irreducible, left closing and synchronized.

Proposition 7. *A tree-shift is AFT if and only if it is accepted by an AFT automaton.*

Proof. Let S be an AFT tree-shift on the alphabet A. Let $\Phi : X \to S$ be a block map from an irreducible tree-shift of finite type X onto S which is right resolving, left closing and one-to-one on a cylinder of S. Without loss of generality (by changing X into a higher block presentation of X), we can assume

that X is an edge tree-shift and that Φ is a one-block map. Again by changing X into a higher block presentation of X, we can assume that Φ is left closing with parameters $a' = 1, m'$.

Let $\mathcal{A} = (V, E, \Delta)$ be an irreducible edge tree automaton accepting X. Let $\mathcal{B} = (V, A, \Delta')$ where $(p, q, \phi(e), r) \in \Delta'$ if and only if $(p, q, e, r) \in \Delta$.

Since Φ is a one-block right-resolving map (with some parameter m), $\mathcal{B}$ is a deterministic automaton.

We now prove that $\mathcal{B}$ has a synchronizing block. Since Φ is one-to-one on a cylinder $\mathcal{C}[z_0]$ of S, for any tree $t \in \mathcal{C}[z_0]$, any two computations of $\mathcal{B}$ on t are equal and thus end in a same state p. As a consequence, and by compacity arguments, there is a finite subtree z such that z_0 is a subtree of z at ε and such that each finite computation of $\mathcal{B}$ on z ends in the same state p_z, I.e. z is a synchronizing block of $\mathcal{B}$. Let us keep in $\mathcal{B}$ only the states accessible from p_z. Since S is irreducible, $\mathcal{B}$ remains irreducible and still accepts S.

Let us now show that $\mathcal{B}$ is left closing. If $\mathcal{B}$ were not left closing, there would be two distinct computations C, C' of $\mathcal{B}$ on a same tree t ending in a same state p. Let $(p, q, e, r) \in \Delta$ a transition going out of (p, q) for some states $p, r \in V$ (if there is none, there is a transition going out of (q, p) since $\mathcal{A}$ is irreducible). We get two distinct computations (r, C, D), (r, C', D) of $\mathcal{B}$ ending in r on a same tree $u = (\phi(e), t, t')$ for some tree t'. These two distinct computations of $\mathcal{B}$ are also two distinct computations of $\mathcal{A}$ on two trees s, s' of X with $s_\varepsilon = s'_\varepsilon = e$ and such that $\Phi(s) = \Phi(s') = u$. Since Φ is left closing with parameters $a' = 1, m'$, we get $s = s'$ and thus $C = C'$.

Conversely, if S is accepted by an AFT automaton $\mathcal{A} = (V, A, \Delta)$, let X be the edge tree-shift accepted by $\mathcal{T} = (V, \Delta, \Delta')$, whose transitions are $(p, q, (p, q, a, r)) \to r$ for $(p, q, a, r) \in \Delta$. Let Φ be the one-block map from X onto S defined by $\phi(p, q, a, r) = a$. This map is right resolving since $\mathcal{A}$ is deterministic. It is left closing since $\mathcal{A}$ is left closing. It is one-to-one on a cylinder since $\mathcal{A}$ is synchronized. As a consequence, S is AFT.

Corollary 1. *An irreducible sofic tree-shift is AFT if and only if its Shannon cover is AFT.*

Proof. Let X be an irreducible sofic tree-shift. By Proposition 5, any irreducible sofic tree-shift is accepted by an irreducible deterministic synchronized automaton $\mathcal{S}$ equal to the Shannon cover of X.

Let us assume that X is AFT. By Proposition 7, $\mathcal{S}$ is equal to the minimization of an AFT automaton $\mathcal{A}$. Let us show that $\mathcal{S}$ is left closing. If it is not left closing, then there is a tree $t \in X$ and two distinct computations of $\mathcal{S}$ on t ending in a same state. As a consequence, there are two distinct computations of $\mathcal{A}$ on t ending in two states p, p' which have the same context.

Let z be a synchronizing pattern of $\mathcal{A}$ focusing to the state q_z. Since $\mathcal{A}$ is irreducible, there is an hyperpath from p to q_z labeled by some pattern w such that z is a subtree of w rooted at ε. Since p and p' have the same context, there is also a hyperpath from p' to q labeled by w. Since z is synchronizing, $q = q_z$.

This gives two distinct computations of $\mathcal{A}$ on a same tree ending in the same state q_z, which contradicts the fact that $\mathcal{A}$ is left closing. This proves that the Shannon cover of an AFT is left closing.

Conversely, if $\mathcal{S}$ is AFT, then X is AFT by Proposition 7.

Corollary 2. *Let S be an irreducible sofic shift accepted by a deterministic tree automaton. It is decidable whether S is AFT.*

Proof. One can compute the Shannon cover of the sofic tree-shift and check whether it is AFT as explained in Section 5.

5 The Algorithmic Issue

In this section, we design algorithms to check whether a deterministic tree automaton is synchronized, and whether it is left closing. We also describe an algorithm to compute the Shannon cover of an irreducible sofic tree-shift given by a deterministic tree automaton that accepts it.

5.1 Computation of the Shannon Cover

Let $\mathcal{A} = (V, A, \delta)$ be a deterministic automaton. We define the deterministic tree automaton $\mathcal{D}(\mathcal{A})$ as the accessible part from the state V of the tree automaton $(\mathfrak{P}(V), A, \delta')$, where for, $P, Q \in \mathfrak{P}(V)$, $\delta'(P, Q, a) = \{\delta(p, q, a) \mid p \in P, q \in Q\}$ if this set is nonempty, and is not defined otherwise.

Proposition 8. *It can be checked in polynomial time whether a tree automaton is irreducible.*

Proposition 9. *It is decidable whether a deterministic tree automaton is synchronized.*

Proof. The automaton $\mathcal{A}$ is synchronized if and only if $\mathcal{D}(\mathcal{A})$ contains a singleton state. The time and space complexity of this algorithm is exponential in the number of states of $\mathcal{A}$.

Proposition 10. *Let $\mathcal{A} = (V, A, \delta)$ be a deterministic automaton accepting an irreducible sofic tree-shift X. The Shannon cover of X is computable from $\mathcal{A}$.*

Proof. Let $\mathcal{D}(\mathcal{A}) = (\mathfrak{P}(V), A, \delta')$ and R be a minimal state of $\mathcal{D}(\mathcal{A})$ for the inclusion. Let u the label of a hyperpath from V to R. Then u a synchronizing pattern of $\mathcal{D}(\mathcal{A})$. Indeed, any finite computation of $\mathcal{D}(\mathcal{A})$ ends in R by minimality of R. We keep in $\mathcal{D}(\mathcal{A})$ only the states accessible from R and get an irreducible and synchronized automaton accepting X. Its minimization gives the Shannon cover of X by Proposition 5.

We now describe algorithms to check whether an irreducible deterministic automaton is left closing.

5.2 The Pair Graph of a Tree Automaton

Given a deterministic tree automaton $\mathcal{A} = (V, A, \Delta)$, we define the *square automaton* of $\mathcal{A}$, denoted by $\mathcal{A} \times \mathcal{A} = (V \times V, A, \Delta')$, as the deterministic automaton whose transitions are $(p, p'), (q, q'), a \to (r, r')$ if and only if $(p, q), a \to r$ and $(p', q'), a \to r'$ are transitions of $\mathcal{A}$. A *diagonal state* of $\mathcal{A} \times \mathcal{A}$ is a state (p, p) for some $p \in V$.

Square automata of finite words (see for instance [16, p. 647]) are used to check properties of pairs of paths. We extend this notion, together with a notion a pair graph, to trees, to check properties of pairs of computations. Seidl [17] used branch automata to check the degree of unambiguity of finite tree automata.

Proposition 11. *A deterministic tree automaton is not left closing if and only if there is a computation in the square automaton ending in a diagonal state and containing a non diagonal one.*

Proof. By definition of $\mathcal{A} \times \mathcal{A}$, the existence of a computation in $\mathcal{A} \times \mathcal{A}$ ending in a state (p, p) and containing a state (r, s) with $r \neq s$ is equivalent to the existence of two distinct computations of $\mathcal{A}$ on a same tree.

In order to check the above property, we build *the pair graph* $G_{\mathcal{A}} = (V_G, E_G)$ of $\mathcal{A}$, where $V_G \subseteq (V^2 \times V^2) \cup V^2$ is the set of vertices, $E_G \subseteq V_G \times \{0, 1\} \times A \times V_G$ is the set of edges labeled by 0 or 1 and a letter from A. For more convenience, an edge labeled by 1 is noted by a plain arrow $\longrightarrow$ and is called a plain edge, and an edge labeled by 0 is noted by a dashed arrow $\dashrightarrow$ and is called a dashed edge. For each pair of transitions $(p, q), a \to r$ and $(p', q'), a \to r'$ of $\mathcal{A}$,

$$
\begin{aligned}
((r, r'), (s, s')) &\xrightarrow{0,a} ((p, p'), (q, q')),\\
((s, s'), (r, r')) &\xrightarrow{1,a} ((p, p'), (q, q')),\\
(r, r') &\xrightarrow{0,a} ((p, p'), (q, q')),\\
(r, r') &\xrightarrow{1,a} ((p, p'), (q, q')),
\end{aligned}
$$

are edges of $G_{\mathcal{A}}$, for each pair (s, s').

A vertex of $G_{\mathcal{A}}$ is *useful* if it has at least one outgoing plain edge and at least one outgoing dashed edge. We keep the *essential part* of the pair graph obtained by discarding vertices which are not useful, and their incoming and outgoing edges. A vertex $((p, q), (r, s))$ of G_A is called *non diagnonal* if either $p \neq q$ or $r \neq s$.

Proposition 12. *A deterministic tree automaton is not left closing if and only its there is a path in its pair graph starting at a vertex (p, p) and ending in a non diagonal vertex.*

The number of vertices of $G_{\mathcal{A}}$ is at most $O(|V|^4)$ and its number of edges of $G_{\mathcal{A}}$ is at most $O(|V|^6)$. The property of Proposition 12 can be checked in a linear time in the size of $G_{\mathcal{A}}$. As a consequence, it can be checked in polynomial

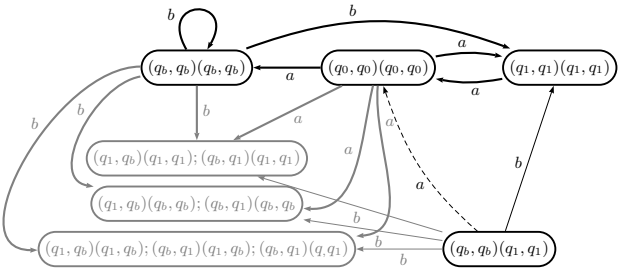


Fig. 3. The pair graph for the tree automaton of Example 1. A thick edge represents a plain edge and a dashed edge with the same label. The non useful edges and vertices are drawn in grey. Each vertex (p, q) is identified with the vertex $(p, q)(p, q)$. For the test, the pair $((p, q), (r, s))$ may not be represented if $((r, s), (p, q))$ does. The tree-shift X accepted by $\mathcal{A}$ satisfies the property of Proposition 12 since there is no path from a diagonal state to a non diagonal one. Then $\mathcal{A}$ is a left closing automaton and as a consequence the tree-shift X is AFT.

time whether the Shannon cover of an irreducible sofic tree-shift is AFT. Note that Seidl's check of the finite degree of ambiguity of tree automata in [17] has a similar complexity (the cube of the size of the transitions of the tree automaton). The pair graph for the tree automaton $\mathcal{A}$ of Example 1 is given in Figure 3.

6 Conclusion

In this article, we have shown that tree-shifts differ from one-sided shifts of infinite sequences at least concerning the following property: there may be more than one minimal deterministic irreducible tree automata accepting the same irreducible sofic tree-shift. The reason is that such automata do not always have a synchronizing block. For irreducible sofic tree-shifts, the Shannon cover remedy for this lack and allows us to define the class of almost finite type tree-shifts. In further work we will focus on topological and syntactic properties of AFT tree-shifts.

References

1. Aubrun, N., Béal, M.-P.: Decidability of conjugacy of tree-shifts of finite type. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 132–143. Springer, Heidelberg (2009)
2. Bates, T., Eilers, S., Pask, D.: Reducibility of covers of AFT shifts. Israel Journal of Mathematics (to appear, 2010)
3. Béal, M.-P., Fiorenzi, F., Perrin, D.: A hierarchy of shift equivalent sofic shifts. Theor. Comput. Sci. 345(2-3), 190–205 (2005)
4. Boyle, M., Kitchens, B., Marcus, B.: A note on minimal covers for sofic systems. Proc. Amer. Math. Soc. 95(3), 403–411 (1985)

5. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007), http://www.grappa.univ-lille3.fr/tata (release October 12, 2007)
6. Coven, E., Johnson, A., Jonoska, N., Madden, K.: The symbolic dynamics of multidimensional tiling systems. Ergodic Theory and Dynamical Systems 23(02), 447–460 (2003)
7. Fujiwara, M., Osikawa, M.: Sofic systems and flow equivalence. Math. Rep. Kyushu Univ. 16(1), 17–27 (1987)
8. Johnson, A.S.A., Madden, K.M.: The decomposition theorem for two-dimensional shifts of finite type. Proc. Amer. Math. Soc. 127(5), 1533–1543 (1999)
9. Jonoska, N., Marcus, B.: Minimal presentations for irreducible sofic shifts. IEEE Trans. Inform. Theory 40(6), 1818–1825 (1994)
10. Kitchens, B.P.: Symbolic dynamics. In: Universitext. Springer, Berlin (1998); One-sided, two-sided and countable state Markov shifts
11. Lind, D., Schmidt, K.: Symbolic and algebraic dynamical systems. In: Handbook of dynamical systems, vol. 1A, pp. 765–812. North-Holland, Amsterdam (2002)
12. Lind, D.A., Marcus, B.H.: An Introduction to Symbolic Dynamics and Coding. Cambridge University Press, Cambridge (1995)
13. Marcus, B.H.: Sofic systems and encoding data. IEEE Transactions on Information Theory 31(3), 366–377 (1985)
14. Nasu, M.: Textile Systems for Endomorphisms and Automorphisms of the Shift. American Mathematical Society, Providence (1995)
15. Perrin, D., Pin, J.: Infinite words. Elsevier, Boston (2004)
16. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press, Cambridge (2009)
17. Seidl, H.: On the finite degree of ambiguity of finite tree automata. In: Csirik, J.A., Demetrovics, J., Gecseg, F. (eds.) FCT 1989. LNCS, vol. 380, pp. 395–404. Springer, Heidelberg (1989)
18. Thomas, W.: Automata on infinite objects. In: Handbook of theoretical computer science, vol. B, pp. 133–191. Elsevier, Amsterdam (1990)
19. Williams, R.F.: Classification of subshifts of finite type. In: Recent advances in topological dynamics (Proc. Conf. Topological Dynamics, Yale Univ., New Haven, Conn.). Lecture Notes in Math., vol. 318, pp. 281–285. Springer, Berlin (1973)
20. Williams, S.: Covers of non-almost-finite type sofic systems. Proc. Amer. Math. Soc. 104(1), 245–252 (1988)

Proof-Based Design of Security Protocols

Nazim Benaissa and Dominique Méry

Université Henri Poincaré Nancy 1, LORIA, BP 239, 54506 Vandœuvre-lès-Nancy, France

Abstract. We consider the refinement-based process for the development of security protocols. Our approach is based on the Event B refinement, which makes proofs easier and which makes the design process faithfull to the structure of the protocol as *the designer thinks of it*. We introduce the notion of mechanism related to a given security property; a mechanism can be combined with another mechanism through the double refinement process ensuring the preservation of previous security properties of mechanisms. Mechanisms and combination of mechanisms are based on Event B models related to the security property of the current mechanism. Analysing cryptographic protocols requires precise modelling of the attacker's knowledge and the attacker's behaviour conforms to the Dolev-Yao model.

1 Introduction

1.1 Analysing Cryptographic Protocols

Cryptographic protocols are complex software systems; they are, therefore, in need of high level modelling tools and concepts. They have also underlying desired properties, which can be expressed logically like secrecy, authentication. Formal methods are widely used for cryptographic protocols as a verification tool, not as a design tool. The goal of this paper is to present an attempt to mix the two: design, predominant in software engineering, and formal methods. We introduce *mechanisms* which are ensuring a property and are characterized by a Event B models; we show how to combine these mechanisms by applying a double refinement process which guarantees the preservation of the basic properties of the underlying mechanisms. This leads to the notion of proof-based design allowing a correct-by-construction approach to security protocols.

Our approach is based on incremental development using refinement. The refinement of a formal model allows us to enrich a model in a *step-by-step* approach, and is the foundation of our *correct-by-construction* approach. Refinement is used to make proofs easier but also to make the design process faithfull to the structure of the protocol as the designer thinks of it. Implicit assertions are identified for proving the protocol secure at each refinement step.

To be able to prove security properties on a protocol, we must be able to *model the knowledge of the attacker*. A *pet* model of attacker's behaviour is the Dolev-Yao model [9]; this model is an *informal* description of all possible behaviours of the attacker as described by N. Benaissa [3]. Hence, we present a

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 25–36, 2010.

proof-based design to model and prove key protocols using Event B [1,7] as a modelling language. We give a presentation of a proof-based design framework and an application of the proof-based design on a simplified version of the Kerberos protocol [19] with smart cards.

Proving properties on cryptographic protocols such as *secrecy* is known to be undecidable. However, works involving formal methods for the analysis of security protocols have been carried out. Theorem provers or model checkers are usually used for proving properties. For model checking, one famous example is Lowe's approach [12] using the process calculus CSP and the model checker FDR. Lowe discovered the famous bug in Needham-Schroeder's protocol. Model checking is efficient for discovering an attack if there is one, but it can not guarantee that a protocol is reliable with respect to a list of given assumptions on the environment or on the possible attacks. We should be carefull on the question of stating properties of a given protocol and it is clear that the modelling language should be able to state a given property and then to check the property either using model checking or theorem proving. Other works are based on theorem proving: Paulson [14] used an inductive approach to prove safety properties on protocols. He defined protocols as sets of traces and used the theorem prover Isabelle.

1.2 Proof-Based Guidelines

Designers can now justify design decisions (and the claims for their designs of being high quality) based on the patterns that they have (or have not) applied. In fact, much like the quality of code can easily be judged by a quick glance at whether certain coding standards have been followed, so the quality of a design can now be judged based on the use of well-understood patterns (and architectures) [10]. Good software design tools have given rise to the notion of design patterns - where expertise is wrapped up in re-usable form and support provided for the (semi-automated) application of these patterns.

Patterns and design patterns [10] provide a very convenient help in the design of object-oriented software. Recently, J.-R. Abrial [1] suggested the introduction of a kind of patterns for the proof-based development. The action/reaction patterns have been applied to the press case study by J.-R. Abrial and they improve the proof process. Another pattern called re-usability pattern, has been suggested by Abrial, Cansell and Méry [2] and applied to the development of voting systems [6]. In previous works [4], we have already analysed cryptographic protocols and the use of Event B without proposing a genera technique for composing mechanisms. Clearly, a growing activity on modelling patterns has started and is addressing many kinds of case studies and domains of problems. No classification is yet given and there is no real repository of patterns validated for a specific modelling language based on proof-based transformations.

We postulate that *proof-based designs* or *proof-based design patterns* will play a vital role in any future measurements with respect to claims of security and trustworthiness based on verification. The key point is that proof-based design patterns have a very important feature: they are based on an objective way to

ensure the validity of the resulting objects. However, we have no formal definition of what is a proof-based design pattern.

1.3 Summary of the Paper

Section 2 contains definitions and notions related to cryptographic protocols. Section 3 is the heart of the paper and defines the proof-based pattern using the hierarchy of properties of authentication and key establishment goals. In section 4, an example of the pattern application is presented. Finally, we conclude the paper.

2 Principles for Modelling the Protocols

Our goal is to define a proof-based guideline or a pattern for modelling cryptographic protocols using Event B. The pattern is defined by a proof-based development of Event B models which are modelling protocols first using a very abstract model followed by several refinements. The definition of models is based on the notion of transaction. The choice of the details added in each refinement is crucial, this choice is guided by several criteria. First, introducing refinement in general helps to make proofs easier and increases the automatic proofs rate. Second, refinement introduces automation in the design process: in each refinement step, assertions for proving the protocol correct are generated and have to be proved. The hierarchy of the properties to prove corresponds to the hierarchy of the chosen refinements. Boyd and Mathuria [8] give a hierarchy of some important properties that have to be proved on cryptographic protocols in general and properties related with key establishment protocols (Figure 1). Far-end operative, peer knowledge, key freshness and key secrecy are generic properties, while the other properties can be obtained by combining these generic ones.

There are several different definitions of security properties in the literature we present here a summary of these definitions given by [8]. These properties can be divided into two categories, user oriented goals and key oriented goals :

- User-oriented goals include:
  - Far-end operative property: A believes B recently said something.
  - Knowledge of peer: A is aware of B as her claimed peer entity in the protocol.

  Combining these two generic properties leads to the *entity authentication* property.
- Key-oriented goals include key freshness and key secrecy.
  - Key freshness: The key is fresh.
  - key secrecy: The key is known only to A, B and any mutually trusted parties.

  Combining these two generic properties leads to *good key* property.

If user-oriented goals and key oriented goals are combined as shown in figure 1, we can obtain other enhanced goals:

- Key confirmation: Key confirmation of A to B is provided if B has assurance that key K is a good key to communicate with A and that principal has possession of K.
- Mutual belief in key: Mutual belief in the key K is provided for B only if K is a good key for use with A and A wishes to communicate with B using key K which A believes is good for the purpose.

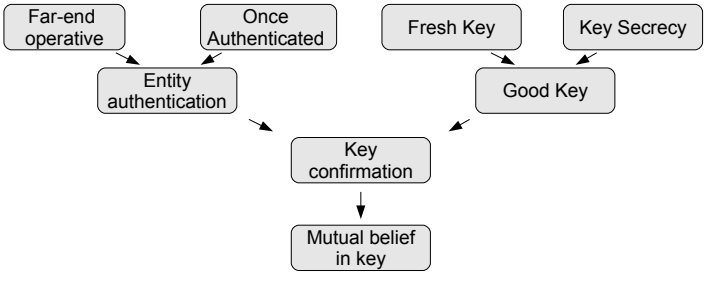


Fig. 1. Hierarchy of authentication and key establishment goals

Key secrecy also known as key authentication means that a shared key is known only by the agents sharing the key and any mutually trusted third party.

The basic idea of our approach is to make the incremental development of the cryptographic protocol faithfull to the structure of the protocol as the engineer thinks of it. What we want to do is to identify all the mechanisms that let a protocol satisfy each safety property, and then combine them by refinement to obtain the final protocol. The general approach is described as follows. We start by identifying properties of the final protocol and decompose these properties to obtain the basic properties ensuring the required properties. Each basic property is validated by a mechanism through a Event B development starting by a first abstract machine and completed by a final machine integrating elements on the protocol itself and on the possible attacks. Then the combination of mechanisms is driven by the target property and is validated by a double refinement. Modelling attacker's knowledge is an important issue, a set of variables `Attack_Know` is used to model this knowledge. As we will see in subsection 3.1, to prove that a mechanism satisfies a safety property, we need to introduce an invariant `I(Attack_Know)` that contains a characterisation of the attacker's knowledge that allow us to prove the desired property. When two mechanism are combined the attacker is more powerfull and has more knowledge since more information are added to the variables in the set `Attack_Know` by events modelling attacker's behaviour of both models together. We need then to prove that `Attack_Know` variable (shared by both mechanism models) still satisfy the invariants of both models.

3 The Pattern Structure

Properties in the figure 1 are defined in an abstract way in Event B using the notion of abstract transactions that will be introduced in subsection 3.1. For each property, a set of mechanisms that guarantee this property are available described with Event B models. Let $\mathcal{P}$ the set of properties and $\mathcal{M}$ the set of Event B models of mechanisms. The figure 2 summarizes relationships among Event B models and properties and we define now definitions for expressing conditions over mechanisms and for defining composition of mechanisms, properties and models. For a given attacker's model, a relation $\rightsquigarrow$ is defined over the sets $\mathcal{P}$ and $\mathcal{M}$:

Definition 1. $\forall\, p, m.\ p \in \mathcal{P} \wedge m \in \mathcal{M}$*:* $m \rightsquigarrow p$ *iff there exists a machine* n *such that* p *is a theorem of the machine* n *and* m *is a refinement of* n*. We say that* m *implements the property* p*.*

Event B models will be defined in the subsection 3.1 and we will formally introduce how properties and mechanisms are modelled in Event B . We will also see how a mechanism is proved to implement a property.

Let us consider an example of a mechanism using shared key cryptography that implements the authentication property. The mechanism 1 is contained in the *ISO/IEC 9798-2 two-pass unilateral authentication protocol* from the international standard *ISO/IEC 9798 Part 2* [18]:

1. $B \rightarrow A : N_b$	1. $A \rightarrow B : N_a$
2. $A \rightarrow B : \{N_b, B\}_{K_{AB}}$	2. $B \rightarrow A : \{N_a, B\}_{K_{AB}}$
Mechanism 1	**Mechanism 2**

In the mechanism 1, B has knowledge of A as her peer entity, we proved that this mechanism implements the knowledge of peer property in the Dolev-Yao attacker model. If the nonce N_b is fresh we can also prove that the mechanism implements the Far-end operative property. Combining two mechanisms is not always possible, for example, combining the mechanism 1 with the mechanism 2 where A has knowledge of B as his peer entity may not be possible: let us consider a possible composition of the two previous mechanisms given in the new mechanism 3:

1. $A \rightarrow B : N_a$	1. $I_A \rightarrow B : N_i$
2. $B \rightarrow A : \{N_a, B\}_{K_{AB}}, N_b$	2. $B \rightarrow I_A : \{N_i, B\}_{K_{AB}}, N_b$
3. $A \rightarrow B : \{N_b, B\}_{K_{AB}}$	3. $I_B \rightarrow A : N_b$
	4. $A \rightarrow I_B : \{N_b, A\}_{K_{AB}}, N_a$
	5. $I_A \rightarrow B : \{N_b, A\}_{K_{AB}}$
Mechanism 3	**Attack 1**

We may think that this new mechanism provides both properties of the composed ones but a possible attack on the new obtained mechanism is shown in the attack 1. Both mechanisms when considered separately implement the authentication property but when we combine them, the attacker can use one mechanism to attack the other as shown in the attack 1. We identified for each mechanism a set of conditions that guarantee that a combination of this mechanism with others is possible and a set of generated proof obligations have to be discharged.

We also introduce a function $\mathcal{R}$ that maps two models $m1$ and $m2$ to the set of possible mechanisms obtained by combining these two mechanisms in a correct way (proof obligations have been discharged):

Definition 2. $\forall\ m1, m2, m.\ m1 \in \mathcal{M} \wedge m2 \in \mathcal{M} \wedge m \in \mathcal{M}$*:* $m \in \mathcal{R}(m1, m2)$ *iff* m *is a double refinement of* $m1$ *and* $m2$ *ie* m *refines* $m1$ *and* m *refines* $m2$.

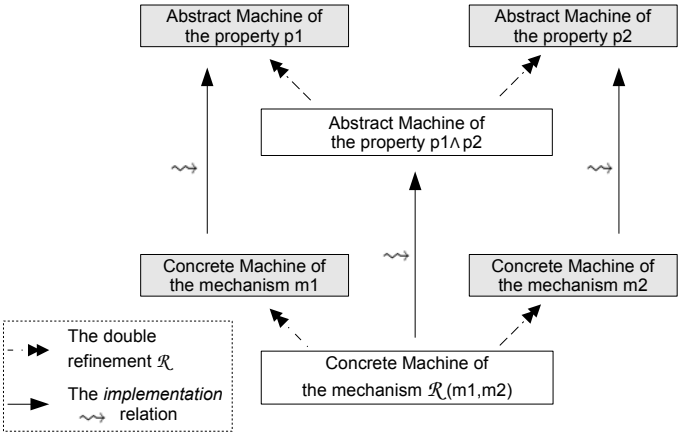


Fig. 2. Structure for composing mechanisms

The concrete Event B machine of the mechanism $\mathcal{R}(m1, m2)$ in the figure 2 is a double refinement ($\mathcal{R}$) of the two Event B models corresponding to the two mechanisms $m1$ and $m2$. The basic idea of our approach is to start with mechanisms that implements ($\rightsquigarrow$) generic properties expressed in the abstract Event B machines (figure 2) and to combine them by a double refinement process to implement more complex properties as shown in figure 1. We have proved the following theorem that we will use to combine mechanisms:

Theorem 1. $\forall\ m, m1, m2, p1, p2. m \in \mathcal{M} \wedge m1 \in \mathcal{M} \wedge m2 \in \mathcal{M} \wedge p1 \in \mathcal{P} \wedge p2 \in \mathcal{P}$*:* *If* $m1 \rightsquigarrow p1 \wedge m2 \rightsquigarrow p2 \wedge m \in \mathcal{R}(m1, m2)$*, then* $m \rightsquigarrow p1 \wedge p2$

When proving a protocol, we will need to prove only the combination proof obligations and not the proofs of different mechanisms that are done only once and can be reused for different protocols. The last definition to introduce before presenting the pattern is the instance of a mechanism. We may need to use several instances of the same mechanism. From an Event B point of view two

instances of the same mechanism model are simply two models where variables are renamed. These models have exactly the same behaviour and satisfy the same properties. Two instances of the same mechanism are linked with a $\sim$ relation defined as follows:

Definition 3. $\forall\ m1, m2.\ m1 \in \mathcal{M} \wedge m2 \in \mathcal{M}$*:*
$m2 \sim m1$ *iff* $m2$ *is obtained by renaming the variables of* $m1$*.*

Theorem 2. $\forall\ m1, m2, p. m1 \in \mathcal{M} \wedge m2 \in \mathcal{M} \wedge p \in \mathcal{P}$*:*
If $m1 \sim m2 \wedge m2 \rightsquigarrow p$ *then* $m1 \rightsquigarrow p$

Using our design patterns has two advantages:

- To have an efficient refinement (in term of automatic proofs), we need to make the right choices when choosing abstraction levels and variables during the modelling process. When using the pattern, the designer of the cryptographic protocol will have to decide only how to combine mechanism to obtain the desired protocol since the mechanisms are already modelled in Event B.
- Proofs of already proved mechanisms are done once and can be reused with different protocols and proof obligations for combining them are defined.

3.1 Event-B Models of the Mechanisms

We present the Event B models of the mechanism 1. Our goal is to prove that a mechanism m implements ($\rightsquigarrow$) a certain property p and also to identify the proof obligation of correct composition of this mechanism with others. To prove that a mechanism m satisfies a given property P we need one abstract model and two refinement steps:

- The first model is the specification, the desired property p is stated in an abstract way using the notion of abstract transaction that will be introduced later in this section. The way a property is expressed in this first abstract model is common to all the mechanisms.
- The second model is the implementation, we exhaustively add all details used by the mechanism m to guarantee the desired property stated in the previous model. The RODIN tool will then automatically generate proof obligations by respect to the property stated in the abstract model.
- Third model: we model the behaviour of the attacker. The attacker knowledge is injected to the previous model. We introduce attacker's knowledge in a separate refinement so we can apply several attacker's behaviours to see if we still can prove safety properties.

Abstract Model. At this abstraction level, we introduce the type of `AGENT`, that is common to all kinds of cryptographic protocols, other notions like nonces, timestamps or cryptographic keys are specific to each kind of protocols and will be introduced in further refinements. To model the different properties, the

pattern is based on the notion of *abstract transactions*. An *abstract transaction* (type T in the models) is a session of the protocol performed by one of the agent involved in the protocol run. In some cryptographic protocols, nonces are used to identify each session or protocol run. Intuitively, each transaction of the abstract model will correspond, in this case, to a fresh nonce (or to a timestamp in other protocols) in the concrete model. The intruder is a particular agent: $I \in AGENT$, note that for most protocols, even if there is more than one dishonnest agent in the system, it suffices to consider only one attacker that will combine the abilities and knowledge of all the other dishonnest agents.

There are several definitions in the literature of entity authentication in cryptographic protocols, Syverson and van Oorschot [16] define entity authentication as: *"A believes B recently replied to a specific challenge."* This property is obtained by combining two generic properties: peer knowledge and far-end operative. In this first model we focus on peer knowledge property. To be able to model it, we introduce variables that model each protocol run (one abstract transaction in the abstract model) attributes. An abstract transaction has a source (t_src) that is the agent that initiated the transaction and a *believed* destination (t_bld_dst) that is the believed destination agent. A running transaction is contained in a set $trans$. When a transaction terminates it is added to a set end, *to prove peer knowledge, we need to prove that both variables are equal when a transaction terminates.*

Theorem 3. $\forall t \cdot t \in end \wedge t\_src(t) \neq I \Rightarrow t\_dst(t) = t\_bld\_dst(t)$

First Refinement. The goal of this first refinement is to understand how the property stated in the previous model is achieved, thus, the corresponding details of the modelled mechanism messages are exhaustively added. Cryptographic keys are introduced in this refinement. For example shared keys are modelled as follows: KEY is the set of all pair keys; $KEY\_A \in KEY \rightarrow AGENT$ contains the first owner agent; $KEY\_B \in KEY \rightarrow AGENT$ is the second owner agent.

For example, to model the attributes of the message in the step *2* of the mechanism 1, we need the following variables: $answer\_Na \in answer \rightarrow T$ that models the the encrypted nonce; $answer\_KAB \in answer \rightarrow KEY$ that models the used encryption key and also $answer\_A \in answer \rightarrow AGENT$ that contains the agent identity included in the message.

Second refinement: attacker's knowledge. To be able to prove properties such as secrecy and authentication on a protocol, we have to be able to model the knowledge of the attacker. To model the knowledge of the attacker, it is necessary to know exactly what the attacker is able to do. This refinement models all the options the attacker has in the Dolev-Yao attacking model and can be reused for different protocols. To model all these options, we use a set of variables $Attack\_Know$ that contains the crucial information the attacker can obtain. Because of the typing constraints in the Event B, we use one variable for each information type : N_Mem for nonces and K_Mem for each type of keys: $N\_Mem \subseteq \mathbb{P}(T)$ and $K\_Mem \subseteq \mathbb{P}(KEY)$.

The attacker can also use fragments of encrypted messages contained in the communication channel, we model the set of fragments available to the attacker using a variable FRAG. In the case of mechanism 1, the fragment has the following structure.

$$
\begin{array}{l}
\mathtt{FRAG} \subseteq \mathtt{MSG} \\
\mathtt{FRAG\_Na} \in \mathtt{FRAG} \rightarrow \mathtt{T} \\
\mathtt{FRAG\_KAB} \in \mathtt{FRAG} \rightarrow \mathtt{KEY} \\
\mathtt{FRAG\_A} \in \mathtt{FRAG} \rightarrow \mathtt{AGENT} \\
\mathtt{FRAG\_Src} \in \mathtt{FRAG} \rightarrow \mathtt{AGENT}
\end{array}
$$

We added EVENT B events modelling all the possible attacker behaviours according to the Dolev-Yao model and we automatically generated a gluing invariant to prove that the refinement is correct using an invariant strengthening technique. In the case of the mechanism 1, we obtained the following invariants:

$$
\begin{array}{l}
\forall\, \mathtt{K}, \mathtt{A}, \mathtt{B}.\ \mathtt{K} \in \mathtt{K\_MEM} \wedge \mathtt{K} \in \mathtt{dom}(\mathtt{KEY\_A}) \wedge \neg \mathtt{A} = \mathtt{I} \wedge \\
\quad ((\mathtt{KEY\_A}(\mathtt{K}) = \mathtt{A} \wedge \mathtt{KEY\_B}(\mathtt{K}) = \mathtt{B}) \vee (\mathtt{KEY\_A}(\mathtt{K}) = \mathtt{B} \wedge \mathtt{KEY\_B}(\mathtt{K}) = \mathtt{A})) \\
\qquad \Rightarrow \mathtt{B} = \mathtt{I}
\end{array}
$$

This first invariant states that to preserve the peer knowledge property, the attacker should only posses keys he share with another agent. The second theorem is as follows:

$$
\begin{array}{l}
\forall\, \mathtt{K}, \mathtt{A}, \mathtt{B}, \mathtt{frag}.\ \mathtt{frag} \in \mathtt{FRAG} \wedge \mathtt{FRAG\_A}(\mathtt{frag}) = \mathtt{A} \wedge \mathtt{FRAG\_Key}(\mathtt{frag}) = \mathtt{K} \wedge \\
\quad ((\mathtt{KEY\_A}(\mathtt{K}) = \mathtt{A} \wedge \mathtt{KEY\_B}(\mathtt{K}) = \mathtt{B}) \vee (\mathtt{KEY\_A}(\mathtt{K}) = \mathtt{B} \wedge \mathtt{KEY\_B}(\mathtt{K}) = \mathtt{A})) \wedge \\
\quad \neg \mathtt{A} = \mathtt{I} \wedge \mathtt{K} \in \mathtt{dom}(\mathtt{KEY\_A}) \\
\qquad \Rightarrow \mathtt{B} = \mathtt{FRAG\_Src}(\mathtt{frag})
\end{array}
$$

Intuitively this invariant states that if a fragment is encrypted with a key owned by two agents and the identity of one of these agent is in the field FRAG_A, then the source of this fragment is the other agent. These two invariants are very important for the composition of mechanisms. If this mechanism is composed with another one we need to prove that the fragments of the same type generated by the other mechanism satisfy these theorems. This is how proof obligations of mechanisms composition are generated. When we tried to compose mechanisms 1 and 2 we could not prove these invariants and we could generate the attack 1. In general, attacker's knowledge contained in the variables N_Mem, K_Mem and the fragments of the same structure is shared between the combined mechanisms. To prove that a composition is correct we need to prove that events of each mechanism model maintain the invariants that characterise the attacker's knowledge of the other mechanism.

3.2 Describing the Pattern

The pattern is organized in three modules:

– The LIBRARY module: contains the available properties and the mechanisms that are proved to implement them.

– The COMPOSITION module: contains the structure of the protocol and shows how it was obtained by composing the different mechanisms. This module has three clauses:
  1. The PROPERTIES clause: contains the properties that the protocol should satisfy.
  2. The MECHANISMS clause: contains the instances of the used mechanisms in the protocol.
  3. The THEOREM clause: shows the $\rightsquigarrow$ relation of the instances of mechanisms used in the protocol.
– The B_MODELS clause: contains the EVENT B models of the mechanisms.

The pattern can be used for two purposes, designing new protocols and analysing existing ones. By analysing a protocol we mean proving it by identifying which component of the protocol guarantees each property and possibly identify any unnecessary component of the protocol.

4 An Example of the Pattern Application

We applied our design pattern on different cryptographic protocols among which: Blake-Wilson-Menezes key transport protocol [5], the well known Needham-Schroeder public key protocol [13] and the Shoup-Rubin key distribution protocol [11]. We will illustrate our approach in this paper on a simplified version (as shown in protocol 1) of the well known Kerberos protocol [19].

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{K_{AB}, B, N_a\}_{K_{AS}}, \{K_{AB}, A\}_{K_{BS}}$
3. $A \rightarrow B : \{A, T_A\}_{K_{AB}}, \{K_{AB}, A\}_{K_{BS}}$

Protocol 1. *A simplified version of the Kerberos protocol*

The basic protocol involves three parties, the client (A), an application server (B) and an authentication server (S). A secret long term key K_{AS} is shared between A and S and another key K_{BS} is shared between B and S. This simplified protocol provides the properties.

	Property	Agent
$p1_A$	Key authentication	A
$p1_B$	Key authentication	B
$p2_A$	Key freshness	A
$p3_B$	far-end operative	B

Three mechanisms are used to guarantee these properties. The first one (mechanism 4) is a part of the $ISO/IEC11770\ Part2$ [17]. The second mechanism is similar to mechanism 1 where the server S is involved, but the identity of agent A in the response message is no longer necessary since reflection attack is not possible anymore. The last mechanism uses time stamps:

1. $S \rightarrow A : \{K_{AB}, B\}_{K_{AS}}$
1. $S \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$

Mechanism 4

1. $A \rightarrow S : N_a$
2. $S \rightarrow A : \{N_a\}_{K_{AS}}$

Mechanism 5

1. $A \rightarrow B : \{A, T_a\}_{K_{AB}}$

Mechanism 6

Table 1. The design pattern for Kerberos protocol

LIBRARY	COMPOSITION	B-MODELS
	PROPERTIES	
PROPERTIES	$< p1_A >< p1_B >< p2_A >< p3_B >$	
$< p1_A >< p1_B >$	**MECHANISM**	
$< p2_A >< p3_B >$	$m4_1 \sim m4$	
...	$m5_1 \sim m5$	$m4_1 = B\ models$
MECHANISM	$m45_1 \in \mathcal{R}(m4_1, m5_1)$	$m5_1 = B\ models$
$m4 \rightsquigarrow p1_A$	$m6_1 \sim m6$	$m45_1 = B\ models$
$m4 \rightsquigarrow p1_B$	$m456_1 \in \mathcal{R}(m45_1, m6_1)$	$m6_1 = B\ models$
$m5 \rightsquigarrow p2_A$	**THEOREM**	$m456_1 = B\ models$
$m6 \rightsquigarrow p3_B$	$m4_1 \rightsquigarrow p1_A \wedge p1_B$	
...	$m5_1 \rightsquigarrow p2_A$	
	$m45_1 \rightsquigarrow (p1_A \wedge p1_B) \wedge p2_A$	
	$m456_1 \rightsquigarrow ((p1_A \wedge p1_B) \wedge p2_A) \wedge p3_B$	

The table 1 shows how the pattern is applied, but due to lack of space we will skip the Event B models, in section 3.1 contains the Event B model of one mechanism taken as an example. Mechanisms are introduced one by one in the pattern and proofs of double refinement are generated and discharged.

We emphasize that this simplified version of Kerberos protocol does not satisfy key freshness property for agent B. The full Kerberos protocol uses a mechanism of expiration time in the message intended to B to fulfil this property.

5 Conclusion

We have introduced an Event-B-based design pattern for cryptographic protocols and we have applied it on three different protocols. Several properties were proved on these protocols, user-oriented and key-oriented properties. Less than 10% of the proofs of the models were interactive. Patterns facilitate proof process by reusing partially former developments; we have not yet designed new cryptographic protocols and it remains to develop other case studies by applying patterns. Like design patterns, proof-based patterns are based on real cases; they should help the use of refinement and proof techniques; it is then clear that specific tools should be developed and further works should be carried out using refinement for discovering new patterns. As a perspective of our work, we want to model more mechanisms and define a plugin of the RODIN tool that implements this pattern.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2009) (forthcoming book)
2. Abrial, J.-R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: Application to event-b. Fundam. Inform. 77(1-2), 1–28 (2007)

3. Benassa, N.: Modelling attacker's knowledge for cascade cryptographic protocols. In: Börger, E., Butler, M., Bowen, J.P., Boca, P. (eds.) ABZ 2008. LNCS, vol. 5238, pp. 251–264. Springer, Heidelberg (2008)
4. Benaïssa, N., Méry, D.: Cryptologic protocols analysis using Event B. In: Marchuk, A. (réd.) Seventh International Andrei Ershov Memorial Conference Perspectives of System Informatics, Novosibirsk, Akademgorodok, Russia, June 15-19. LNCS. Springer, Heidelberg (2010)
5. Blake-Wilson, S., Menezes, A.: Entity authentication and authenticated key transport protocols employing asymmetric techniques. In: Christianson, B., Lomas, M. (eds.) Security Protocols 1997. LNCS, vol. 1361, pp. 137–158. Springer, Heidelberg (1998)
6. Cansell, D., Paul Gibson, J., Méry, D.: Refinement: A constructive approach to formal software design for a secure e-voting interface. Electr. Notes Theor. Comput. Sci. 183, 39–55 (2007)
7. Cansell, D., Méry, D.: The Event B Modelling Method: Concepts and Case Studies, pp. 33–140. Springer, Heidelberg (2007)
8. Mathuria, A., Boyd, C.: Protocols for Authentication and Key Establishment (2003)
9. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2), 198–208 (1983)
10. Gamma, E., Helm, R., Johnson, R., Vlissides, R., Gamma, P.: Design Patterns: Elements of Reusable Object-Oriented Software Design Patterns. Addison-Wesley Professional Computing, Reading (1997)
11. Jerdonek, R., Honeyman, P., Coffman, K., Rees, J., Wheeler, K.: Implementation of a provably secure, smartcard-based key distribution protocol. In: Schneier, B., Quisquater, J.-J. (eds.) CARDIS 1998. LNCS, vol. 1820, pp. 229–235. Springer, Heidelberg (2000)
12. Lowe, G.: Breaking and fixing the needham-schroeder public-key protocol using fdr. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
13. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM 21(12), 993–999 (1978)
14. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6, 85–128 (1998)
15. Project RODIN. The rodin project: Rigorous open development environment for complex systems, http://rodin-b-sharp.sourceforge.net/
16. Syverson, P.F., Van Oorschot, P.C.: On unifying some cryptographic protocol logics. In: SP '94: Proceedings of the 1994 IEEE Symposium on Security and Privacy, Washington, DC, USA, p. 14. IEEE Computer Society, Los Alamitos (1994)
17. ISO. Information technology - Security techniques - Key management - Part 2: Mechanisms Using Symmetric techniques ISO/IEC 11770-2, International Standard (1996)
18. ISO. Information technology - Security techniques - Entity authentication - Part 2: Mechanisms Using Symmetric Encipherment Algorithms ISO/IEC 9798-2, 2nd edn. International Standard (1999)
19. Clifford Neuman, B., Ts'o, T.: Kerberos: An authentication service for computer networks. IEEE Communications magazine 32(9), 33–38 (1994)

Approximating the Minimum Length of Synchronizing Words Is Hard

Mikhail V. Berlinkov

Department of Algebra and Discrete Mathematics
Ural State University
620083 Ekaterinburg, Russia
berlm@mail.ru

Abstract. We prove that, unless P = NP, no polynomial-time algorithm can approximate the minimum length of synchronizing words for a given synchronizing automaton within a constant factor.

Background and Overview

Let $A = \langle Q, \Sigma, \delta \rangle$ be a complete *deterministic finite automaton* (DFA), where Q is the state set, Σ is the input alphabet, and $\delta : Q \times \Sigma \to Q$ is the transition function. The function δ extends uniquely to a function $Q \times \Sigma^* \to Q$, where Σ^* stands for the free monoid over Σ; the latter function is still denoted by δ. Thus, each word in Σ^* acts on the set Q via δ. The DFA A is called *synchronizing* if there exists a word $w \in \Sigma^*$ whose action resets A, that is, leaves the automaton in one particular state no matter which state in Q it starts at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$. Any such word w is called a *synchronizing word* for A. The minimum length of synchronizing words for A is denoted by $\min_{synch}(A)$.

Synchronizing automata serve as transparent and natural models of error-resistant systems in many applications (coding theory, robotics, testing of reactive systems) and also reveal interesting connections with symbolic dynamics and other parts of mathematics. For a brief introduction to the theory of synchronizing automata we refer the reader to the recent survey [10]. Here we discuss only some complexity-theoretic issues of the theory. In the following we assume the reader's acquaintance with some basics of computational complexity that may be found, e.g., in [3,6].

There is a polynomial-time algorithm (basically due to Černý [1]) that decides whether or not a given DFA is synchronizing. In contrast, determining the minimum length of synchronizing words for a given synchronizing automaton is known to be computationally hard. More precisely, deciding, given a synchronizing automaton A and a positive integer ℓ, whether or not $\min_{synch}(A) \leq \ell$ is NP-complete [2,5,9,8]. Moreover, deciding, given the same instance, whether or not $\min_{synch}(A) = \ell$ is both NP-hard and co-NP-hard [8]. Thus, unless NP = co-NP, even non-deterministic algorithms cannot find the minimum length of synchronizing words for a given synchronizing automaton in polynomial time.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 37–47, 2010.

There are some polynomial-time algorithms that, given a synchronizing automaton, find synchronizing words for it, see [2,7]. Such algorithms can be considered as approximation algorithms for calculating the minimum length of synchronizing words but it seems that they have not been systematically studied from the approximation viewpoint. Experiments show that Eppstein's greedy algorithm [2] behaves rather well on average and approximates $\min_{synch}(A)$ within a logarithmic factor on all tested instances; however, no theoretical justification for these observations has been found so far.

In this paper we prove that, unless P = NP, no polynomial-time algorithm can approximate the minimum length of synchronizing words for a given synchronizing automaton within a constant factor. This result was announced in the survey [10] (with a reference to the present author's unpublished manuscript) but its proof appears here for the first time. We also mention that a special case of our result, namely, non-approximability of $\min_{synch}(A)$ within a factor of 2, was announced by Gawrychowski [4].

The paper is organized as follows. First we exhibit an auxiliary construction that shows non-approximability of $\min_{synch}(A)$ within a factor of $2-\varepsilon$ for automata with 3 input letters. Then we show how to iterate this construction in order to obtain the main result, again for automata with 3 input letters. Finally, we describe how the construction can be modified to extend the result also to automata with only 2 input letters.

1 Non-approximability within a Factor of $2-\varepsilon$

First we fix our notation and introduce some definitions. When we have specified a DFA $A = \langle Q, \Sigma, \delta\rangle$, we can simplify the notation by writing $q.w$ instead of $\delta(q,w)$ for $q \in Q$ and $w \in \Sigma^*$. For each subset $S \subseteq Q$ and each word $w \in \Sigma^*$, we write $S.w$ instead of $\{q.w \mid q \in S\}$. We say that a subset $S \subseteq Q$ is *occupied* after applying some word $v \in \Sigma^*$ if $S \subseteq Q.v$.

The length of a word $w \in \Sigma^*$ is denoted by $|w|$. If $1 \le s \le |w|$, then $w[s]$ denotes the letter in the s-th position of w; similarly, if $1 \le s < t \le |w|$, then $w[s..t]$ stands for the word $w[s]w[s+1]\cdots w[t]$.

Let $\mathcal{K}$ be a class of synchronizing automata We say that an algorithm M *approximates the minimal length of synchronizing words in* $\mathcal{K}$ if, for an arbitrary DFA $A \in \mathcal{K}$, the algorithm calculates a positive integer $M(A)$ such that $M(A) \ge \min_{synch}(A)$. The *performance ratio* of M at A is $R_M(A) = \dfrac{M(A)}{\min_{synch}(A)}$. The algorithm is said to *approximate the minimal length of synchronizing words within a factor of* $k \in \mathbb{R}$ if

$$\sup\{R_M(A) \mid A \in \mathcal{K}\} = k.$$

Even though the following theorem is subsumed by our main result, we prove it here because the proof demonstrates underlying ideas in a nutshell and in the same time presents a construction that serves as the induction basis for the proof of the main theorem.

Theorem 1. *If* P $\neq$ NP*, then for no* $\varepsilon > 0$ *a polynomial-time algorithm approximates the minimal length of synchronizing words within a factor of* $2-\varepsilon$ *in the class of all synchronizing automata with* 3 *input letters.*

Proof. Arguing by contradiction, assume that there exist a real number $\varepsilon > 0$ and a polynomial-time algorithm M such that $R_M(A) \leq 2-\varepsilon$ for every synchronizing automaton A with 3 input letters.

We fix an arbitrary $n > 2$ and take an arbitrary instance ψ of the classical NP-complete problem SAT (the satisfiability problem for a system of clauses, that is, formulae in conjunctive normal form) with n variables. Let m be the number of clauses in ψ. We shall construct a synchronizing automaton $A(\psi)$ with 3 input letters and the number of states a polynomial in m, n such that $\min_{synch}(A(\psi)) = n+2$ if ψ is satisfiable and $\min_{synch}(A(\psi)) > 2(n-1)$ if ψ is not satisfiable. If n is large enough, namely, $n \geq \frac{6}{\varepsilon} - 2$, then we can decide whether or not ψ is satisfiable by running the algorithm M on $A(\psi)$. Indeed, if ψ is not satisfiable, then $M(A(\psi)) \geq \min_{synch}(A(\psi)) > 2(n-1)$, but, if ψ is satisfiable, then

$$M(A(\psi)) \leq (2-\varepsilon)\min_{synch}(A(\psi)) = (2-\varepsilon)(n+2) \leq (2-\frac{6}{n+2})(n+2) = 2(n-1).$$

Clearly, this yields a polynomial-time algorithm for SAT: given an instance of SAT, we can first, if necessary, enlarge the number of variables to at least $\frac{6}{\varepsilon} - 2$ without influencing satisfiability and then apply the above procedure. This contradicts the assumption that P $\neq$ NP.

Now we describe the construction of the automaton $A(\psi) = \langle Q, \Sigma, \delta\rangle$. The state set Q of $A(\psi)$ is the disjoint union of the three following sets:

$$S^1 = \{q_{i,j} \mid 1 \leq i \leq m+1,\ 1 \leq j \leq n+1,\ i \neq m+1 \text{ or } j \neq n+1\},$$
$$S^2 = \{p_{i,j} \mid 1 \leq i \leq m+1,\ 1 \leq j \leq n+1\},$$
$$S^3 = \{z_1, z_0\}.$$

The size of Q is equal to $2(m+1)(n+1)+1$, and hence is a polynomial in m, n.

The input alphabet Σ of $A(\psi)$ is the set $\{a, b, c\}$. In order to describe the transition function $\delta : Q \times \Sigma \to Q$, we need an auxiliary function $f : \{a, b\} \times \{1, \ldots, m\} \times \{1, \ldots, n\} \to Q$ defined as follows. Let the variables involved in ψ be $x_1, \ldots, x_n$ and the clauses of ψ be $c_1, \ldots, c_m$. For a literal $y \in \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$ and a clause c_i, we write $y \in c_i$ to denote that y appears in c_i. Now set

$$f(d,i,j) = \begin{cases} z_0, & \text{if } d = a \text{ and } x_j \in c_i; \\ z_0, & \text{if } d = b \text{ and } \neg x_j \in c_i; \\ q_{i,j+1}, & \text{otherwise.} \end{cases}$$

The transition function δ is defined according to the following table:

State $q \in Q$	$\delta(q,a)$	$\delta(q,b)$	$\delta(q,c)$
$q_{i,j}$ for $1 \le i \le m, 1 \le j \le n$	$f(a,i,j)$	$f(b,i,j)$	$q_{i,1}$
$q_{m+1,j}$ for $1 \le j < n$	$q_{m+1,j+1}$	$q_{m+1,j+1}$	$q_{m+1,1}$
$q_{m+1,n}$	z_1	z_1	$q_{m+1,1}$
$q_{i,n+1}$ for $1 \le i \le m$	z_0	z_0	$q_{m+1,1}$
$p_{i,j}$ for $1 \le i \le m+1, 1 \le j \le n$	$p_{i,j+1}$	$p_{i,j+1}$	$p_{i,j+1}$
$p_{i,n+1}$ for $1 \le i \le m+1$	z_0	z_0	$q_{i,1}$
z_1	$q_{m+1,1}$	$q_{m+1,1}$	z_0
z_0	z_0	z_0	z_0

Let us informally comment on the essence of the above definition. Its most important feature is that, if the literal x_j (respectively $\neg x_j$) occurs in the clause c_i, then the letter a (respectively b) moves the state $q_{i,j}$ to the state z_0. This encodes the situation when one can satisfy the clause c_i by choosing the value 1 (respectively 0) for the variable x_j. Otherwise, the letter a (respectively b) increases the second index of the state. This means that one cannot make c_i be true by letting $x_j = 1$ (respectively $x_j = 0$), and the next variable has to be inspected. Of course, this encoding idea is not new, see, e.g., [2].

By the definition, z_0 is the zero state of the automaton $A(\psi)$. Since there is a path to z_0 from each state $q \in Q$, the automaton $A(\psi)$ is synchronizing.

Figure 1 shows two automata of the form $A(\psi)$ built for the SAT instances

$$\psi_1 = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\},$$
$$\psi_2 = \{x_1 \vee x_2,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}.$$

If at some state $q \in Q$ the picture has no outgoing arrow labelled $d \in \Sigma$, the arrow $q \xrightarrow{d} z_0$ is assumed (all those arrows are omitted in the picture to improve readability). The two instances differ only in the first clause: in ψ_1 it contains the variable x_3 while in ψ_2 it does not. Correspondingly, the automata $A(\psi_1)$ and $A(\psi_2)$ differ only by the outgoing arrow labelled a at the state $q_{1,3}$: in $A(\psi_1)$ it leads to z_0 (and therefore, it is not shown) while in $A(\psi_2)$ it leads to the state $q_{1,4}$ and is shown by the dashed line.

Observe that ψ_1 is satisfiable for the truth assignment $x_1 = x_2 = 0$, $x_3 = 1$ while ψ_2 is not satisfiable. It is not hard to check that the word *cbbac* synchronizes $A(\psi_1)$ and the word a^7c is one of the shortest reset words for $A(\psi_2)$.

To complete the proof, it remains to show that $\min_{synch}(A(\psi)) = n + 2$ if ψ is satisfiable and $\min_{synch}(A(\psi)) > 2(n-1)$ if ψ is not satisfiable. First consider the case when ψ is satisfiable. Then there exists a truth assignment

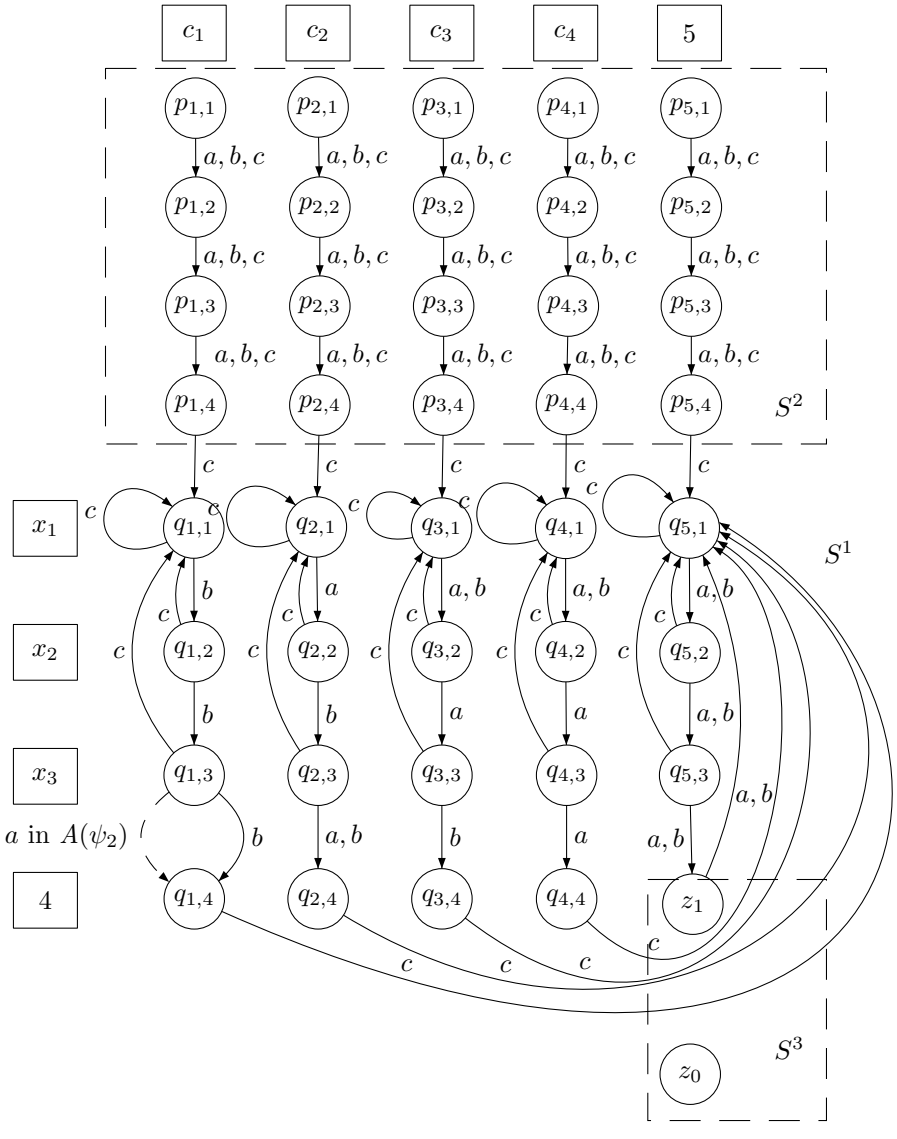


Fig. 1. The automata $A(\psi_1)$ and $A(\psi_2)$

$\tau : \{x_1, \ldots, x_n\} \to \{0, 1\}$ such that $c_i(\tau(x_1), \ldots, \tau(x_n)) = 1$ for every clause c_i of ψ. We construct a word $v = v(\tau)$ of length n as follows:

$$v[j] = \begin{cases} a, & \text{if } \tau(x_j) = 1; \\ b, & \text{if } \tau(x_j) = 0. \end{cases} \tag{1}$$

We aim to prove that the word $w = cvc$ is a synchronizing word for $A(\psi)$, that is, $Q.w = \{z_0\}$. Clearly, $z_1.c = z_0$. Further, $S^2.cv = \{z_0\}$ because every word of length $n+1$ that does not end with c sends S^2 to z_0. Now let $T = \{q_{i,1} \mid 1 \le i \le m+1\}$, so T is the "first row" of S^1. Observe that $S^1.c = T$. Since $c_i(\tau(x_1), \dots, \tau(x_n)) = 1$ for every clause c_i, there exists an index j such that either $x_j \in c_i$ and $\tau(x_j) = 1$ or $\neg x_j \in c_i$ and $\tau(x_j) = 0$. This readily implies (see the comment following the definition of the transition function of $A(\psi)$) that $q_{i,1}.v = z_0$ for all $1 \le i \le m$. On the other hand, $q_{m+1,1}.v = z_1$ because every word of length n that does not involve c sends $q_{m+1,1}$ to z_1. Thus, $S^1.cv = T.v = S^3$ and $S^1.w = \{z_0\}$. We have shown that w synchronizes $A(\psi)$, and it is clear that $|w| = n+2$ as required.

Now we consider the case when ψ is not satisfiable.

Lemma 1. *If ψ is not satisfiable, then, for each word $v \in \{a,b\}^*$ of length n, there exists $i \le m$ such that $q_{i,n+1} \in T.v$.*

Proof. Define a truth assignment $\tau : \{x_1, \dots, x_n\} \to \{0,1\}$ as follows:

$$\tau(x_j) = \begin{cases} 1, & \text{if } v[j] = a; \\ 0, & \text{if } v[j] = b. \end{cases}$$

Since ψ is not satisfiable, we have $c_i(\tau(x_1), \dots, \tau(x_n)) = 0$ for some clause c_i, $1 \le i \le m$. According to our definition of the transition function of $A(\psi)$, this means that $q_{i,j}.v[j] = q_{i,j+1}$ for all $j = 1, \dots, n$. Hence $q_{i,n+1} = q_{i,1}.v \in T.v$. □

Lemma 2. *If ψ is not satisfiable, then for each word $v \in \{a,b\}^*$ of length n and each letter $d \in \Sigma$, the state $q_{m+1,1}$ belongs to $T.vd$.*

Proof. If $d = c$, the claim follows from Lemma 1 and the equalities $q_{m+1,1} = q_{i,n+1}.c$ that hold for all $i \le m$. If $d \ne c$, we observe that the state $q_{m+1,1}$ is fixed by all words of length $n+1$ not involving c. □

Let w' be a synchronizing word of minimal length for $A(\psi)$ and denote $w = cw'c$. Then the word w is also synchronizing and $\ell = |w| > n$ because already the length of the shortest path from $q_{m+1,1}$ to z_0 is equal to $n+1$. Let k be the rightmost position of the letter c in the word $w[1..n]$.

Lemma 3. $T \subseteq Q.w[1..k]$.

Proof. Indeed, since $k \le n$, for each $1 \le i \le m+1$ we have

$$p_{i,n+2-k}.w[1..k-1]w[k] = p_{i,n+1}.c = q_{i,1} \in T. \qquad \square$$

We let v denote the longest prefix of the word $w[k+1..\ell]$ such that $v \in \{a,b\}^*$ and $|v| \le n$. Since w ends with c, the word v cannot be a suffix of w. Let $d \in \Sigma$ be the letter that follows v in w. If $|v| = n$, then Lemma 2 implies that $q_{m+1,1} \in T.vd$. If $|v| < n$, then by the definition of v we have $d = c$. Hence

$$q_{m+1,1}.vd = q_{m+1,|v|+1}.c = q_{m+1,1}.$$

Thus, $q_{m+1,1} \in T.vd$ also in this case. Combining this with Lemma 3, we have

$$Q.w[1..k]vd \supseteq T.vd \ni q_{m+1,1}. \tag{2}$$

From the definitions of k and v it readily follows that $w[k+1..n]$ is a prefix of v whence $|v| \geq n-k$. Thus, $|w[1..k]vd| \geq k+(n-k)+1 = n+1$. Recall that the length of the shortest path from $q_{m+1,1}$ to z_0 is equal to $n+1$, and the suffix of w following $w[1..k]vd$ must bring the state $q_{m+1,1}$ to z_0 in view of (2). Hence $|w| \geq (n+1)+(n+1) = 2n+2 > 2n$ and $|w'| > 2(n-1)$. We have proved that $\min_{synch}(A(\psi)) > 2(n-1)$ if ψ is not satisfiable. □

2 The Main Result

The main result of this paper is

Theorem 2. *If* P $\neq$ NP*, then no polynomial-time algorithm can approximate the minimal length of synchronizing words within a constant factor in the class of all synchronizing automata with* 3 *input letters.*

Proof. Again we fix an arbitrary $n > 2$ and take an arbitrary instance ψ of SAT with n variables. We shall prove by induction that for every $r = 2, 3, \ldots$ there exists a synchronizing automaton $A_r(\psi) = \langle Q_r, \Sigma, \delta_r \rangle$ with the following properties:

- $\Sigma = \{a, b, c\}$;
- $|Q_r|$ is bounded by a polynomial of n and the number m of clauses of ψ (for any fixed r);
- if ψ is satisfiable under a truth assignment $\tau : \{x_1, \ldots, x_n\} \to \{0, 1\}$, then the word $w = c^{r-1}v(\tau)c$ of length $n+r$ synchronizes $A_r(\psi)$ (see (1) for the definition of the word $v(\tau)$);
- $\min_{synch}(A_r) > r(n-1)$ if ψ is not satisfiable.

Then, applying the same standard argument as in the proof of Theorem 1, we conclude that for no $\varepsilon > 0$ the minimal length of synchronizing words can be approximated by a polynomial-time algorithm within a factor of $r - \varepsilon$. Since r can be arbitrarily large, the statement of the main result follows.

The induction basis is verified in the proof of Theorem 1: we can choose the synchronizing automaton $A(\psi)$ to play the role of $A_2(\psi)$. For the sake of uniformity, in the sequel we refer to the state set Q of $A(\psi)$ and its transition function δ as to Q_2 and respectively δ_2.

Suppose that $r > 2$ and the automaton $A_{r-1}(\psi) = \langle Q_{r-1}, \Sigma, \delta_{r-1} \rangle$ with the desired properties has already been constructed. We let

$$Q_r = Q_{r-1} \cup [(Q_2 - z_0) \times Q_{r-1}].$$

Clearly, $|Q_r| = |Q_{r-1}| \cdot |Q_2|$ and from the induction assumption it follows that $|Q_r|$ is a polynomial in m, n.

We now define the transition function $\delta_r : Q_r \times \Sigma \to Q_r$. Let $d \in \Sigma$, $q \in Q_r$. If $q \in Q_{r-1}$, then we set

$$\delta_r(q,d) = \delta_{r-1}(q,d). \tag{3}$$

If $q = (q', q'') \in (Q_2 \setminus \{z_0\}) \times Q_{r-1}$, we define

$$\delta_r(q,d) = \begin{cases} z_0, & \text{if } \delta_2(q',d) = z_0; \\ q'', & \text{if } \delta_2(q',d) = q_{m+1,1} \text{ and either} \\ & q' = q_{i,n+1} \text{ for } i \in \{1,\dots,m\} \\ & \text{or } q' = q_{m+1,j} \text{ for } j \in \{2,\dots,n\} \\ & \text{or } q' = z_1; \\ (\delta_2(q',d), q''), & \text{in all other cases.} \end{cases} \tag{4}$$

Using this definition and the induction assumption, one can easily verify that the state z_0 is the zero state of the automaton $A_r(\psi)$ and that there is a path to z_0 from every state in Q_r. Thus, $A_r(\psi)$ is a synchronizing automaton.

In order to improve readability, we denote the subset $\{q_{i,j}\} \times Q_{r-1}$ by $Q_{i,j}$ for each state $q_{i,j} \in S^1$ and the subset $\{p_{i,j}\} \times Q_{r-1}$ by $P_{i,j}$ for each state $p_{i,j} \in S^2$. Slightly abusing notation, we let T denote the "first row" of $S^1 \times Q_{r-1}$, i.e. $T = \bigcup_{1 \le i \le m+1} Q_{i,1}$. Similarly, let $P = \bigcup_{1 \le i \le m+1} P_{i,1}$ be the "first row" of $S^2 \times Q_{r-1}$. We also specify that the dot-notation (like $q.d$) always refers to the function δ_r.

First we aim to show that if ψ is satisfiable under a truth assignment $\tau : \{x_1,\dots,x_n\} \to \{0,1\}$, then the word $w = c^{r-1}v(\tau)c$ synchronizes the automaton $A_r(\psi)$. By (3) and the induction assumption we have $Q_{r-1}.c \subseteq Q_{r-1}$ and $Q_{r-1}.c^{r-2}v(\tau)c = z_0$. Further, we can decompose $((Q_2 \setminus \{z_0\}) \times Q_{r-1}).c$ as $\{z_0\} \cup F_{r-1} \cup F_r$ for some sets $F_{r-1} \subseteq Q_{r-1}$ and $F_r \subseteq (Q_2 \setminus \{z_0\}) \times Q_{r-1}$. By the induction assumption,

$$F_{r-1}.c^{r-2}v(\tau)c \subseteq Q_{r-1}.c^{r-2}v(\tau)c = z_0$$

Consider the set F_r. Using the definition of the action of c on Q_2 via δ_2, one can observe that $F_r = T \cup G$ where G stands for $S^2 \times Q_{r-1} \setminus P$. From (4) we see that $T.c = T$ and $G.c \subseteq T \cup G$. Thus we have $F_r.c^{r-2}v(\tau)c \subseteq T.v(\tau)c \cup G.v(\tau)c$, and combining the first alternative in (4) with properties of the automaton $A_2(\psi)$ established in the proof of Theorem 1, we obtain $T.v(\tau)c = G.v(\tau)c = \{z_0\}$.

Now we consider the case when ψ is not satisfiable. The following lemma is parallel to Lemma 1 and has the same proof because the action of a and b on the "blocks" $Q_{i,j}$ with $1 \le i \le m$ and $1 \le j \le n$ via δ_r precisely imitates the action of a and b on the states $q_{i,j}$ in the automaton $A(\psi)$, see the last alternative in (4).

Lemma 4. *If ψ is not satisfiable, then, for each word $v \in \{a,b\}^*$ of length n, there exists $i \le m$ such that $Q_{i,n+1} \subseteq \delta_r(T,v)$.* □

In contrast, the next lemma, which is a counterpart of Lemma 2 uses the fact that in some cases the action of the letters via δ_r drops states from $((Q_2 \setminus z_0) \times Q_{r-1})$ down to Q_{r-1}, see the middle alternative in (4).

Lemma 5. *If ψ is not satisfiable, then for each word $v \in \{a,b\}^*$ of length n and each letter $d \in \Sigma$, we have $Q_{r-1} \subseteq \delta_r(T, vd)$.*

Proof. If $d = c$, the claim follows from Lemma 4 and the equalities $\delta_r((q_{i,n+1}, q''), c) = q''$ that hold for all $i \leq m$ and all $q'' \in Q_{r-1}$. If $d \neq c$, we observe that $\delta_r((q_{m+1,1}, q''), v) = (z_1, q'')$ and $\delta_r((z_1, q''), a) = \delta_r((z_1, q''), b) = q''$ for all $q'' \in Q_{r-1}$. □

Let w' be a synchronizing word of minimal length for $A_r(\psi)$ and denote $w = cw'c$. Then the word w is also synchronizing and $\ell = |w| > (r-1)n$ by the induction assumption. Let k be the rightmost position of the letter c in the word $w[1..n]$. We have the next lemma parallel to Lemma 3 and having the same proof (with the "blocks" $P_{i,j}$ with $1 \leq i \leq m+1$, $n+2-k \leq j \leq n$ playing the role of the states $p_{i,j}$).

Lemma 6. $T \subseteq \delta_r(Q_r, w[1..k])$. □

Now, as in the proof of Theorem 1, we let v denote the longest prefix of the word $w[k+1..\ell]$ such that $v \in \{a,b\}^*$ and $|v| \leq n$. Clearly, v cannot be a suffix of w. Let $d \in \Sigma$ be the letter that follows v in w. If $|v| = n$ then Lemma 5 implies that $Q_{r-1} \subseteq \delta_r(T, vd)$. If $|v| < n$, then by the definition of v we have $d = c$. Hence

$$\delta_r(Q_{m+1,1}, vd) = \delta_r(Q_{m+1,|v|+1}, c) = Q_{r-1}.$$

Thus, $Q_{r-1} \subseteq \delta_r(T, vd)$ also in this case. Combining this with Lemma 6, we have

$$\delta_r(Q_r, w[1..k]vd) \supseteq \delta_r(T, vd) \supseteq Q_{r-1}. \tag{5}$$

From the definitions of k and v it readily follows that $|v| \geq n - k$. Thus, $|w[1..k]vd| \geq k + (n-k) + 1 = n + 1$. The suffix of w following $w[1..k]vd$ must bring the set Q_{r-1} to a single state in view of (5). However, by (3) the restriction of δ_r to Q_{r-1} coincides with δ_{r-1} whence the suffix must be a synchronizing word for $A_{r-1}(\psi)$. By the induction assumption $\min_{synch}(A_{r-1}(\psi)) > (r-1)(n-1)$, and therefore,

$$|w| > (n+1) + (r-1)(n-1) = r(n-1) + 2$$

and $|w'| > r(n-1)$. We have thus proved that $\min_{synch}(A_r(\psi)) > r(n-1)$ if ψ is not satisfiable. This completes the induction step. □

3 The Case of 2-Letter Alphabets

We show that the main result extends to synchronizing automata with only 2 input letters.

Corollary 1. *If* P $\neq$ NP*, then no polynomial-time algorithm can approximate the minimal length of synchronizing words within a constant factor in the class of all synchronizing automata with* 2 *input letters.*

Proof. Our aim is to show, that for any synchronizing automaton $A = (Q, \{a_1, a_2, a_3\}, \delta)$ we can construct a synchronizing automaton $B = (Q', \{a, b\}, \delta')$ such that

$$\min_{synch}(A) \leq \min_{synch}(B) \leq 3 \min_{synch}(A) \tag{6}$$

and $|Q'|$ is a polynomial of $|Q|$. Then any polynomial-time algorithm approximating the minimal length of synchronizing words for 2-letter synchronizing automata within a factor of r would give rise to a polynomial-time algorithm approximating the minimal length of synchronizing words for 3-letter synchronizing automata within a factor of $3r$. This would contradict Theorem 2.

We let $Q' = Q \times \{a_1, a_2, a_3\}$ and define the transition function $\delta' : Q' \times \{a, b\} \to Q'$ as follows:

$$\delta'((q, a_i), a) = (q, a_{\min(i+1,3)}),$$
$$\delta'((q, a_i), b) = (\delta(q, a_i), a_1).$$

Thus, the action of a on a state $q' \in Q'$ substitutes an appropriate letter from in the alphabet $\{a_1, a_2, a_3\}$ of A for the second component of q' while the action of b imitates the action of the second component of q' on its first component and resets the second component to a_1. Now let a word $w \in \{a_1, a_2, a_3\}$ of length ℓ be a synchronizing word for A. Define

$$v_s = \begin{cases} b, & \text{if } w[s] = a_1; \\ ab, & \text{if } w[s] = a_2; \\ aab, & \text{if } w[s] = a_3. \end{cases}$$

Then the word $v = bv_1 \cdots v_\ell$ is easily seen to be a synchronizing word for B and $|v| \leq 3\ell$ unless all letters in w are a_3, but in this case we can just let a_2 and a_3 swap their names. Hence the second inequality in (6) holds true, and the first inequality is clear.

Acknowledgments

The author acknowledges support from the Federal Education Agency of Russia, grant 2.1.1/3537, and from the Russian Foundation for Basic Research, grant 09-01-12142.

References

1. Černý, J.: Poznámka k homogénnym eksperimentom s konečnými automatami. Matematicko-fyzikálny Časopis Slovensk. Akad. Vied 14(3) 208–216 (1964) (in Slovak)
2. Eppstein, D.: Reset sequences for monotonic automata. SIAM J. Comput. 19, 500–510 (1990)

3. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
4. Gawrychowski, P.: Complexity of the approximation of the shortest synchronizing word. In: Workshop "Around the Černý Conjecture". Univ. Wrocław (2008) (unpublished)
5. Goralčik, P., Koubek, V.: Rank problems for composite transformations. Int. J. Algebra and Computation 5, 309–316 (1995)
6. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Reading (1994)
7. Roman, A.: Synchronizing finite automata with short reset words. Appl. Math. and Computation 209, 125–136 (2009)
8. Samotij, W.: A note on the complexity of the problem of finding shortest synchronizing words. In: Electronic Proc. AutoMathA 2007, Automata: from Mathematics to Applications. Univ. Palermo, Palermo (2007)
9. Salomaa, A.: Composition sequences for functions over a finite domain. Theoret. Comp. Sci. 292, 263–281 (2003)
10. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)

Realizability of Dynamic MSC Languages*

Benedikt Bollig[1] and Loïc Hélouët[2]

[1] LSV, ENS Cachan, CNRS, INRIA, France
[2] IRISA, INRIA, Rennes, France

Abstract. We introduce dynamic communicating automata (DCA), an extension of communicating finite-state machines that allows for dynamic creation of processes. Their behavior can be described as sets of message sequence charts (MSCs). We consider the realizability problem for DCA: given a dynamic MSC grammar (a high-level MSC specification), is there a DCA defining the same set of MSCs? We show that this problem is decidable in doubly exponential time, and identify a class of realizable grammars that can be implemented by *finite* DCA.

1 Introduction

Requirements engineering with scenario-based visual languages such as message sequence charts (MSCs) is a well established practice in industry. However, the requirements phase usually stops when a sufficiently large *finite* base of scenarios covering expected situations of the modeled system has been created. Although more elaborated formalisms have been proposed, such as HMSCs [13], compositional MSCs [8], or causal MSCs [6], requirements frequently consist in a finite set of finite behaviors over a finite set of processes. The existing higher-level constructs are often neglected. A possible reason might be that, in view of their huge expressive power, MSC specifications are not always implementable. As a part of the effort spent in the requirements design is lost when designers start implementing a system, scenarios remain confined to expressions of finite examples, and the higher-level constructs are rarely used. Another reason that may prevent designers from using high-level scenarios is that most models depict the interactions of an a priori *fixed* set of processes. Nowadays, many applications rely on threads, and most protocols are designed for an open world, where all the participating actors are not known in advance. A first step towards MSCs over an evolving set of processes was made by Leucker, Madhusudan, and Mukhopadhyay [11]. Their *fork-and-join MSC grammars* allow for dynamic creation of processes and have good properties, such as decidability of MSO model checking. However, it remains unclear how to implement fork-and-join MSC grammars. In particular, a corresponding automata model with a clear behavioral semantics based on MSCs is missing. Dynamic process creation and its realizability are then two important issues that must be considered jointly.

This paper introduces dynamic communicating automata (DCA) as a model of programs with process creation. In a DCA, there are three types of actions: (1) a new process can be created, (2) a message can be sent to an already existing process, and

* Partly supported by the projects ANR-06-SETI-003 DOTS and ARCUS Île de France-Inde.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 48–59, 2010.

(3) a message can be received from an existing process. Processes are identified by means of process variables, whose values can change dynamically during an execution of an automaton and be updated when a message is received. A message is sent through bidirectional unbounded FIFO channels, which are available to any pair of existing processes. Our model extends classical communicating finite-state machines [5], which allow only for actions of the form (2) and (3) and serve as an implementation model for existing specification languages such as HMSCs or compositional MSCs.

In a second step, we propose dynamic MSC grammars (DMG for short) as a specification language. They are inspired by the fork-and-join grammars from [11] but closer to an implementation. We keep the main idea of [11]: when unfolding a grammar, MSCs are concatenated on the basis of finitely many process identifiers. While, in [11], the location of identifiers can be changed by means of a very general and powerful split-operator, our grammars consider an identifier as a pebble, which can be moved *locally* within one single MSC. In addition to process identifiers, we introduce a new means of process identification that allows for a more concise description of some protocols.

Given an implementation model and a specification formalism, the *realizability* problem consists in asking whether a given specification comes with a corresponding implementation. Realizability for MSC languages has been extensively studied in the setting of a fixed number of processes [2,12,1]. In a dynamic framework where DMGs are seen as specifications and DCA as distributed implementations, we have to consider a new aspect of realizability, which we call *proximity realizability*. This notion requires that two processes know each other at the time of (asynchronous) communication. We show that proximity realizability can be checked in doubly exponential time. Note that the representation of the behavior of each process may require infinitely many states (due to the nature of languages generated by the grammar), and that the notion of proximity realizability does not take into account the structure of processes. The next step is then to identify a class of DMGs that is realizable in terms of *finite* DCA.

The paper is organized as follows: Section 2 introduces MSCs. Sections 3 and 4 present dynamic communicating automata and dynamic MSC grammars, respectively. In Section 5, we define proximity realizability and show that the corresponding decision problem can be solved in doubly exponential time. Moreover, we present an implementation of local-choice MSC grammars in terms of finite DCA. Section 6 concludes with some directions for future work. Missing proofs can be found in [3].

2 Message Sequence Charts

A message sequence chart (MSC) consists of a number of processes (or threads). Each process p is represented by a totally ordered set of events E_p. The total order is given by a direct successor relation $<_{\mathsf{p}}$. An event is labeled by its type. The minimal element of each thread is labeled with start. Subsequent events can then execute send (!), receive (?), or spawn (spawn) actions. The relation $<_{\mathsf{m}}$ associates each send event with a corresponding receive event on a different thread. The exchange of messages between two threads has to conform with a FIFO policy. Similarly, $<_{\mathsf{s}}$ relates a spawn event $e \in E_p$ with the (unique) start action of a different thread $q \neq p$, meaning that p has created q.

Definition 1. *An* MSC *is a tuple* $M = (\mathcal{P}, (E_p)_{p\in\mathcal{P}}, <_{\mathsf{p}}, <_{\mathsf{s}}, <_{\mathsf{m}}, \lambda)$ *where*

(a) $\mathcal{P} \subseteq \mathbb{N} = \{0, 1, \ldots\}$ *is a nonempty finite set of* processes,
(b) the E_p *are disjoint nonempty finite sets of* events *(we let* $E := \bigcup_{p \in \mathcal{P}} E_p$*),*
(c) $\lambda : E \to \{!, ?, \mathsf{spawn}, \mathsf{start}\}$ *assigns a* type *to each event, and*
(d) $<_{\mathsf{p}}$, $<_{\mathsf{s}}$, *and* $<_{\mathsf{m}}$ *are binary relations on* E.

There are further requirements: $\leq := (<_{\mathsf{p}} \cup <_{\mathsf{s}} \cup <_{\mathsf{m}})^*$ *is a partial order;* $\lambda^{-1}(\mathsf{start}) = \{e \in E \mid$ *there is no* $e' \in E$ *such that* $e' <_{\mathsf{p}} e\}$*;* $<_{\mathsf{p}} \subseteq \bigcup_{p \in \mathcal{P}} (E_p \times E_p)$ *and, for every* $p \in \mathcal{P}$*,* $<_{\mathsf{p}} \cap (E_p \times E_p)$ *is the direct-successor relation of some total order on* E_p*;* $(E, \leq)$ *has a unique minimal element, denoted by* $start(M)$*;* $<_{\mathsf{s}}$ *induces a bijection between* $\lambda^{-1}(\mathsf{spawn})$ *and* $\lambda^{-1}(\mathsf{start}) \setminus \{start(M)\}$*;* $<_{\mathsf{m}}$ *induces a bijection between* $\lambda^{-1}(!)$ *and* $\lambda^{-1}(?)$ *satisfying the following: for* $e_1, e_2 \in E_p$ *and* $e_1', e_2' \in E_q$ *with* $e_1 <_{\mathsf{m}} e_1'$ *and* $e_2 <_{\mathsf{m}} e_2'$*, we have both* $p \neq q$ *and* $e_1 \leq e_2$ *iff* $e_1' \leq e_2'$ *(FIFO).*

In Figure 1, M is an MSC with set of processes $\mathcal{P} = \{1, 2, 3, 4\}$. An MSC can be seen as one single execution of a distributed system. To generate infinite collections of MSCs, specification formalisms usually provide a concatenation operator. It will allow us to append to an MSC a partial MSC, which is a kind of suffix that does not necessarily have start events on each process. Let $M = (\mathcal{P}, (E_p)_{p \in \mathcal{P}}, <_{\mathsf{p}}, <_{\mathsf{s}}, <_{\mathsf{m}}, \lambda)$ be an MSC and let $E' \subseteq E$ be a nonempty set satisfying $E' = \{e \in E \mid (e, e') \in <_{\mathsf{m}} \cup <_{\mathsf{s}} \cup \leq^{-1}$ for some $e' \in E'\}$ (i.e., E' is an upward-closed set containing only *complete* messages and spawning pairs). Then, the restriction of M to E' is called a *partial MSC* (PMSC). In particular, the new process set is $\{p \in \mathcal{P} \mid E' \cap E_p \neq \emptyset\}$. The set of PMSCs is denoted by $\mathbb{P}$, the set of MSCs by $\mathbb{M}$. Consider Figure 1. It depicts the simple MSC I_p, with one event on process $p \in \mathbb{N}$. Moreover, $M_1, M_2 \in \mathbb{P} \setminus \mathbb{M}$.

Let $M = (\mathcal{P}, (E_p)_{p \in \mathcal{P}}, <_{\mathsf{p}}, <_{\mathsf{s}}, <_{\mathsf{m}}, \lambda)$ be a PMSC. For $e \in E$, we denote by $loc(e)$ the unique process $p \in \mathcal{P}$ such that $e \in E_p$. For every $p \in \mathcal{P}$, there are a unique minimal and a unique maximal event in $(E_p, \leq \cap (E_p \times E_p))$, which we denote by $\min_p(M)$ and $\max_p(M)$, respectively. We let $Proc(M) = \mathcal{P}$. By $Free(M)$, we denote the set of processes $p \in \mathcal{P}$ such that $\lambda^{-1}(\mathsf{start}) \cap E_p = \emptyset$. Finally, $Bound(M) = \mathcal{P} \setminus Free(M)$. Intuitively, free processes of a PMSC M are processes that are not initiated in M. In Figure 1, $Bound(\mathsf{I}_p) = \{p\}$, $Free(M_1) = \{1\}$, and $Free(M_2) = \{1, 2\}$.

Visually, concatenation of PMSCs corresponds to drawing identical processes one below the other. For $i = 1, 2$, let $M^i = (\mathcal{P}^i, (E_p^i)_{p \in \mathcal{P}^i}, <_{\mathsf{p}}^i, <_{\mathsf{s}}^i, <_{\mathsf{m}}^i, \lambda^i)$ be PMSCs. Consider the structure $M = (\mathcal{P}, (E_p)_{p \in \mathcal{P}}, <_{\mathsf{p}}, <_{\mathsf{s}}, <_{\mathsf{m}}, \lambda)$ where $E_p = E_p^1 \uplus E_p^2$ for all $p \in \mathcal{P} = \mathcal{P}^1 \cup \mathcal{P}^2$ (assuming $E_p^i = \emptyset$ if $p \notin \mathcal{P}^i$) and $<_{\mathsf{p}} = <_{\mathsf{p}}^1 \cup <_{\mathsf{p}}^2 \cup \{(\max_p(M^1), \min_p(M^2)) \mid p \in \mathcal{P}$ with $E_p^1 \neq \emptyset$ and $E_p^2 \neq \emptyset\}$. In addition, $<_{\mathsf{m}}$ and λ arise as simple unions. If M is a PMSC, then we set $M^1 \circ M^2 := M$. Otherwise, $M^1 \circ M^2$ is undefined (e.g., if some $p \in \mathcal{P}^2$ has a start event and $E_p^1 \neq \emptyset$).

In the context of partial orders, it is natural to consider linearizations. We fix the infinite alphabet $\Sigma = \{!(p, q)\,, ?(p, q)\,, \mathsf{spawn}(p, q) \mid p, q \in \mathbb{N}$ with $p \neq q\}$. For a PMSC $M = (\mathcal{P}, (E_p)_{p \in \mathcal{P}}, <_{\mathsf{p}}, <_{\mathsf{s}}, <_{\mathsf{m}}, \lambda)$, we let $poset(M) := (E', \leq', \lambda')$ where $E' = E \setminus \lambda^{-1}(\mathsf{start})$, $\leq' = \leq \cap (E' \times E')$, and $\lambda' : E' \to \Sigma$ such that, for all $(e, \hat{e}) \in <_{\mathsf{s}}$, we have $\lambda'(e) = \mathsf{spawn}(loc(e), loc(\hat{e}))$, and, for all $(e, \hat{e}) \in <_{\mathsf{m}}$, both $\lambda'(e) = !(loc(e), loc(\hat{e}))$ and $\lambda'(\hat{e}) = ?(loc(e), loc(\hat{e}))$. The set $Lin(poset(M))$ of *linearizations* of $poset(M)$ is defined as usual as a subset of Σ^*.

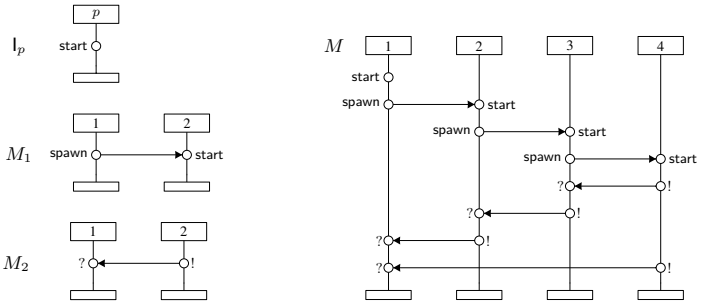


Fig. 1. (Partial) message sequence charts

3 Dynamic Communicating Automata

Dynamic communicating automata (DCA) extend classical communicating finite-state machines [5]. They allow for the dynamic creation of processes, and *asynchronous* FIFO communication between them. Note that most of existing dynamic models lack such asynchronous communication (see [4] for some references). Each process p holds a set of process variables. Their values represent process identities that p remembers at a given time, and they allow p to communicate with them. This model is close to the threading mechanism in programming languages such as JAVA and Erlang, but also borrows elements of the routing mechanisms in protocols implemented over partially connected mesh topologies. Threads will be represented by dynamically created copies of the same automaton. At creation time, the creating thread will pass known process identities to the created thread. A thread can communicate with another one if both threads know each other, i.e., they have kept their identities in memory. This mechanism is chosen to preserve the partial-order concurrency of MSCs, which provide the semantics of DCA.

We introduce DCA with an example. The DCA in Figure 2 comes with sets of process variables $X = \{x_1, x_2, x_3\}$, messages $Msg = \{m\}$, states $Q = \{s_0, \ldots, s_6\}$ where s_0 is the initial state, final states $F = \{s_3, s_4, s_6\}$, and transitions, which are labeled with actions. Each process associates with every variable in X the identity of an existing process. At the beginning, there is one process, say 1. Moreover, all process variables have value 1, i.e., $(x_1, x_2, x_3) = (1, 1, 1)$. When process 1 moves from s_0 to s_1, it executes $x_1 \leftarrow \mathsf{spawn}(s_0, (\mathsf{self}, \mathsf{self}, x_3))$, which creates a new process, say 2, starting in s_0. In the creating process, we obtain $x_1 = 2$. In process 2, on the other hand, we initially have $(x_1, x_2, x_3) = (1, 1, 1)$. So far, this scenario is captured by the first three events in the MSC M of Figure 1. Process 2 itself might now spawn a new process 3, which, in turn, can create a process 4 in which we initially have $(x_1, x_2, x_3) = (3, 3, 1)$. Now assume that, instead of spawning a new process, 4 moves to state s_5 so that it sends the message $(m, (4, 3, 1))$ to process 3. Recall that process 3 is in state s_1 with $(x_1, x_2, x_3) = (4, 2, 1)$. Thus, 3 can execute $x_1\,?\,(m, \{x_1\})$, i.e., receive $(m, (4, 3, 1))$ and set x_1 to 4. We then have $(x_1, x_2, x_3) = (4, 2, 1)$ on process 3. The DCA accepts, e.g., the behavior M depicted in Figure 1.

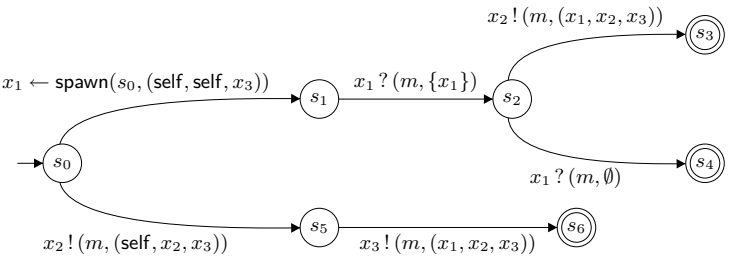


Fig. 2. A dynamic communicating automaton

Definition 2. *A* dynamic communicating automaton *(or simply DCA) is a tuple* $\mathcal{A} = (X, Msg, Q, \Delta, \iota, F)$ *where* X *is a set of* process variables*,* Msg *is a set of* messages*,* Q *is a set of* states*,* $\iota \in Q$ *is the* initial state*,* $F \subseteq Q$ *is the set of* final states*, and* $\Delta \subseteq Q \times Act_{\mathcal{A}} \times Q$ *is the set of* transitions*. Here,* $Act_{\mathcal{A}}$ *is a set of* actions *of the form* $x \leftarrow \mathsf{spawn}(s,\eta)$ *(*spawn action*),* $x\,!\,(m,\eta)$ *(*send action*),* $x\,?\,(m,Y)$ *(*receive action*), and* $\mathsf{rn}(\sigma)$ *(*variable renaming*) where* $x \in X$*,* $s \in Q$*,* $\eta : (X \uplus \{\mathsf{self}\})^X$*,* $\sigma : X \to X$*,* $Y \subseteq X$*, and* $m \in Msg$*. We say that* $\mathcal{A}$ *is* finite *if* X*,* Msg*, and* Q *are finite.*

We define the semantics of a DCA as a word language over Σ. This language is the set of linearizations of some set of MSCs and therefore yields a natural semantics in terms of MSCs. Let $\mathcal{A} = (X, Msg, Q, \Delta, \iota, F)$ be some DCA.

A *configuration* of $\mathcal{A}$ is a quadruple $(\mathcal{P}, state, proc, ch)$ where $\mathcal{P} \subseteq \mathbb{N}$ is a non-empty finite set of active processes (or identities), $state : \mathcal{P} \to Q$ maps each active process to its current state, $proc : \mathcal{P} \to \mathcal{P}^X$ contains the identities that are known to some process, and $ch : (\mathcal{P} \times \mathcal{P}) \to (Msg \times \mathcal{P}^X)^*$ keeps track of the channels contents. The configurations of $\mathcal{A}$ are collected in $Conf_{\mathcal{A}}$. We define a global transition relation $\Longrightarrow_{\mathcal{A}} \subseteq Conf_{\mathcal{A}} \times (\Sigma \cup \{\varepsilon\}) \times Conf_{\mathcal{A}}$ as follows: For $a \in \Sigma \cup \{\varepsilon\}$, $c = (\mathcal{P}, state, proc, ch) \in Conf_{\mathcal{A}}$, and $c' = (\mathcal{P}', state', proc', ch') \in Conf_{\mathcal{A}}$, we let $(c, a, c') \in \Longrightarrow_{\mathcal{A}}$ if there are $p \in \mathcal{P}$ and $\hat{p} \in \mathbb{N}$ with $p \neq \hat{p}$ (the process executing a and the communication partner or spawned process), $x \in X$, $s_0 \in Q$, $\eta : (X \uplus \{\mathsf{self}\})^X$, $Y \subseteq X$, $\sigma : X \to X$, and $(s, b, s') \in \Delta$ such that $state(p) = s$, and one of the cases in Figure 3 holds (c and c' coincide for all values that are not specified below a line).

An *initial configuration* is of the form $(\{p\}, p \mapsto \iota, proc, (p,p) \mapsto \varepsilon) \in Conf_{\mathcal{A}}$ for some $p \in \mathbb{N}$ where $proc(p)[x] = p$ for all $x \in X$. A configuration $(\mathcal{P}, state, proc, ch)$ is *final* if $state(p) \in F$ for all $p \in \mathcal{P}$, and $ch(p,q) = \varepsilon$ for all $(p,q) \in \mathcal{P} \times \mathcal{P}$. A *run* of DCA $\mathcal{A}$ on a word $w \in \Sigma^*$ is an alternating sequence $c_0, a_1, c_1, \ldots, a_n, c_n$ of configurations $c_i \in Conf_{\mathcal{A}}$ and letters $a_i \in \Sigma \cup \{\varepsilon\}$ such that $w = a_1.a_2 \ldots a_n$, c_0 is an initial configuration and, for every $i \in \{1, \ldots, n\}$, $(c_{i-1}, a_i, c_i) \in \Longrightarrow_{\mathcal{A}}$.[1] The run is *accepting* if c_n is a final configuration. The *word language* of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words $w \in \Sigma^*$ such that there is an accepting run of $\mathcal{A}$ on w. Finally, the *(MSC) language* of $\mathcal{A}$ is $L(\mathcal{A}) := \{M \in \mathbb{M} \mid Lin(poset(M)) \subseteq \mathcal{L}(\mathcal{A})\}$. Figure 2 shows a *finite* DCA. It accepts the MSCs that look like M in Figure 1.

[1] Here and elsewhere, $u.w$ denotes the concatenation of words u and v. In particular, $a.\varepsilon = a$.

$$\textsf{spawn}\ \frac{\begin{array}{c} a = \textsf{spawn}(p,\hat{p}) \qquad b = x \leftarrow \textsf{spawn}(s_0,\eta) \\ \mathcal{P}' = \mathcal{P} \uplus \{\hat{p}\} \quad ch'(q,q') = \varepsilon \quad proc'(p)[x] = \hat{p} \\ state'(p) = s' \quad \text{if } \hat{p} \in \{q,q'\} \quad proc'(\hat{p})[y] = \begin{cases} proc(p)[\eta[y]] & \text{if } \eta[y] \neq \textsf{self} \\ p & \text{if } \eta[y] = \textsf{self} \end{cases} \\ state'(\hat{p}) = s_0 \qquad \text{for all } y \in X \end{array}}{}$$

$$\textsf{send}\ \frac{a = !(p,\hat{p}) \qquad b = x\,!\,(m,\eta) \qquad \hat{p} = proc(p)[x]}{state'(p) = s' \qquad ch'(p,\hat{p}) = (m,\gamma).ch(p,\hat{p})}$$
$$\text{where } \gamma \in \mathcal{P}^X \text{ with}$$
$$\gamma[y] = \begin{cases} proc(p)[\eta[y]] & \text{if } \eta[y] \neq \textsf{self} \\ p & \text{if } \eta[y] = \textsf{self} \end{cases}$$

$$\textsf{receive}\ \frac{a = ?(\hat{p},p) \qquad b = x\,?\,(m,Y) \qquad \hat{p} = proc(p)[x]}{state'(p) = s' \qquad \text{there is } \gamma \in \mathcal{P}^X \text{ such that}}$$
$$\left[\begin{array}{l} ch(\hat{p},p) = ch'(\hat{p},p).(m,\gamma) \\ \wedge\ \text{for all } y \in Y, proc'(p)[y] = \gamma[y] \end{array}\right]$$

$$\textsf{renaming}\ \frac{a = \varepsilon \qquad b = \textsf{rn}(\sigma)}{state'(p) = s' \qquad proc'(p)[y] = proc(p)[\sigma(y)] \quad \text{for all } y \in X}$$

Fig. 3. Global transition relation of a DCA

DCA actually generalize the classical setting of communicating finite-state machines [5]. To simulate them, the starting process spawns the required number of processes and broadcasts their identities to any other process.

4 Dynamic MSC Grammars

In this section, we introduce *dynamic MSC grammars* (DMGs). They are inspired by the grammars from [11], but take into account that we want to implement them in terms of DCA. We keep the main idea of [11] and use process identifiers to tag active processes in a given context. Their concrete usage is different, though, and allows us to define protocols such as the language of the DCA from Figure 2 in a more compact way.

Let us start with an example. Figure 4 depicts a DMG with non-terminals $\mathcal{N} = \{S, A, B\}$, start symbol S, process identifiers $\Pi = \{\pi_1, \pi_2\}$, and five rules. Any rule has a left-hand side (a non-terminal), and a right-hand side (a sequence of non-terminals and PMSCs). In a derivation, the left-hand side can be replaced with the right-hand side. This replacement, however, depends on a more subtle structure of a rule. The bottom left one, for example, is actually of the form $A \longrightarrow_f \alpha$ with $\alpha = \mathcal{M}_1.A.\mathcal{M}_2$, where f is a function that maps the first process of α, which is considered *free*, to the process identifier π_2. This indicates where α has to be inserted when replacing A in a configuration. To illustrate this, consider a derivation as depicted in Figure 5, which is a sequence of configurations, each consisting of an upper and a lower part. The upper part is a *named* MSC [11], an MSC where some processes are tagged with process

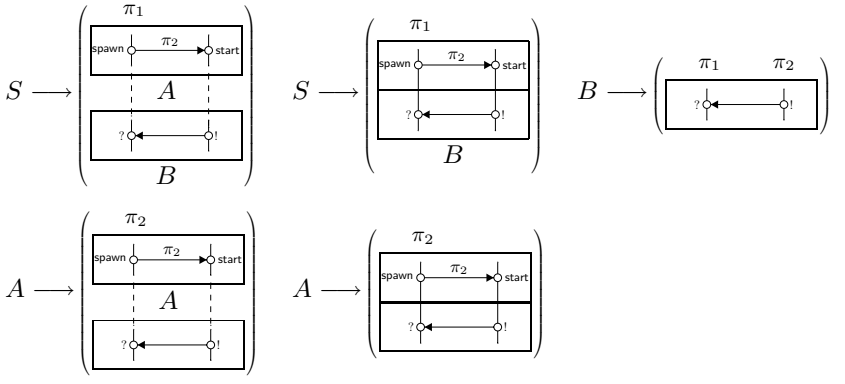


Fig. 4. A dynamic MSC grammar

identifiers. The lower part, a sequence of PMSCs and non-terminals, is subject to further evaluation. In the second configuration, which is of the form $(\mathfrak{M}, A.\beta)$ (with named MSC $\mathfrak{M}$), replacing A with α requires a renaming σ of processes: the first process of α, tagged with π_2, takes the identity of the second process of $\mathfrak{M}$, which also carries π_2. The other process of α is considered newly created and obtains a fresh identity. Thereafter, A can be replaced with $\alpha\sigma$ so that we obtain a configuration of the form $(\mathfrak{M}, \mathcal{M}.\gamma)$, $\mathcal{M}$ being a PMSC. The next configuration is $(\mathfrak{M} \circ \mathcal{M}, \gamma)$ where the concatenation $\mathfrak{M} \circ \mathcal{M}$ is simply performed on the basis of process names and does not include any further renaming. Process identifiers might migrate, though. Actually, $\mathcal{M}$ is a pair (M, μ) where M is a PMSC and μ partially maps process identifiers π to process pairs (p, q), allowing π to change its location from p to q during concatenation (cf. the third configuration in Figure 5, where π_2 has moved from the second to the third process).

Let us formalize the components of a DMG. Let Π be a nonempty and finite set of *process identifiers*. A *named MSC* over Π is a pair (M, ν) where M is an MSC and $\nu : \Pi \to \mathit{Proc}(M)$. An *in-out PMSC* over Π is a pair (M, μ) where M is a PMSC and $\mu : \Pi \rightharpoonup \mathit{Free}(M) \times \mathit{Proc}(M)$ is a partial mapping. We denote by $\mathrm{n}\mathbb{M}$ the set of named MSCs and by $\mathrm{m}\mathbb{P}$ the set of in-out PMSCs over Π (we assume that Π is clear from the context). We let $\mathfrak{M}$ range over named MSCs and $\mathcal{M}$ over in-out PMSCs.

A derivation of a DMG is a sequence of configurations $(\mathfrak{M}, \beta)$. The named MSC $\mathfrak{M}$ represents the scenario that has been executed so far, and β is a sequence of non-terminals and in-out PMSCs that will be evaluated later, proceeding from left to right. If $\beta = \mathcal{M}.\gamma$ for some in-out PMSC $\mathcal{M}$, then the next configuration is $(\mathfrak{M} \circ \mathcal{M}, \gamma)$. However, the concatenation $\mathfrak{M} \circ \mathcal{M}$ is defined only if $\mathfrak{M}$ and $\mathcal{M}$ are compatible. Formally, we define a partial operation $\_ \circ \_ : \mathrm{n}\mathbb{M} \times \mathrm{m}\mathbb{P} \rightharpoonup \mathrm{n}\mathbb{M}$ as follows: Let $(M_1, \nu_1) \in \mathrm{n}\mathbb{M}$ and $(M_2, \mu_2) \in \mathrm{m}\mathbb{P}$. Then, $(M_1, \nu_1) \circ (M_2, \mu_2)$ is defined if $M_1 \circ M_2$ is defined and contained in $\mathbb{M}$, and, for all $\pi \in \Pi$ such that $\mu_2(\pi) = (p, q)$ is defined, we have $\nu_1(\pi) = p$. If defined, we set $(M_1, \nu_1) \circ (M_2, \mu_2) := (M, \nu)$ where $M = M_1 \circ M_2$, $\nu(\pi) = \nu_1(\pi)$ if $\mu_2(\pi)$ is undefined, and $\nu(\pi) = q$ if $\mu_2(\pi) = (p, q)$ is defined.

Consider a configuration $(\mathfrak{M}, A.\gamma)$. Replacing non-terminal A with a sequence α includes a renaming of processes to make sure that those that are *free* in α and carry

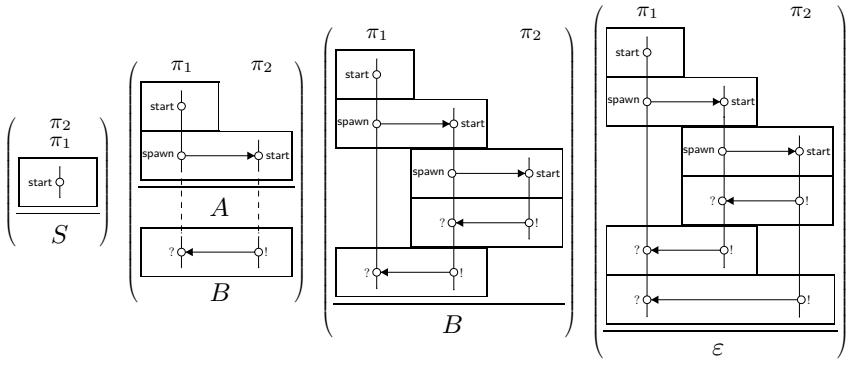


Fig. 5. A derivation

identifier π have the same name as an existing process of $\mathfrak{M}$ carrying π. I.e., processes that occur free in α take identities of processes from $\mathfrak{M}$. To be able to distinguish between free and bound processes in α, we introduce the notion of an expression. Let $\mathcal{N}$ be a set of non-terminals, and Π be a set of process identifiers. An *expression* over $\mathcal{N}$ and Π is a sequence $\alpha \in (\mathrm{m}\mathbb{P} \cup \mathcal{N})^*$ of the form $u_0.(M_1, \mu_1).u_1 \ldots (M_k, \mu_k).u_k$, $k \geq 1$ and $u_i \in \mathcal{N}^*$, such that $M(\alpha) := M_1 \circ \ldots \circ M_k \in \mathbb{P}$. We let $Proc(\alpha) := Proc(M(\alpha))$, $Free(\alpha) := Free(M(\alpha))$, and $Bound(\alpha) := Bound(M(\alpha))$.

Definition 3. *A* dynamic MSC grammar *(DMG) is a quadruple* $G = (\Pi, \mathcal{N}, S, \longrightarrow)$ *where* Π *and* $\mathcal{N}$ *and are nonempty finite sets of* process identifiers *and* non-terminals*,* $S \in \mathcal{N}$ *is the* start non-terminal*, and* $\longrightarrow$ *is a finite set of* rules*. A rule is a triple* $r = (A, \alpha, f)$ *where* $A \in \mathcal{N}$ *is a non-terminal,* α *is an expression over* $\mathcal{N}$ *and* Π *with* $Free(\alpha) \neq \emptyset$*, and* $f : Free(\alpha) \to \Pi$ *is injective. We may write* r *as* $A \longrightarrow_f \alpha$*.*

In the sequel, let $|G| := |\Pi| + \sum_{A \longrightarrow_f \alpha} (|\alpha| + |M(\alpha)|)$ be the *size* of G ($|\alpha|$ denoting the length of α as a word and $|M(\alpha)|$ the number of events of $M(\alpha)$). Moreover, we set $Proc(G) := \bigcup_{A \longrightarrow_f \alpha} Proc(\alpha)$.

A *renaming* is a bijective mapping $\sigma : \mathbb{N} \to \mathbb{N}$. For an in-out PMSC $\mathcal{M} = (M, \mu)$ with $M = (\mathcal{P}, (E_p)_{p \in \mathcal{P}}, <_\mathsf{p}, <_\mathsf{s}, <_\mathsf{m}, \lambda)$, we let $\mathcal{M}\sigma = (M\sigma, \mu\sigma)$ where $M\sigma = (\sigma(\mathcal{P}), (E_{\sigma^{-1}(p)})_{p \in \sigma(\mathcal{P})}, <_\mathsf{p}, <_\mathsf{s}, <_\mathsf{m}, \lambda)$ and $\mu\sigma(\pi) = (\sigma(p), \sigma(q))$ if $\mu(\pi) = (p, q)$ is defined; otherwise, $\mu\sigma(\pi)$ is undefined. For a rule $r = (A, \alpha, f)$ with $\alpha = u_0.\mathcal{M}_1.u_1 \ldots \mathcal{M}_k.u_k$, we set $r\sigma := (A, \alpha\sigma, f\sigma)$ where $\alpha\sigma = u_0.\mathcal{M}_1\sigma.u_1 \ldots \mathcal{M}_k\sigma.u_k$ and $f\sigma(q) = f(\sigma^{-1}(q))$ for $q \in Free(\alpha\sigma)$.

A *configuration* of DMG $G = (\Pi, \mathcal{N}, S, \longrightarrow)$ is a pair $(\mathfrak{M}, \beta)$ where $\mathfrak{M} \in \mathrm{n}\mathbb{M}$ and $\beta \in (\mathrm{m}\mathbb{P} \cup \mathcal{N})^*$. If $\beta = \varepsilon$, then the configuration is said to be *final*. Let $Conf_G$ be the set of configurations of G. A configuration is *initial* if it is of the form $((\mathsf{l}_p, \nu), S)$ for some $p \in \mathbb{N}$, where l_p is depicted in Figure 1 and $\nu(\pi) = p$ for all $\pi \in \Pi$. The semantics of G is given as the set of (named) MSCs appearing in final configurations that can be derived from an initial configuration by means of relations $\stackrel{r}{\Longrightarrow}_G \subseteq Conf_G \times Conf_G$ (for every rule r) and $\stackrel{\mathrm{e}}{\Longrightarrow}_G \subseteq Conf_G \times Conf_G$.

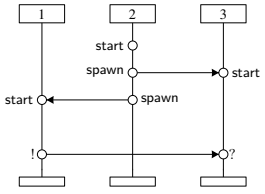


Fig. 6. Not realizable

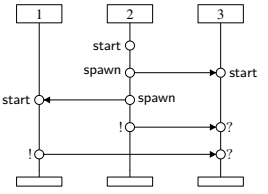


Fig. 7. 2-realizable

- For configurations $\mathcal{C} = (\mathfrak{M}, A.\gamma)$ and $\mathcal{C}' = (\mathfrak{M}, \alpha.\gamma)$, $\mathfrak{M} = (M, \nu)$, and $r \in \longrightarrow$, we let $\mathcal{C} \overset{r}{\Longrightarrow}_G \mathcal{C}'$ if there is a renaming σ such that $r\sigma = (A, \alpha, f)$, $\nu(f(p)) = p$ for all $p \in \mathit{Free}(\alpha)$, and $\mathit{Proc}(M) \cap \mathit{Bound}(\alpha) = \emptyset$.
- For configurations $\mathcal{C} = (\mathfrak{M}, \mathcal{M}.\gamma)$ and $\mathcal{C}' = (\mathfrak{M}', \gamma)$, we let $\mathcal{C} \overset{\mathrm{e}}{\Longrightarrow}_G \mathcal{C}'$ if $\mathfrak{M}' = \mathfrak{M} \circ \mathcal{M}$ (in particular, $\mathfrak{M} \circ \mathcal{M}$ must be defined).

We define $\Longrightarrow_G$ to be $\overset{\mathrm{e}}{\Longrightarrow}_G \cup \bigcup_{r \in \longrightarrow} \overset{r}{\Longrightarrow}_G$. The *language* of G is the set $L(G) := \{M \in \mathbb{M} \mid \mathcal{C}_0 \Longrightarrow_G^* ((M, \nu), \varepsilon)$ for some initial configuration $\mathcal{C}_0$ and $\nu\}$.

Let us formalize $G = (\Pi, \mathcal{N}, S, \longrightarrow)$ from Figure 4. Given the PMSCs M_1 and M_2 from Figure 1, we let $\mathcal{M}_1 = (M_1, \mu_1)$, $\mathcal{M}_2 = (M_2, \mu_2)$, and $\mathcal{M}_{12} = (M_1 \circ M_2, \mu_1)$ be in-out PMSCs with $\mu_1(\pi_1), \mu_2(\pi_1), \mu_2(\pi_2)$ undefined and $\mu_1(\pi_2) = (1, 2)$. We have

$$S \longrightarrow_{f_S} \mathcal{M}_1.A.\mathcal{M}_2.B \qquad S \longrightarrow_{f_S} \mathcal{M}_{12}.B \qquad B \longrightarrow_{f_B} \mathcal{M}_2$$
$$A \longrightarrow_{f_A} \mathcal{M}_1.A.\mathcal{M}_2 \qquad A \longrightarrow_{f_A} \mathcal{M}_{12}$$

where $f_S(1) = f_B(1) = \pi_1$ and $f_A(1) = f_B(2) = \pi_2$. Recall that $\Longrightarrow_G^*$ is illustrated in Figure 5. In a configuration, the part above a first non-terminal (if there is any) illustrates a named MSC. Note that $L(G) = L(\mathcal{A})$ for the DCA $\mathcal{A}$ from Figure 2.

5 Realizability of Dynamic MSC Grammars

Definition 4. *Let $L \subseteq \mathbb{M}$ be an MSC language. We call L* (proximity) realizable *if there is a DCA $\mathcal{A}$ such that $L = L(\mathcal{A})$. For $B \in \mathbb{N}$, we say that L is B-*realizable *if there is a DCA $\mathcal{A} = (X, \mathit{Msg}, Q, \Delta, \iota, F)$ such that $L = L(\mathcal{A})$ and $|X| \leq B$.*

The MSC M from Figure 1, considered as a singleton set, is 3-realizable. It is not 2-realizable. The singleton set from Figure 6 is not realizable, as process 3 receives a message from an unknown process. Adding a message makes it 2-realizable (Figure 7).

Theorem 1. *For a DMG G, one can decide in exponential time (wrt. $|G|$) if $L(G)$ is empty, and in doubly exponential time if $L(G)$ is realizable.*

Proof (sketch). Let $G = (\Pi, \mathcal{N}, S, \longrightarrow)$ be a DMG. To answer the first question, we build a tree automaton $\mathcal{A}_G$ that accepts all parse trees that correspond to successful derivations of G. Thus, we have $L(\mathcal{A}_G) = \emptyset$ iff $L(G) = \emptyset$. To answer the second question, we build a tree automaton $\mathcal{B}_G$ for those parse trees that give rise to realizable MSCs (considering an MSC as a singleton set). One can show that $L(G)$ is realizable iff all MSCs in $L(G)$ are realizable. Thus, $L(G)$ is realizable iff $L(\mathcal{A}_G) \setminus L(\mathcal{B}_G) = \emptyset$.

We restrict here to the more involved construction of the tree automaton $\mathcal{B}_G$. To illustrate the idea of $\mathcal{B}_G$, we use the DMG G from Figure 4. The left-hand side of Figure 8 depicts the parse tree t of G that corresponds to the derivation from Figure 5. We, therefore, call t legal. Note that, for technical reasons, the function f from a rule $A \longrightarrow_f \alpha$ is located at its non-terminal A. The crucial point of the construction is to record, during a derivation, only a bounded amount of information on the current communication structure of the system. A communication structure is a partition of the set of process identifiers together with a binary relation that provides information on what processes know of other processes. The right-hand side of Figure 8 depicts a run of $\mathcal{B}_G$ on t. States, which are assigned to nodes, are framed by a rectangle. A state is hence either a pair of communication structures (together with a non-terminal, which is omitted), or an element from $\mathrm{m}\mathbb{P}$ that occurs in G. Our automaton works bottom-up. Consider the upper right leaf of the run tree, which is labeled with its state $\mathcal{M}_{12}$. Suppose that, when it comes to executing $\mathcal{M}_{12}$, the current communication structure C_0 of the system contains two processes carrying π_1 and π_2, respectively, that know each other (represented by the two edges). When we apply $\mathcal{M}_{12}$, the outcome will be a new structure, C_1, with a newly created process that collects process identifier π_2. Henceforth, the process carrying π_1 is known to that carrying π_2, but the converse does not hold. Names of nodes are omitted; instead, identical nodes are combined by a dotted line. We conclude that applying $A \longrightarrow_{f_A} \mathcal{M}_{12}$ has the effect of transforming C_0 into C_1. Therefore, (C_0, A, C_1) is a state that can be assigned to the (A, f_A)-labeled node, as actually done in our example run. It is important here that the first structure C_0 of a state (C_0, A, C_1) is *reduced* meaning that it restricts to nodes carrying process identifiers. The structure C_1, however, might keep some unlabeled nodes, but only those that stem from previously labeled ones. Hence, the set of states of $\mathcal{B}_G$ will be finite, though exponential in $|G|$. Like elements of $\mathrm{m}\mathbb{P}$, a triple (C_0, A, C_1) can be applied to a communication structure. E.g., the states that label the successors of the root transform D_0 into D_1. It is crucial here that, at any time, communicating processes know each other in the current communication structure. Now, we can reduce D_1 to D_2 by removing the middle node, as it does not carry a process identifier nor does it arise from an identifier-carrying node. Thus, (D_0, S, D_2) is the state assigned to the root. It is final, as D_0 consists of only one process, which carries all the process identifiers. A final state at the root ensures that the run tree represents a derivation that starts in the initial configuration gathering all process identifiers, and ends in a realizable MSC. □

Corollary 1. *For every DMG* $G = (\Pi, \mathcal{N}, S, \longrightarrow)$, $L(G)$ *is realizable iff* $L(G)$ *is* $(|Proc(G)| + a \cdot |\Pi|)$*-realizable where* $a = \max\{|\alpha| \mid A \longrightarrow_f \alpha\}$.

A realizable DMG is not necessarily implementable as a finite DCA, as the behavior of a single process need not be finite-state. We will determine a simple (but non-trivial) class of DMGs that are finitely realizable. To guarantee finiteness, we restrict to right-linear rules: a rule $A \longrightarrow_f \alpha$ is *right-linear* if α is of the form $\mathcal{M}$ or $\mathcal{M}.B$. Our class is inspired by *local-choice HMSCs* as introduced in [9]. Local-choice HMSCs are scenario descriptions over a fixed number of processes in which every choice of the specification is taken by a root process for that choice. This root is in charge of executing the minimal event of every scenario, and the subsequent messages can then be tagged to inform other

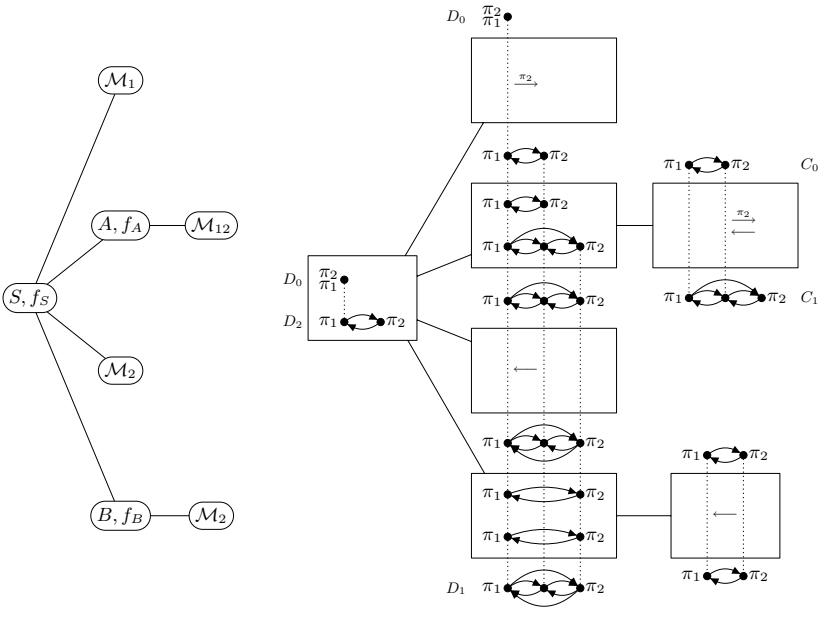


Fig. 8. A legal parse tree of G and a run of $\mathcal{B}_G$

processes about the choice. Note that locality allows for a deadlock-free implementation if the number of processes is fixed [7]. This is not guaranteed in our setting.

To adapt the notion of local-choice to DMGs, we essentially replace "process" in HMSCs by "process identifier". I.e., the root process that chooses the next rule to be applied must come with a process identifier π that is *active* in the current rule. So, for a right-linear rule $r = A \longrightarrow_f (M, \mu).\alpha$, we set $Active(r) = f(Free(M)) \cup \mathrm{dom}(\mu)$.

Definition 5. *A DMG $(\Pi, \mathcal{N}, S, \longrightarrow)$ is* local *if, for every rule $r = A \longrightarrow_f \alpha$, r is right-linear and $M(\alpha)$ has a unique minimal element. Moreover, if $\alpha = \mathcal{M}.B$, then there is $\pi \in Active(r)$ such that, for all B-rules $B \longrightarrow_g \beta$, $M(\beta)$ has a unique minimal element e satisfying $g(loc(e)) = \pi$.*

Theorem 2. *Let G be a local DMG such that $L(G)$ is realizable. There is a finite DCA $\mathcal{A} = (X, Msg, Q, \Delta, \iota, F)$ such that $L(\mathcal{A}) = L(G)$. Hereby, $|X|$ and $|Msg|$ are polynomial in $|G|$. Moreover, $|Q|$ and $|Act_{\mathcal{A}}|$ are exponential in $|G|$.*

Proof (sketch). A state of $\mathcal{A}$ will locally keep track of the progress that has been made to implement a rule. The root process may choose the next rule and inform its communication partners about this choice. The main difficulty in the implementation is the correct identification of process identities in terms of process variables. We introduce a variable x_π for each $\pi \in \Pi$ and a variable x_p for each $p \in Proc(G)$. As G is right-linear, $L(G)$ is indeed $|\Pi| + |Proc(G)|$-realizable. We pursue the following strategy of transmitting process identities: When a process p is about to instantiate a non-terminal with a new rule r, an arbitrary renaming σ is applied. We assume hereby, that the "free processes"

of r are known to p, though it is not clear to which variables they belong. Thus, σ is a simple guess, which has to be verified in the following. Indeed, the subsequent execution can pass through only if that guess is correct. The reason is that identifiers of free processes are held in local states and are sent in messages (in terms of events) so that the receiving process can be sure to receive from the correct process. Yet, we need to make sure that bound processes q are correctly identified. The idea is to enforce an update of x_q whenever a message is received from a process that "knows" q. □

6 Future Work

A nice theory of regular sets of MSCs over a fixed number of processes has been established [10] (a set of MSCs is regular if its linearization language is regular). We would like to extend this notion to our setting. Preferably, any regular set of MSCs should have an implementation in terms of a DCA. Note that, however, the linearizations of a set of (dynamic) MSCs are words over an infinite alphabet. Another challenge is to extend the class of DMGs that can be implemented by finite DCA beyond that of right-linear specifications (and preferably without deadlock). Last, we think that logics (e.g., MSO logic) may serve as an alternative specification language for DCA implementations.

References

1. Adsul, B., Mukund, M., Narayan Kumar, K., Narayanan, V.: Causal closure for MSC languages. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 335–347. Springer, Heidelberg (2005)
2. Alur, R., Etessami, K., Yannakakis, M.: Realizability and verification of MSC graphs. Theoretical Computer Science 331(1), 97–114 (2005)
3. Bollig, B., Hélouët, L.: Realizability of dynamic MSC languages. Research report, LSV, ENS Cachan (2010), http://www.lsv.ens-cachan.fr/Publis/
4. Bozzelli, L., La Torre, S., Peron, A.: Verification of well-formed communicating recursive state machines. Theoretical Computer Science 403(2-3), 382–405 (2008)
5. Brand, D., Zafiropulo, P.: On communicating finite-state machines. JACM 30(2) (1983)
6. Gazagnaire, T., Genest, B., Hélouët, L., Thiagarajan, P.S., Yang, S.: Causal message sequence charts. Theor. Comput. Sci. 410(41), 4094–4110 (2009)
7. Genest, B., Muscholl, A., Seidl, H., Zeitoun, M.: Infinite-state high-level MSCs: Model-checking and realizability. Journal on Comp. and System Sciences 72(4), 617–647 (2006)
8. Gunter, E.L., Muscholl, A., Peled, D.: Compositional message sequence charts. STTT 5(1), 78–89 (2003)
9. Hélouët, L., Jard, C.: Conditions for synthesis of communicating automata from HMSCs. In: FMICS'00, pp. 203–224. Springer, Heidelberg (2000)
10. Henriksen, J.G., Mukund, M., Narayan Kumar, K., Sohoni, M., Thiagarajan, P.S.: A theory of regular MSC languages. Information and Computation 202(1), 1–38 (2005)
11. Leucker, M., Madhusudan, P., Mukhopadhyay, S.: Dynamic message sequence charts. In: Agrawal, M., Seth, A.K. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 253–264. Springer, Heidelberg (2002)
12. Lohrey, M.: Realizability of high-level message sequence charts: closing the gaps. Theoretical Computer Science 309(1-3), 529–554 (2003)
13. Rudolph, E., Graubmann, P., Grabowski, J.: Tutorial on message sequence charts. Computer Networks and ISDN Systems 28(12), 1629–1641 (1996)

The MAX QUASI-INDEPENDENT SET Problem*

N. Bourgeois[1], A. Giannakos[1], G. Lucarelli[1,2], I. Milis[2], V. Th. Paschos[1], and O. Pottié[1]

[1] LAMSADE, CNRS and Université Paris-Dauphine, France
{bourgeois,giannako,lucarelli,paschos,pottie}@lamsade.dauphine.fr
[2] Department of Informatics, Athens University of Economics and Business, Greece
{gluc,milis}@aueb.gr

Abstract. In this paper, we deal with the problem of finding quasi-independent sets in graphs. This problem is formally defined in three versions, which are shown to be polynomially equivalent. The one that looks most general, namely, f-QIS, consists of, given a graph and a non-decreasing function f, finding a maximum size subset Q of the vertices of the graph, such that the number of edges in the induced subgraph is less than or equal to $f(|Q|)$. For this problem, we show an exact solution method that runs within time $O^*(2^{\frac{d-27/23}{d+1}n})$ on graphs of average degree bounded by d. For the most specifically defined γ-QIS and k-QIS problems, several results on complexity and approximation are shown, and greedy algorithms are proposed, analyzed and tested.

1 Introduction and Preliminaries

The problem of finding in a graph a maximum size subgraph whose density differs (being smaller or larger) from that of the whole graph, arises often in various application contexts. For example, inputs may represent graphs, wherein dense (with respect to the input) subgraphs are sought, as it is the case for call details database mining [1], or protein-protein interaction networks analysis [8]. In other cases, inputs may represent graphs from which one wants to extract maximum size, sparser than the input graphs, as for example in visualization tools for stock market interaction graphs [3].

In this paper we address the problem of finding in a graph a maximum size subgraph whose sparsity is less than or equal to a value specified with the input. In the case that appears as most general, the sparsity of a graph is measured by means of a function bounding the number of edges in the sought subgraph and depending on its size; we also study some special forms of this function, namely, when it has the form of the ratio of the number of edges of the solution to the number of edges in a complete graph of equal size, and also when it is a numeric parameter of the input.

We denote by G a simple finite graph without loops, by V and $E(G)$ its vertex set and its edge set, respectively, and by n and m their respective sizes.

* This work is partially supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 60–71, 2010.

Let A, B be two subsets of V. The induced subgraph by A in G is denoted by $G[A]$ and its edge set by $E[A]$, respectively. The edge set with one extremity in A and the other in $B \setminus A$ will be denoted as $E[A, B]$. Clearly, if A and B are disjoint then $E[A, B] = E[B, A]$ and $E[A, A] = \emptyset$. The *degree of A towards B* is equal to $|E[A, B]|$ and is denoted by $\delta_B(A)$; when A is reduced to a singleton $\{v\}$, we denote its degree in B by $\delta_B(v)$, or simply by $\delta(v)$ whenever $B = V$. The maximum vertex degree in a graph G is denoted by Δ_G or simply by Δ if there is no risk of confusion. We also set $d_B(A) = 1/|A| \sum_{v \in A} \delta_B(v)$, and $d = 1/n \sum_{v \in V} \delta(v)$.

We tackle the following variants of the quasi-independent set problem.

MAXIMUM f-QUASI-INDEPENDENT SET (f-QIS, the general quasi-independent set problem):
Given a graph G and a polynomially computable non-decreasing real function $f : \mathbb{N} \to \mathbb{R}$, find the largest possible $Q \subseteq V$ such that $|E[Q]| \leq f(|Q|)$.

In the above definition, f is used as a *sparsity specification* for the induced subgraph of the sought solution. We study in particular two variants of f-QIS, denoted by γ-QIS and k-QIS, respectively, formally defined in what follows.

In the first one, sparsity specification is given in the special form of the ratio of the number of edges in the subgraph induced by the quasi-independent set over the number of edges induced by a complete graph of the same size:

MAXIMUM γ-QUASI-INDEPENDENT SET (γ-QIS):
Given a graph G and a real γ, $0 \leq \gamma \leq 1$, find the largest possible $Q \subseteq V$ such that $|E[Q]| \leq \gamma \binom{|Q|}{2}$.

It is easy to see that γ-QIS is not hereditary (a problem is said hereditary if its solutions satisfy some non-trivial hereditary property). Indeed, given a feasible solution for γ-QIS, the induced subgraph $G[Q']$ of a subset Q' of Q may violate the sparsity condition $|E(Q')| \leqslant \gamma |Q'|(|Q'| - 1)/2$.

In the second restricted variant of the problem considered in the paper, we simply seek for a maximum vertex subset of the graph with no more than a constant number of edges having both extremities in it:

MAXIMUM k-QUASI-INDEPENDENT SET (k-QIS):
Given a graph G and a positive integer k, find the largest possible $Q \subseteq V$ such that $|E[Q]| \leq k$.

Clearly, k-QIS is hereditary. In fact, it is easy to see that k-QIS belongs to the family of node-deletion problems, first defined in [10] and further studied in [11]. Formally, a node-deletion problem consists of finding, given a graph G and a non-trivial hereditary property P, the minimum number of vertices of G that one has to delete from G, in order to have P satisfied in the remaining graph. In [11], it is proved that the decision version of such problems is **NP**-complete even for planar graphs.

For $f \equiv 0$ (resp., $\gamma = 0$ and $k = 0$), Q is simply a maximum independent set in G, while for $f : f(|Q|) \geq m$ (resp., for $\gamma \geq 2m/n(n-1)$ and $k \geq m$), $Q = V$

is a trivial solution; being a direct generalization of MAX INDEPENDENT SET, the f-, γ- and k-QIS problems are obviously inapproximable within better than $O(n^{1-\epsilon})$, unless $\mathbf{P} = \mathbf{NP}$ [13].

In [1] essentially the same problem as γ-QIS is addressed, formulated as the research, given a graph and $0 \leq \gamma \leq 1$, of a maximum subgraph of sparsity (defined as the ratio of the number of its edges over the number of edges of the complete graph of the same size) at least γ; any solution to this problem can be obtained as the complementary of a quasi-independent set of sparsity at most $1 - \gamma$ in the complementary of the input graph. The authors present an algorithm equivalent to the greedy algorithm for γ-QIS, analyzed later in this paper; however, they focus in implementation issues on very large instances and they don't attempt to analyze its performance. To our knowledge, the k-QIS problem has been specifically formulated for the first time in [9]. The authors call it the *k-edge-in subgraph* problem. Nevertheless, in this paper the problem is not addressed, but simply mentioned as related to other subgraph problems on which it focuses.

A kind of dual of the maximum quasi-independent set problem is to search, given a graph and a positive integer k, for the sparsest - or densest - (maximal) subgraph with exactly k vertices. This kind of problems have been extensively studied during the last years under the names of "k-sparsest" or "k-densest subgraph" problem (see, for example, [2,6,9]).

The remainder of the paper is organized as follows. Section 2 gives several bounds to the optimal solutions for γ-QIS. Section 3 tackles a specific polynomial case for the three variants of MAX QUASI-INDEPENDENT SET and proves its **NP**-hardness in bipartite graphs. In Section 4 an exact solution method with non-trivial worst-case running time for the general f-QIS problem is presented and analyzed. As we discuss here this method applies also to other combinatorial optimization problems. Finally, in Section 5 approximation results are proved for both γ-QIS (Subsection 5.1) and k-QIS (Subsection 5.2).

In what follows, when we indifferently refer to either one of the quasi-independent set versions defined above, we use the term MAX QUASI INPENDENT SET instead. Also, when no confusion arises, we sometimes use f-, or γ-, or k-QIS in a twofold way: either to denote the problem itself, or to denote a feasible solution of the corresponding problem. Also, due to limits to paper's length some of the results are given without proofs. They can be found in [4].

2 Solution Properties and Bounds

As it is already mentioned, in general, γ-QIS is not a hereditary problem; just consider an instance where the graph is an edge plus some isolated vertices, and γ is the smallest possible for having the whole graph as a trivial solution. Obviously, the sparsity condition will be violated for any strict part of the solution containing the edge. However, γ-QIS is still a weakly hereditary problem, in the sense given by the following lemma that will be used later.

Lemma 1. *Let Q be a γ-QIS in G, of size q. Then, for any $k \leq q$, there exists in G some γ-QIS $R(k) \subseteq Q$, of size k.*

Next lemma gives some bounds for the solutions of the γ-QIS.

Lemma 2. *Let Q be any non-trivial γ-quasi-independent set ($0 < \gamma < m/\binom{n}{2}$), with size q, not contained in any γ-quasi-independent set of size $q+1$. Let $\vartheta(Q) = \min_{v \in V \setminus Q}\{\delta_Q(v)\}$, let $Q^* \neq Q$ be an optimal solution for γ-QIS, q^* be its size, and for any vertex-subset P, let $d(P)$ be the average degree of the subgraph induced by P (recall that we denote $d(V)$ by d). Finally, let $\alpha_{\min}$ be the size of a smallest maximal independent set (minimum independent dominating set) in G. Then: (1) $q^* \leq \alpha_{\min}\Delta$; (2) $\frac{q^*}{q} \leq \frac{\Delta}{\vartheta(Q)}$; (3) $q \geq n - \frac{\Delta}{\gamma}$; (4) $q^* \leq \frac{\Delta}{\gamma}$; (5) $q^* \leq \sqrt{\frac{dn}{\gamma}}$; (6) $q^* \leq \frac{d(Q^*)+2}{\gamma} - 1$.*

3 Complexity Results for MAX QUASI-INDEPENDENT SET in Various Graph-Classes

The following proposition claims that all three variants of MAX QUASI INPENDENT SET dealt in this paper are closely interrelated.

Proposition 1. *f-QIS, γ-QIS and k-QIS are polynomially equivalent with respect to their exact solution.*

We now tackle MAX QUASI-INDEPENDENT SET in bipartite graphs. The following result characterizes its complexity.

Theorem 1. MAX QUASI-INDEPENDENT SET *is **NP**-hard on bipartite graphs.*

There are several hereditary graph classes the definitions of which imply direct conditions on their sparsity, independently of the measure used; take for instance complete graphs, split graphs or trees. Such proprieties, together with heredity, can be exploited in order to polynomially solve the k-QIS (in fact, by Proposition 1, any of the three variants of MAX QUASI-INDEPENDENT SET). In the sequel, we present a polynomial algorithm for k-QIS, that works on split graphs.

Let $S = (I, C, E)$ be a split graph, where I is an independent set, C is a clique, and E is the set of edges between I and C plus the edges of the clique C. The following lemma holds.

Lemma 3. *There is an optimal k-QIS on a split graph $S = (I, C, E)$ such that it contains the independent set I.*

Based upon Lemma 3, the following theorem holds:

Theorem 2. MAX QUASI-INDEPENDENT SET *is polynomial on split graphs.*

The optimal solution Q^* to k-QIS on a split graph $S = (I, C, E)$ can be found in polynomial time. Indeed, by Lemma 3, Q^* can be initialized to I. Next, we

consider the vertices of the clique C in increasing order with respect to their degree, that is in increasing order with respect to the number of edges between any vertex $c \in C$ to its neighbors in I. Using this order, we add vertices to Q^* until the number of edges of $S[Q^*]$ becomes greater than k. The proof for this greedy selection is straightforward.

4 Exact Solution of MAX QUASI-INDEPENDENT SET

In this section we give an exact algorithm for f-QIS with non-trivial worst-case running time. Let us note that to our knowledge, no algorithm that optimally solves MAX QUASI-INDEPENDENT SET with running time better than $O^*(2^n)$ is known. Also, as we will see in the sequel, the scope of the results of this section is even larger than the MAX QUASI-INDEPENDENT SET case. Indeed, the method described in what follows concerns a broad class of optimization problems, those that "match vertex branching", defined in Definition 1, below.

4.1 Problems That Match Vertex Branching

The intuition behind the exact solution method for MAX QUASI-INDEPENDENT SET, lies in the possibility of organizing the solution space of the problem in a tree-like manner. So we need first to formally characterize the class of optimization graph-problems for which such an organization is possible. This is done in Definition 1.

Definition 1. *We say that a graph problem Π matches vertex branching, if for any graph instance $G(V, E)$, for any $v \in V$, there exist some sets of parameters $\mathcal{S}_i$, some subsets $v \in H_i \subset V$ and two functions f_1, f_2 bounded above by some polynomial of n, such that*

$$opt_{\Pi}(G, \mathcal{S}_3) \leq \max\{f_1(opt_{\Pi}(G[V \setminus H_1], \mathcal{S}_1)), f_2(opt_{\Pi}(G[V \setminus H_2], \mathcal{S}_2))\}$$

where $opt_{\Pi}(G, \mathcal{S})$ denotes the value of the optimal solution of Π for G with parameter set $\mathcal{S}$.

Notice that, with appropriate choice for f_1, f_2, it is possible to replace max by min, or to make a single reduction.

Several problems whose aim is to find a specific subset in a given graph may be generalized as a problem that matches vertex branching. For example, for the MAXIMUM WEIGHTED INDEPENDENT SET: Given a graph $G(V, E)$ and a weight function $w : V \to \mathbb{R}$, we search for an independent set S maximizing $\sum_{v \in S} w(v)$, we have $\text{opt}(G, w) \leq \max\{\text{opt}(G[V \setminus v], w), \text{opt}(G[V \setminus N[v]], w) + w(v)\}$. Obviously, this remains true for the non-weighted version, i.e., whenever $w = 1$.

Also the f-QIS can be reformulated as a problem that matches vertex branching, in the following manner: Given a graph $G(V, E)$, two constants w_0, q_0 and a weight function $w : V \to \mathbb{R}$, we search for a maximal size vertex subset $Q \subseteq V$ whose induced graph $G[Q] = (Q, R)$ verifies $|R| + w_0 + \sum_{v \in Q} w(v) \leq f(|Q| + q_0)$.

Let $w_+ \equiv w + 1$ on $N(v) = \{u : \{u, v\} \in E\}$ and $w_+ \equiv w$ elsewhere. Then, it is $\text{opt}(G, w_0, q_0, w) \leq \max\{\text{opt}(G[V \setminus v], w_0, q_0, w), 1 + \text{opt}(G[V \setminus v], w_0 + w(v), q_0 + 1, w_+)\}$, and the formulation is completed by setting initially $q_0 = w_0 = 0, w \equiv 0$.

Informally, w_0 and q_0 stand, respectively, for the number of edges and of vertices that are already in the solution, while $w(v)$ represents the number of edges that will be added, if one decides to keep v.

Notice that, as it can be shown by straightforward recurrence, any problem that matches vertex branching can be solved within time $O(2^n \times poly(n, k))$, where $|\mathcal{S}_i| < k$. Obviously, this is useless for problems where a specific subgraph is sought, since they can be solved in $O^*(2^n)$.

However, an exact algorithm for f-QIS based upon vertex branching would be interesting if its running time $T(n)$ could be shown to be in $2^{\phi(\Delta)n}$ with ϕ some increasing function bounded above by 1 for any Δ. Intuitively, a possibility for such an improvement lies in finding an efficient vertex branching rule for fast reduction of the remaining graph's degree, and showing fast (polynomial) algorithms for computing a maximum f-QIS problem in bounded degree graphs.

4.2 Bottom-Up Algorithms

We give below a general scheme, using vertex branching for finding a maximum f-QIS in a graph G. Recall that by Proposition 1 such a method can be used for computing an optimal solution for any of the three variants of the MAX QUASI-INDEPENDENT SET problem, with a polynomial overhead. This scheme, parameterized by a graph G, some integer function f, two integers q_0 and w_0, and some vertex weight function w, can be written as follows:

```
procedure exactrec(G(V,E), f, q0, w0, w)
in, not_in: integer;
    if (V = ∅) then
      if (w0 ≤ f(q0)) then
        return q0
      else
        return −∞
      endif;
    endif;
    pick v ∈ V(G) such that δV(v) = max;
    not_in ← exactrec(G[V \ v], f, q0, w0, w);
    for all u neighbors of v
        w(u) ← w(u) + 1
    endfor;
    in ← exactrec(G[V \ v], f, q0 + 1, w0 + w(v), w);
    return max{ in, not_in};
end exactrec;
procedure f_QIS(G(V,E): graph, f: integer function)
    return exactrec(G, f, 0, 0, 0);
end f_QIS
```

As noted at the end of the previous subsection, the running time for an exact method based upon the above scheme can be improved if the f-QIS problem is shown polynomial on graphs of bounded small degree graphs.

Lemma 4. *Assume that some problem that matches vertex branching can be computed on graphs whose average degree is at most $d-1$, $d \in \mathbb{N}$, within time $O^*(2^{\alpha_d n})$ for a given $\alpha_d \geq 1/2$. Then, it can be computed on graphs whose average degree is at least $d-1$ within time $O^*(2^{\alpha_d n+\beta_d(m-(d-1)n/2)})$, where $\beta_d = \frac{2(1-\alpha_d)}{d+1}$.*

As a straightforward consequence of the above lemma, the following proposition holds:

Proposition 2. *Assume that some problem that matches vertex branching can be computed on graphs with average degree at most $d-1$, $d \in \mathbb{N}$, in time $O^*(2^{\alpha_d n})$ for a given $\alpha_d \geq 0.5$. Then, it may be computed on graphs whose average degree is at most d within time $O^*(2^{\alpha_{d+1} n})$, where $\alpha_{d+1} = \frac{d\alpha_d+1}{d+1}$.*

Direct consequence of Proposition 2, is the following theorem:

Theorem 3. *Any problem that is polynomial on totally disconnected graphs and matches vertex branching can be solved on graphs of average degree at most d with running time $O^*(2^{dn/(d+1)})$.*

Notice that this bound is tight for some problems that fit Definition 1. If the problem has the worst possible recurrence, then:

$$\mathrm{opt}(G, w) = \max\{f_1(\mathrm{opt}(G \setminus \{v\}, w_1)), f_2(\mathrm{opt}(G \setminus \{v\}, w_2))\}$$

For instance, this is the case of maximum quasi-independent set. If we either make a greedy choice for the branching, i.e., if we always branch on a vertex of maximal degree, or we select an independent set of maximal size, then there exists an instance where the running time is at least $2^{\frac{d}{d+1}n}$. To see this, consider for any $\delta \leq d$ the graph G_δ that is composed of $n/(d+1)$ cliques of size $\delta+1$. $T(G_1) = 2^{n/(d+1)}$. The algorithm removes one vertex in each connected component; we so have $T(G_\delta) = 2^{n/(d+1)}T(G_{\delta-1})$ and finally $T(G_d) = 2^{dn/(d+1)}$. On the other hand, $\alpha(G_d) = n/(d+1)$ (one vertex per clique).

Unfortunately, there is little hope for generalizing this corollary, since, unless $\mathbf{P} = \mathbf{NP}$ no problem is polynomial on graphs of average degree bounded above by some $d > 0$, unless it belongs to $\mathbf{P}$ (just add some independent set to decrease d). Furthermore, restricting the instance set to graphs without isolated vertices, or even to connected graphs, does not help much, since the greedy branching may disconnect the graph as well. On the other hand, some improved results can be obtained for graphs of bounded *maximum* degree.

Many problems that match vertex branching are in fact well-known to be polynomial on graphs of maximum degree 2, for instance MAX INDEPENDENT SET (or equivalently MAX CLIQUE and MIN VERTEX COVER). For some difficult problems like MAX QUASI-INDEPENDENT SET this remains true, but it is not straightforward. The corresponding result is stated in Subsection 4.3 (Proposition 8).

Proposition 3. *Any problem that is polynomial on graphs of maximum degree* 2 *and matches vertex branching can be solved on graphs of average degree* d *with running time* $O^*(2^{dn/6})$. *This bound is tight for* $d = 3$ *and a greedy choice of the branching.*

Proposition 4. *Any problem that is polynomial on graphs of maximum degree* 2 *and matches vertex branching can be solved on graphs of average degree that rounds up to* $d \leq 2$ *with running time* $O^*(2^{\frac{d-1}{d+1}n})$. *If the average degree is* d, *this bound is tight for a greedy choice of the branching.*

We now study the performance of bottom-up algorithms for problems that match vertex branching, on graphs of bounded maximum degree. We first deal with the case of graphs of maximum degree 3, followed by the general case of graphs with bounded maximum degree.

Proposition 5. *Any problem that matches vertex branching and is polynomial on graphs of maximum degree* 2 *can be solved on graphs of maximum degree* 3 *within time* $O^*(2^{3n/8})$.

A rather immediate corollary is that any problem that matches vertex branching and is polynomial on graphs of maximum degree 2 can be solved on graphs of maximum degree Δ with running time $O^*(2^{n(1-(5/8)^{\Delta-2})})$. Unfortunately, for $\Delta \geq 4$, result from Proposition 4 overlap this one.

In case we can only make the weaker hypothesis that the problem is polynomial on totally disconnected graphs (in fact, we need a somewhat stronger hypothesis, namely, polynomiality on collection of bounded cliques), it is still possible to improve the $O^*(2^{3n/4})$ result from Theorem 3, if we know that our graph has maximum degree 3 instead of average degree 3 or less:

Proposition 6. *Any problem that matches vertex branching and is polynomial on collections of cliques of bounded cardinality can be solved on graphs of maximum degree* 3 *within time* $O^*(2^{2n/3})$.

Proposition 7. *Any problem that matches vertex branching and is polynomial on graphs of maximum degree* 2 *can be solved on graphs of average degree* $d \leq 3$ *with running time* $O^*(2^{21n/46})$.

Thus, the following theorem holds:

Theorem 4. *Any problem that matches vertex branching and is polynomial on graphs of maximum degree* 2 *can be solved on graphs of average degree bounded above by* d *within time* $O^*(2^{\frac{d-27/23}{d+1}n})$, *for any* $d \geq 3$.

4.3 Applying the Bottom-Up Scheme for Exact Solution of the f-QIS Problem

Next proposition establishes the possibility to use a vertex branching method directly derived from the bottom-up scheme, for finding an optimal f-QIS within the time stated in Theorem 4.

Proposition 8. MAX QUASI-INDEPENDENT SET *is polynomial on graphs of maximum degree* 2 *or less.*

From the discussion made in this Section, the following result is immediate.

Theorem 5. *Optimal* MAX QUASI-INDEPENDENT SET*-solutions in graphs of average degree* $\leq d$ *can be found in time* $O^*(2^{\frac{d-27/23}{d+1}n})$.

For instance, MAX QUASI-INDEPENDENT SET on graphs of average degree 3 can be solved in time $O^*(2^{\frac{21}{46}n})$, while in graphs of average degree 4, the corresponding time is $O^*(2^{\frac{13}{23}n})$.

5 Approximation Algorithms

5.1 Approximation of γ-QIS

When γ is bounded from below by a fixed constant. In this case, things are rather optimistic, since the following result holds.

Theorem 6. *Consider the* γ*-QIS problem when* γ *is bounded below by a positive constant* c*. For any fixed* $k \leq \Delta$*, a solution of size at least* k/Δ *the optimal can be computed within polynomial time.*

Corollary 1. *If* γ *is bounded below by some positive constant, then* γ*-QIS is polynomial for graphs with bounded degree.*

A greedy algorithm. In this subsection, a greedy algorithm for computing a γ-QIS is discussed; the solution is initialized to some independent set S, and at each step a vertex of minimum degree to the current solution is being inserted; the insertions keep on, until the largest solution, respecting the sparsity specification, is reached.

procedure γ_QIS ($G(V,E)$: graph, $0 \leq \gamma \leq 1$: real, $S \subseteq V$: some independent set)
  $Q \leftarrow S$;
  $Q' \leftarrow S$;
  while ($|Q'| \leq |V|$)
    pick $v \in V \setminus Q'$ such that $\delta_{Q'}(v) = \min$ (break ties arbitrarily)
    $Q' \leftarrow Q' \cup \{v\}$;
    if ($|E[Q']| \leq \gamma\binom{|Q'|}{2}$) **and** ($|Q| \leq |Q'|$)
      $Q \leftarrow Q'$;
    endif;
  endwhile;
  return(Q);
end γ_QIS

Obviously, γ_QIS always returns a solution, if S is set to some γ-QIS (any independent set in G, for instance the empty set, would do).

As it has already been mentioned, the non-hereditary character of $\gamma - QIS$ is reflected to the algorithm by the fact that it may some Q' produced during the execution of the algorithm be infeasible while after some later vertex-insertions it may become feasible. This non-hereditary character of the problem is a major difficulty for a more refined analysis of algorithm γ-QIS. The following lemma gives a lower bound on the size of the solutions returned by this algorithm.

Lemma 5. *Let q be the size of the γ-QIS returned by the algorithm, where S has been initialized to some independent vertex set. It holds that $q > \frac{\alpha-1}{\sqrt{1-\gamma}}$ where α is the size of Q during the last step of the algorithm's execution before the first edges insertion.*

Combining Lemma 5 and item 1 of Lemma 2, we finally get:

Theorem 7. *For the γ-QIS problem it is possible to find in polynomial time a solution of size q achieving approximation ratio $\frac{q^*}{q} \leq \frac{\Delta\alpha_{\min}}{\alpha-1}\sqrt{1-\gamma}$, where q^* is the size of an optimal quasi-independent set and $\alpha_{\min}$ the size of a minimum independent dominating set of the input graph. This ratio tends to $\Delta\sqrt{1-\gamma}$.*

Algorithm γ-QIS has been run on 20 randomly generated graphs of each size (10, 20 and 30 vertices; edges in an instance have been generated with a probability p, $0.1 < p < 0.5$). Optimal solutions have been computed with the exact method of Section 4. The following table gives a summary of the experimental results obtained. It contains for every value of γ, the worst, best and average ratios and the percentage of optima returned by the algorithm.

γ value	Worst ratio	Best ratio	Average ratio	% of optimal solutions
$0.2d(G)$	0.667	1	0.947	66.667%
$0.4d(G)$	0.7	1	0.969	75%
$0.6d(G)$	0.75	1	0.983	83.333%
$0.8d(G)$	0.905	1	0.996	93.333%
$1/n$	0.667	1	0.96	75%
$1/\sqrt{n}$	0.778	1	0.98	85%
$\log(n)/n$	0.778	1	0.973	76.667%

One may remark that the more the density of the sought subgraph comes close to the density of the instance, the best the quality of the returned solution is.

Moderately exponential approximation for γ-QIS. We finish the approximation section for γ-QIS by showing how it can be approximated within any constant ratio by exponential algorithms with running time better than that of an exact computation.

Theorem 8. *For any $k \geq 1$, it is possible to compute a γ-QIS of size at least $1/k$ of the optimal, within time $O^*(2^{(\log_2(k+1)-k/(k+1)\log_2 k)n})$.*

The following result exhibits a further link between γ-QIS and MAX INDEPENDENT SET.

Theorem 9. *Given some algorithm that computes an exact solution for* MAX INDEPENDENT SET *on G within time $O^*(c^n)$, for some constant c, a γ-QIS of size at least $1+\gamma n/2$ can be computed within time $O^*(c^n)$.*

5.2 Approximation of k-QIS

In this section we deal with polynomial approximation of k-QIS. We propose a greedy algorithm for that purpose, based upon the same idea as γ_QIS presented above; however, k-QIS being a hereditary problem, the algorithm stops as soon as it finds the first vertex whose insertion violates the condition on the number of edges allowed in the solution.

Procedure k_QIS($G(V,E)$: Graph, k: integer≥ 0, S: some independent set)
  $Q \leftarrow S$;
  while ($|E(Q)| \leq k$)
    pick $v \in V \setminus Q$ such that $\delta_Q(v) = \min$ (break ties arbitrarily)
    $Q \leftarrow Q \cup \{v\}$;
  endwhile
  return$(Q \setminus \{v\})$;
end k_QIS

Theorem 10. *For the k-QIS problem it is possible to find in polynomial time a solution of size q achieving approximation ratio:*

$$\frac{q^*}{q} \leq \frac{\alpha^*\theta(S) + k\theta(Q)}{\alpha\theta(Q)+k} \leq \max\left\{\frac{\alpha^*}{\alpha}, \theta(Q)\right\}$$

where q^* is the size of the optimal, α^* is the size of a maximum independent set in G and α the size of some independent set.

Recall that the best approximation ratio (as function of Δ) known for MAX INDEPENDENT SET is $(\Delta+2)/3$ and is guaranteed by the natural greedy MAX INDEPENDENT SET-algorithm [7].

Suppose first that $\theta(Q) \geq 3$. Then, by item 2 of Lemma 2, the approximation ratio of algorithm k_QIS is bounded from above by $\Delta/3$. Assume now that $\theta(Q) \leq 3$. Then, by Theorem 10, the approximation ratio of the algorithm is bounded above by $\max\left\{\frac{\alpha^*}{\alpha}, 6\right\} \leq \frac{\alpha^*}{\alpha} \leq \frac{\Delta+2}{3}$, and the following holds.

Corollary 2. *k-QIS is approximable in polynomial time within ratio $(\Delta+2)/3$.*

Another set of tests have been implemented in order to experimentally observe the behavior of performance of algorithm k_QIS presented for several values of k. As for the case of γ-QIS, we have used the exact method of Section 4 to compute optimal solutions of the test instances. Instances have been generated randomly, following a probability p, $0.1 < p < 0.5$, for an edge to be present in the instance graph. The tests indicate that the algorithm performs fairly well in small instances. A summary of the obtained results is given in the following table.

k value	Worst ratio	Best ratio	Average ratio	% of optimal solutions
$2\sqrt{m}$	0.93	1	0.99	90%
$\sqrt{n}$	0.83	1	0.98	80%
$\log(m)$	0.83	1	0.98	80%
$\log(n)$	0.83	1	0.97	75%
$m/2$	0.96	1	0.99	92.5%
$m/3$	0.96	1	0.99	95%
$n/2$	0.90	1	0.99	90%
$n/3$	0.90	1	0.99	87.5%

References

1. Abello, J., Resende, M.G.C., Sudarsky, S.: Massive quasi-clique detection. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 598–612. Springer, Heidelberg (2002)
2. Asahiro, Y., Iwama, K., Tamaki, H., Tokuyama, T.: Greedily finding a dense subgraph. Journal of Algorithms 34(2), 203–221 (2000)
3. Boginski, V., Butenko, S., Pardalos, P.: Mining market data: a network approach. Computers and Operations Research (2005), http://www.sciencedirect.com
4. Bourgeois, N., Giannakos, A., Lucarelli, G., Milis, I., Paschos, V.T., Pottié, O.: The The Max Quasi-Independent Set Problem. Cahier du LAMSADE (292) (2010)
5. Corneil, D.G., Perl, Y.: Clustering and domination in perfect graphs. Discrete Applied Mathematics 9, 27–39 (1984)
6. Feige, U., Kortsarz, G., Peleg, D.: The dense k-subgraph problem. Algorithmica 29(3), 410–421 (2001)
7. Halldórsson, M.M., Radhakrishnan, J.: Greed is good: approximating independent sets in sparse and bounded-degree graphs. In: Proceedings of STOC 1994, pp. 439–448 (1994)
8. Hartwell, L.H., Hopfield, J.J., Leibler, S., Murray, A.W.: From molecular to modular cell biology. Nature 402, C47–C52 (1999)
9. Hochbaum, D.S., Goldschmidt, O.: k-edge subgraph problems. Discrete Applied Mathematics 74(2), 159–169 (1997)
10. Krishnamoorthy, M.S., Deo, N.: Node-deletion NP-complete problems. SIAM J. Comput. 8, 619–625 (1979)
11. Yannakakis, M., Lewis, J.: The node-deletion problem for hereditary properties is NP-complete. Journal of Computer and System Sciences 20(2), 219–230 (1980)
12. Zissimopoulos, V.: Private communication
13. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. In: Proceedings of STOC 2006, pp. 681–690 (2006)

Equilibria in Quantitative Reachability Games*

Thomas Brihaye, Véronique Bruyère, and Julie De Pril**

University of Mons - UMONS
Place du Parc 20, 7000 Mons, Belgium
{thomas.brihaye,veronique.bruyere,julie.depril}@umons.ac.be

Abstract. In this paper, we study turn-based quantitative multiplayer non zero-sum games played on finite graphs with reachability objectives. In this framework each player aims at reaching his own goal as soon as possible. We prove existence of finite-memory Nash (resp. secure) equilibria in multiplayer (resp. two-player) games.

Keywords: Nash equilibrium, Turn-based quantitative game, Secure equilibrium.

1 Introduction

General framework. The construction of correct and efficient computer systems (hardware or software) is recognized as an extremely difficult task. To support the design and verification of such systems, mathematical logic, automata theory [10] and more recently model-checking [7] have been intensively studied. The model-checking approach, which is now an important part of the design cycle in industries, have proved its efficiency when applied to systems which can be accurately modeled as a finite-state automaton. In contrast, the application of these techniques to computer software, complex systems like embedded systems or distributed systems has been less successful. This could be partly explained by the following reasons: classical automata-based models do not faithfully capture the complex interactive behavior of modern computational systems that are usually composed of several interacting components, also interacting with an environment that is only partially under control. Recent research works show that it is suitable to generalize automata models used in the classical approach to verification, with the more flexible and mathematically deeper game-theoretic framework [13,14].

Game theory meets automata theory. The basic framework that extends computational models with concepts from game theory is the so-called two-player zero-sum games played on graphs [8]. Many problems in verification and design of reactive systems can be modeled with this approach, like modeling controller-environment interactions. Given a model of a system interacting with a hostile

* This work has been partly supported by the ESF project GASICS and a grant from the National Bank of Belgium.

** This author is supported by a grant from L'Oreal-UNESCO/F.R.S.-FNRS.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 72–83, 2010.

environment, given a control objective (like preventing the system to reach some bad configurations), the controller synthesis problem asks to build a controller ensuring that the control objective is enforced whatever the environment will do. Two-player zero-sum games played on graphs are adequate models to solve this problem [15]. Moves of Player 1 model actions of the controller whereas moves of Player 2 model the uncontrollable actions of the environment, and a winning strategy for Player 1 is an abstract form of a control program that enforces the control objective.

The controller synthesis problem is suitable to model purely antagonist interactions between a controller and a hostile environment. However in order to study more complex systems with more than two components whose objectives are not necessarily antagonists, we need multiplayer and non zero-sum games to model them adequately. Moreover, we are not looking for winning strategies, but rather try to find relevant notions of equilibria, for instance the famous notion of Nash equilibria [13]. On the other hand, only qualitative objectives have been considered so far to specify, for example, that a player must be able to reach a target set of states in the underlying game graph. But, in line with the previous point, we also want to express and solve games for quantitative objectives such as forcing the game to reach a particular set of states within a given time bound, or within a given energy consumption limit. In summary, we need to study *equilibria* for *multiplayer non zero-sum* games played on graphs with *quantitative* objectives. This article provides some new results in this research direction.

Related work. Several recent papers have considered two-player zero-sum games played on finite graphs with regular objectives enriched by some *quantitative aspects*. Let us mention some of them: games with *finitary objectives* [6], games with *prioritized requirements* [1], *request-response games* where the waiting times between the requests and the responses are minimized [11,16], and games whose winning conditions are expressed via *quantitative languages* [2].

Other works concern qualitative non zero-sum games. The notion of secure equilibrium, an interesting refinement of Nash equilibrium, has been introduced in [5]. It has been proved that a unique secure equilibrium always exists for two-player non zero-sum games with regular objectives. In [9], general criteria ensuring existence of Nash equilibria, subgame perfect equilibria (resp. secure equilibria) are provided for n-player (resp. 2-player) games, as well as complexity results.

Finally, we mention reference [3] that combines both the quantitative and the non zero-sum aspects. It is maybe the nearest related work compared to us, however the framework and the objectives are pretty different. In [3], the authors study games played on graphs with terminal vertices where quantitative payoffs are assigned to the players. These games may have cycles but all the infinite plays form a single outcome (like in chess where every infinite play is a draw). In that paper, criteria are given that ensure existence of Nash (resp. subgame perfect) equilibria in pure and memoryless strategies.

Our contribution. We here study turn-based quantitative multiplayer non zero-sum games played on finite graphs with reachability objectives. In this framework

each player aims at reaching his own goal as soon as possible. We focus on existence results for two solution concepts: Nash equilibrium and secure equilibrium. We prove existence of Nash (resp. secure) equilibria in n-player (resp. 2-player) games. Moreover, we show that these equilibria can be chosen with finite memory. Our results are not a direct consequence of the existing results in the qualitative framework, they require some new proof techniques. To the best of our knowledge, this is the first general result about existence of equilibria in quantitative multiplayer games played on graphs.

Organization of the paper. Section 2 is dedicated to definitions. We present the games and the equilibria we study. In Section 3 we first prove an existence result for Nash equilibria and provide the finite-memory characterization. Existence of secure equilibria in two-player games is then established. Detailed proofs and examples can be found in [4].

2 Preliminaries

2.1 Definitions

We consider here *quantitative* games played on a graph where all the players have *reachability objectives*. It means that, given a certain set of vertices Goal_i, each player i wants to reach one of these vertices as soon as possible.

This section is mainly inspired by reference [9].

Definition 1. *An* infinite turn-based quantitative multiplayer reachability game *is a tuple* $\mathcal{G} = (\Pi, V, (V_i)_{i\in\Pi}, v_0, E, (\mathsf{Goal}_i)_{i\in\Pi})$ *where*

- Π *is a finite set of players,*
- $G = (V, (V_i)_{i\in\Pi}, v_0, E)$ *is a finite directed graph where* V *is the set of vertices,* $(V_i)_{i\in\Pi}$ *is a partition of* V *into the state sets of each player,* $v_0 \in V$ *is the initial vertex, and* $E \subseteq V \times V$ *is the set of edges, and*
- $\mathsf{Goal}_i \subseteq V$ *is the goal set of player* i*.*

We assume that each vertex has at least one outgoing edge. The game is played as follows. A token is first placed on the vertex v_0. Player i, such that $v_0 \in V_i$, has to choose one of the outgoing edges of v_0 and put the token on the vertex v_1 reached when following this edge. Then, it is the turn of the player who owns v_1. And so on.

A *play* $\rho \in V^\omega$ (respectively a *history* $h \in V^+$) of $\mathcal{G}$ is an *infinite* (respectively a *finite*) path through the graph G starting from vertex v_0. Note that a history is always non empty because it starts with v_0. The set $H \subseteq V^+$ is made up of all the histories of $\mathcal{G}$. A *prefix* (respectively *proper prefix*) $\mathfrak{p}$ of a history $h = h_0 \dots h_k$ is a finite sequence $h_0 \dots h_l$, with $l \leq k$ (respectively $l < k$), denoted by $\mathfrak{p} \leq h$ (respectively $\mathfrak{p} < h$). We similarly consider a prefix $\mathfrak{p}$ of a play ρ, denoted by $\mathfrak{p} < \rho$.

We say that a play $\rho = \rho_0\rho_1 \dots$ *visits* a set $S \subseteq V$ (respectively a vertex $v \in V$) if there exists $l \in \mathbb{N}$ such that ρ_l is in S (respectively $\rho_l = v$). The same terminology also stands for a history h. Similarly, we say that ρ *visits* S *after* (respectively

in) *a prefix* $\rho_0 \dots \rho_k$ if there exists $l > k$ (respectively $l \leq k$) such that ρ_l is in S. For any play ρ we denote by $\mathsf{Visit}(\rho)$ the set of $i \in \Pi$ such that ρ visits Goal_i. The set $\mathsf{Visit}(h)$ for a history h is defined similarly. The function Last returns, given a history $h = h_0 \dots h_k$, the last vertex h_k of h, and the *length* $|h|$ of h is the number k of its *edges*[1].

For any play $\rho = \rho_0 \rho_1 \dots$ of $\mathcal{G}$, we note $\mathsf{Cost}_i(\rho)$ the *cost* of player i, defined by:

$$\mathsf{Cost}_i(\rho) = \begin{cases} l & \text{if } l \text{ is the } \textit{least} \text{ index such that } \rho_l \in \mathsf{Goal}_i, \\ +\infty & \text{otherwise.} \end{cases}$$

We note $\mathsf{Cost}(\rho) = (\mathsf{Cost}_i(\rho))_{i \in \Pi}$ the *cost profile* for the play ρ. The aim of each player i is to *minimize* the cost he has to pay, i.e. reach his goal set Goal_i as soon as possible.

A *strategy* of player i in $\mathcal{G}$ is a function $\sigma : V^* V_i \to V$ assigning to each history hv ending in a vertex v of player i a next vertex $\sigma(hv)$ such that $(v, \sigma(hv))$ belongs to E. We say that a play $\rho = \rho_0 \rho_1 \dots$ of $\mathcal{G}$ is *consistent* with a strategy σ of player i if $\rho_{k+1} = \sigma(\rho_0 \dots \rho_k)$ for all $k \in \mathbb{N}$ such that $\rho_k \in V_i$. The same terminology is used for a history h of $\mathcal{G}$. A *strategy profile* of $\mathcal{G}$ is a tuple $(\sigma_i)_{i \in \Pi}$ where σ_i is a strategy for player i. It determines a unique play of $\mathcal{G}$ consistent with each strategy σ_i, called the *outcome* of $(\sigma_i)_{i \in \Pi}$ and denoted by $\langle (\sigma_i)_{i \in \Pi} \rangle$.

A strategy σ of player i is *memoryless* if σ depends only on the current vertex, i.e. $\sigma(hv) = \sigma(v)$ for all $h \in H$ and $v \in V_i$. More generally, σ is a *finite-memory strategy* if the equivalence relation $\approx_\sigma$ on H defined by $h \approx_\sigma h'$ if $\sigma(h\delta) = \sigma(h'\delta)$ for all $\delta \in V^* V_i$ has finite index. In other words, a finite-memory strategy is a strategy that can be implemented by a finite automaton with output. A strategy profile $(\sigma_i)_{i \in \Pi}$ is called *memoryless* or *finite-memory* if each σ_i is a memoryless or a finite-memory strategy, respectively.

We now introduce the notion of *Nash equilibrium* and *secure equilibrium*.

Definition 2. *A strategy profile* $(\sigma_i)_{i \in \Pi}$ *of a game* $\mathcal{G}$ *is a* Nash equilibrium *if for all player* $j \in \Pi$ *and for all strategy* σ'_j *of player* j*, we have:*

$$\mathsf{Cost}_j(\rho) \leq \mathsf{Cost}_j(\rho')$$

where $\rho = \langle (\sigma_i)_{i \in \Pi} \rangle$ *and* $\rho' = \langle \sigma'_j, (\sigma_i)_{i \in \Pi \setminus \{j\}} \rangle$.

This definition means that player j (for all $j \in \Pi$) has no incentive to deviate since he increases his cost when using σ'_j instead of σ_j. A strategy σ'_j such that $\mathsf{Cost}_j(\rho) > \mathsf{Cost}_j(\rho')$ is called a *profitable deviation* for player j with respect to $(\sigma_i)_{i \in \Pi}$. In this case either player j pays an infinite cost for ρ and a finite cost for ρ' (ρ' visits Goal_j, but ρ does not), or player j pays a finite cost for ρ and a strictly lower cost for ρ' (ρ' visits Goal_j earlier than ρ does).

In order to define the notion of secure equilibrium[2] we first need to associate an appropriate binary relation $\prec_j$ on cost profiles with each player $j \in \Pi$. Given two cost profiles $(x_i)_{i \in \Pi}$ and $(y_i)_{i \in \Pi}$:

[1] Note that the length is not defined as the number of vertices.

[2] Our definition naturally extends the notion of *secure equilibrium* proposed in [5] to the quantitative reachability framework. A longer discussion comparing the two notions can be found in [4].

$$(x_i)_{i\in\Pi} \prec_j (y_i)_{i\in\Pi} \quad \text{iff} \quad (x_j > y_j) \vee (x_j = y_j \wedge \forall k\ x_k \leq y_k \wedge \exists k\ x_k < y_k)\,.$$

We then say that *player j prefers $(y_i)_{i\in\Pi}$ to $(x_i)_{i\in\Pi}$*. In other words, player j prefers a cost profile to another either if he can decrease his own cost, or if he can increase the costs of all his opponents, among which one is strictly increased, while keeping his own cost.

Definition 3. *A strategy profile $(\sigma_i)_{i\in\Pi}$ of a game $\mathcal{G}$ is a* secure equilibrium *if for all players $j \in \Pi$, there does not exist a strategy σ'_j of player j such that:*

$$\mathsf{Cost}(\rho) \prec_j \mathsf{Cost}(\rho')$$

where $\rho = \langle(\sigma_i)_{i\in\Pi}\rangle$ and $\rho' = \langle\sigma'_j, (\sigma_i)_{i\in\Pi\setminus\{j\}}\rangle$.

In other words, player $j \in \Pi$ (for all $j \in \Pi$) has no incentive to deviate, with respect to the relation $\prec_j$. Note that any secure equilibrium is a Nash equilibrium. A strategy σ'_j such that $\mathsf{Cost}(\rho) \prec_j \mathsf{Cost}(\rho')$ is called a *$\prec_j$-profitable deviation* for player j with respect to $(\sigma_i)_{i\in\Pi}$.

Definition 4. *The type of a Nash or a secure equilibrium $(\sigma_i)_{i\in\Pi}$ in a reachability game $\mathcal{G}$ is the set of players $j \in \Pi$ such that the outcome ρ of $(\sigma_i)_{i\in\Pi}$ visits Goal_j. It is denoted by $\mathtt{Type}((\sigma_i)_{i\in\Pi})$.*

In other words, $\mathtt{Type}((\sigma_i)_{i\in\Pi}) = \mathsf{Visit}(\rho)$.

The previous definitions are illustrated on a simple two-player game in the technical report [4].

The questions studied in this article are the following ones:

Problem 1. *Given $\mathcal{G}$ a quantitative multiplayer reachability game, does there exist a Nash equilibrium (respectively a secure equilibrium) in $\mathcal{G}$?*

Problem 2. *Given a Nash equilibrium (respectively a secure equilibrium) in a quantitative multiplayer reachability game $\mathcal{G}$, does there exist a memoryless or a finite-memory Nash equilibrium (respectively secure equilibrium) with the same type?*

We provide partial positive answers in Section 3. These problems have been investigated in the qualitative framework (see [9]). In [4], we show that Problems 1 and 2 can not be reduced to problems on qualitative games.

2.2 Unraveling

In the proofs of this article we need to unravel the graph $G = (V, (V_i)_{i\in\Pi}, v_0, E)$ from the initial vertex v_0, which ends up in an *infinite tree*, denoted by T. This tree can be seen as a new graph where the set of vertices is the set H of histories of $\mathcal{G}$, the initial vertex is v_0, and a pair $(hv, hvv') \in H \times H$ is an edge of T if $(v, v') \in E$. A history h is a vertex of player i in T if $\mathsf{Last}(h) \in V_i$, and it belongs to the goal set of player i if $\mathsf{Last}(h) \in \mathsf{Goal}_i$.

We denote by $\mathcal{T}$ the related game. This game $\mathcal{T}$ played on the unraveling T of G is equivalent to the game $\mathcal{G}$ played on G in the following sense. A play $(\rho_0)(\rho_0\rho_1)(\rho_0\rho_1\rho_2)\dots$ in $\mathcal{T}$ induces a unique play $\rho = \rho_0\rho_1\rho_2\dots$ in $\mathcal{G}$, and conversely. Thus, we denote a play in $\mathcal{T}$ by the respective play in $\mathcal{G}$. The bijection between plays of $\mathcal{G}$ and plays of $\mathcal{T}$ allows us to use the same cost function Cost, and to transform easily strategies in $\mathcal{G}$ to strategies in $\mathcal{T}$ (and conversely).

We also need to study the tree T limited to a certain depth $d \geq 0$: we note $\mathsf{Trunc}_d(T)$ the *truncated tree of* T *of depth* d and $\mathsf{Trunc}_d(\mathcal{T})$ the *finite* game played on $\mathsf{Trunc}_d(T)$. More precisely, the set of vertices of $\mathsf{Trunc}_d(T)$ is the set of histories $h \in H$ of length $\leq d$; the edges of $\mathsf{Trunc}_d(T)$ are defined in the same way as for T except that for the histories h of length d, there exists no edge (h, hv). A play ρ in $\mathsf{Trunc}_d(\mathcal{T})$ corresponds to a history of $\mathcal{G}$ of length *equal to* d. The notions of cost and strategy are defined exactly like in the game $\mathcal{T}$, but limited to the depth d. For instance, a player pays an infinite cost for a play ρ (of length d) if his goal set is not visited by ρ.

3 Nash Equilibria and Secure Equilibria

From now on we will often use the term *game* to denote a quantitative multiplayer reachability game according to Definition 1.

3.1 Existence of a Nash Equilibrium

In this section we positively solve Problem 1 for Nash equilibria.

Theorem 5. *In every quantitative multiplayer reachability game, there exists a finite-memory Nash equilibrium.*

From the previous theorem we can find a Nash equilibrium such that each player pays either an infinite cost, or a cost bounded by $|\Pi| \cdot 2 \cdot |V|$.

The proof of this theorem is based on the following ideas. By Kuhn's theorem (Theorem 6), there exists a Nash equilibrium in the game $\mathsf{Trunc}_d(\mathcal{T})$ played on the finite tree $\mathsf{Trunc}_d(T)$, for any depth d. By choosing an adequate depth d, Proposition 8 will enable to extend this Nash equilibrium to a Nash equilibrium in the infinite tree T, and thus in $\mathcal{G}$. Let us detail these ideas.

We first recall Kuhn's theorem [12]. A *preference relation* is a total reflexive transitive binary relation.

Theorem 6 (Kuhn's Theorem). *Let Γ be a* finite *tree and $\mathcal{G}$ a game played on Γ. For each player $i \in \Pi$, let $\precsim_i$ be a preference relation on cost profiles. Then there exists a strategy profile $(\sigma_i)_{i\in\Pi}$ such that for every player $j \in \Pi$ and every strategy σ'_j of player j in $\mathcal{G}$ we have*

$$\mathsf{Cost}(\rho') \precsim_j \mathsf{Cost}(\rho)$$

where $\rho = \langle(\sigma_i)_{i\in\Pi}\rangle$ and $\rho' = \langle\sigma'_j, (\sigma_i)_{i\in\Pi\setminus\{j\}}\rangle$.

Note that $\mathsf{Cost}(\rho') \preceq_j \mathsf{Cost}(\rho)$ means that player j prefers the cost profile of the play ρ than the one of ρ', or they are equivalent for him.

Corollary 7. *Let $\mathcal{G}$ be a game and T be the unraveling of G. Let $\mathsf{Trunc}_d(\mathcal{T})$ be the game played on the truncated tree of T of depth d, with $d \geq 0$. Then there exists a Nash equilibrium in $\mathsf{Trunc}_d(\mathcal{T})$.*

Proof. For each player $j \in \Pi$, we define the relation $\precsim_j$ on cost profiles in the following way: let $(x_i)_{i\in\Pi}$ and $(y_i)_{i\in\Pi}$ be two cost profiles, we say that $(x_i)_{i\in\Pi} \precsim_j (y_i)_{i\in\Pi}$ iff $x_j \geq y_j$. It is clearly a preference relation which captures the Nash equilibrium. The strategy profile $(\sigma_i)_{i\in\Pi}$ of Kuhn's theorem is then a Nash equilibrium in $\mathsf{Trunc}_d(\mathcal{T})$. □

The next proposition states that it is possible to extend a Nash equilibrium in $\mathsf{Trunc}_d(\mathcal{T})$ to a Nash equilibrium in the game $\mathcal{T}$, if the depth d is equal to $(|\Pi| + 1)\cdot 2\cdot|V|$. We obtain Theorem 5 as a consequence of Corollary 7 and Proposition 8.

Proposition 8. *Let $\mathcal{G}$ be a game and T be the unraveling of G. Let $\mathsf{Trunc}_d(\mathcal{T})$ be the game played on the truncated tree of T of depth $d = (|\Pi| + 1) \cdot 2 \cdot |V|$. If there exists a Nash equilibrium in the game $\mathsf{Trunc}_d(\mathcal{T})$, then there exists a finite-memory Nash equilibrium in the game $\mathcal{T}$.*

The proof of Proposition 8 roughly works as follows. Let $(\sigma_i)_{i\in\Pi}$ be a Nash equilibrium in $\mathsf{Trunc}_d(\mathcal{T})$. A well-chosen prefix $\alpha\beta$, with β being a cycle, is first extracted from the outcome ρ of $(\sigma_i)_{i\in\Pi}$. The outcome of the required Nash equilibrium $(\tau_i)_{i\in\Pi}$ in $\mathcal{T}$ will be equal to $\alpha\beta^\omega$. As soon as a player deviates from this play, all the other players form a coalition against him to punish him in a way that this deviation is not profitable for him. These ideas are detailed in the next two lemmas whose complete proofs can be found in [4, Section C].

In Lemma 10 we consider the qualitative two-player zero-sum game $\mathcal{G}_j$ played on the graph G, where player j plays in order to reach his goal set Goal_j, against the coalition of all other players that wants to prevent him from reaching his goal set. Player j plays on the vertices from V_j and the coalition on $V \setminus V_j$. We have the following proposition (see [8]).

Proposition 9. *Let $\mathcal{G}_j = (V, V_j, V \setminus V_j, E, \mathsf{Goal}_j)$ be the qualitative two-player zero-sum reachability game associated to player j. Then player j has a memoryless strategy ν_j that enables him to reach Goal_j within $|V| - 1$ edges from each vertex v from which he wins the game $\mathcal{G}_j$. On the contrary, the coalition has a memoryless strategy ν_{-j} that forces the play to stay in $V \setminus \mathsf{Goal}_j$ from each vertex v from which it wins the game $\mathcal{G}_j$.*

The play ρ of Lemma 10 is illustrated in Figure 1.

Lemma 10. *Let $d \geq 0$. Let $(\sigma_i)_{i\in\Pi}$ be a Nash equilibrium in $\mathsf{Trunc}_d(\mathcal{T})$ and ρ the (finite) outcome of $(\sigma_i)_{i\in\Pi}$. Suppose that ρ has a prefix $\alpha\beta\gamma$, where β contains at least one vertex, such that*

$$\mathsf{Visit}(\alpha) = \mathsf{Visit}(\alpha\beta\gamma)$$
$$\mathsf{Last}(\alpha) = \mathsf{Last}(\alpha\beta)$$
$$|\alpha\beta| \leq l \cdot |V|$$
$$|\alpha\beta\gamma| = (l+1) \cdot |V|$$

for some $l \geq 1$.
Let $j \in \Pi$ *be such that* α *does not visit* Goal_j. *Consider the qualitative two-player zero-sum game* $\mathcal{G}_j = (V, V_j, V \setminus V_j, E, \mathsf{Goal}_j)$. *Then for all histories* hu *of* $\mathcal{G}$ *consistent with* $(\sigma_i)_{i\in\Pi\setminus\{j\}}$ *and such that* $|hu| \leq |\alpha\beta|$, *the coalition of the players* $i \neq j$ *wins the game* $\mathcal{G}_j$ *from* u.

Condition $\mathsf{Visit}(\alpha) = \mathsf{Visit}(\alpha\beta\gamma)$ means that if Goal_i is visited by $\alpha\beta\gamma$, it has already been visited by α. Condition $\mathsf{Last}(\alpha) = \mathsf{Last}(\alpha\beta)$ means that β is a cycle.

This lemma means in particular that the players $i \neq j$ can play together to prevent player j from reaching his goal set Goal_j, in case he deviates from the play $\alpha\beta$ (as $\alpha\beta$ is consistent with $(\sigma_i)_{i\in\Pi\setminus\{j\}}$). We denote by ν_{-j} the memoryless winning strategy of the coalition. For each player $i \neq j$, let $\nu_{i,j}$ be the memoryless strategy of player i in $\mathcal{G}$ induced by ν_{-j}.

Lemma 11 states that one can define a Nash equilibrium $(\tau_i)_{i\in\Pi}$ in the game $\mathcal{T}$, based on the Nash equilibrium $(\sigma_i)_{i\in\Pi}$ in the game $\mathsf{Trunc}_d(\mathcal{T})$.

Lemma 11. *Let* $d \geq 0$. *Let* $(\sigma_i)_{i\in\Pi}$ *be a Nash equilibrium in* $\mathsf{Trunc}_d(\mathcal{T})$ *and* $\alpha\beta\gamma$ *be a prefix of* $\rho = \langle(\sigma_i)_{i\in\Pi}\rangle$ *as defined in Lemma 10. Then there exists a Nash equilibrium* $(\tau_i)_{i\in\Pi}$ *in the game* $\mathcal{T}$. *Moreover* $(\tau_i)_{i\in\Pi}$ *is finite-memory, and* $\mathtt{Type}((\tau_i)_{i\in\Pi}) = \mathsf{Visit}(\alpha)$.

Proof. Let $\Pi = \{1, \ldots, n\}$. As α and β end in the same vertex, we can consider the infinite play $\alpha\beta^\omega$ in the game $\mathcal{T}$. Without loss of generality we can order the players $i \in \Pi$ so that

$$\forall i \leq k \qquad \mathsf{Cost}_i(\alpha\beta^\omega) < +\infty \quad (\alpha \text{ visits } \mathsf{Goal}_i)$$
$$\forall i > k \qquad \mathsf{Cost}_i(\alpha\beta^\omega) = +\infty \quad (\alpha \text{ does not visit } \mathsf{Goal}_i)$$

where $0 \leq k \leq n$. In the second case, notice that ρ could visit Goal_i (but after the prefix $\alpha\beta\gamma$).

The Nash equilibrium $(\tau_i)_{i\in\Pi}$ required by Lemma 11 is intuitively defined as follows. First the outcome of $(\tau_i)_{i\in\Pi}$ is exactly $\alpha\beta^\omega$. Secondly the first player j who deviates from $\alpha\beta^\omega$ is punished by the coalition of the other players in the following way. If $j \leq k$ and the deviation occurs in the tree $\mathsf{Trunc}_d(T)$, then the coalition plays according to $(\sigma_i)_{i\in\Pi\setminus\{j\}}$ in this tree. It prevents player j from reaching his goal set Goal_j faster than in $\alpha\beta^\omega$. And if $j > k$, the coalition plays according to $(\nu_{i,j})_{i\in\Pi\setminus\{j\}}$ (given by Lemma 10) so that player j does not reach his goal set at all.

We begin by defining a punishment function P on the vertex set H of T such that $P(h)$ indicates the first player j who has deviated from $\alpha\beta^\omega$, with respect to h. We write $P(h) = \bot$ if no deviation has occurred. For $h \in V^*$ and $v \in V_i$ we let:

$$P(hv) = \begin{cases} \bot & \text{if } P(h) = \bot \text{ and } hv < \alpha\beta^\omega, \\ i & \text{if } P(h) = \bot \text{ and } hv \not< \alpha\beta^\omega, \\ P(h) & \text{otherwise } (P(h) \neq \bot)\,. \end{cases}$$

The Nash equilibrium $(\tau_i)_{i\in\Pi}$ is then defined as follows: let h be a history ending in a vertex of V_i,

$$\tau_i(h) = \begin{cases} v & \text{if } P(h) = \bot \ (h < \alpha\beta^\omega) \text{; such that } hv < \alpha\beta^\omega, \\ \textit{arbitrary} & \text{if } P(h) = i, \\ \nu_{i,P(h)}(h) & \text{if } P(h) \neq \bot, i \text{ and } P(h) > k, \\ \sigma_i(h) & \text{if } P(h) \neq \bot, i,\ P(h) \leq k \text{ and } |h| < d, \\ \textit{arbitrary} & \text{otherwise } (P(h) \neq \bot, i,\ P(h) \leq k \text{ and } |h| \geq d) \end{cases} \tag{1}$$

where *arbitrary* means that the next vertex is chosen arbitrarily (in a memoryless way). Clearly the outcome of $(\tau_i)_{i\in\Pi}$ is the play $\alpha\beta^\omega$, and $\mathtt{Type}((\tau_i)_{i\in\Pi})$ is equal to $\mathsf{Visit}(\alpha)$ $(= \mathsf{Visit}(\alpha\beta))$.

It remains to prove that $(\tau_i)_{i\in\Pi}$ is a finite-memory Nash equilibrium in the game $\mathcal{T}$. The end of this proof can be found in [4, Section C]. □

We can now proceed to the proof of Proposition 8.

Proof (of Proposition 8). Let $\Pi = \{1, \ldots, n\}$ and $d = (n+1) \cdot 2 \cdot |V|$. Let $(\sigma_i)_{i\in\Pi}$ be a Nash equilibrium in the game $\mathsf{Trunc}_d(\mathcal{T})$ and ρ its outcome.

To be able to use Lemmas 10 and 11, we consider the prefix $\mathfrak{pq}$ of ρ of minimal length such that

$$\exists\, l \geq 1 \qquad \begin{aligned} &|\mathfrak{p}| = (l-1) \cdot |V| \\ &|\mathfrak{pq}| = (l+1) \cdot |V| \\ &\mathsf{Visit}(\mathfrak{p}) = \mathsf{Visit}(\mathfrak{pq})\,. \end{aligned} \tag{2}$$

The following statements are true.

(i) $l \leq 2 \cdot n + 1$.

(ii) If $\mathsf{Visit}(\mathfrak{p}) \subsetneq \mathsf{Visit}(\rho)$, then $l < 2 \cdot n + 1$.

Indeed the first statement results from the fact that in the worst case, the play ρ visits the goal set of a new player in each prefix of length $i \cdot 2 \cdot |V|$, $1 \leq i \leq n$, i.e. $|\mathfrak{p}| = n \cdot 2 \cdot |V|$. It follows that $\mathfrak{pq}$ exists as a prefix of ρ, because the length d of ρ is equal to $(n+1) \cdot 2 \cdot |V|$ by hypothesis. Thus $\mathsf{Visit}(\mathfrak{p}) \subseteq \mathsf{Visit}(\rho)$. Suppose that there exists $i \in \mathsf{Visit}(\rho) \setminus \mathsf{Visit}(\mathfrak{p})$, then ρ visit Goal_i after the prefix $\mathfrak{pq}$ by Equation (2). The second statement follows easily.

Given the length of $\mathfrak{q}$, one vertex of V is visited at least twice by $\mathfrak{q}$. More precisely, we can write

$$\mathfrak{pq} = \alpha\beta\gamma \quad \text{with} \quad \begin{aligned} &\mathsf{Last}(\alpha) = \mathsf{Last}(\alpha\beta) \\ &|\alpha| \geq (l-1) \cdot |V| \\ &|\alpha\beta| \leq l \cdot |V|\,. \end{aligned} \tag{3}$$

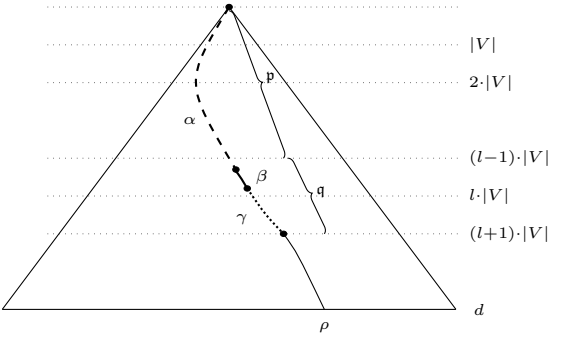


Fig. 1. Slicing of the play ρ in the tree $\mathsf{Trunc}_d(T)$

In particular, $|\mathfrak{p}| \leq |\alpha|$. See Figure 1. We have $\mathsf{Visit}(\alpha) = \mathsf{Visit}(\alpha\beta\gamma)$, and $|\alpha\beta\gamma| = (l+1) \cdot |V|$.

As the hypotheses of Lemmas 10 and 11 are verified, we can apply them in this context to get a Nash equilibrium $(\tau_i)_{i \in \Pi}$ in the game $\mathcal{T}$ with $\mathsf{Type}((\tau_i)_{i \in \Pi}) = \mathsf{Visit}(\alpha)$. □

Proposition 8 asserts that given a game $\mathcal{G}$ and the game $\mathsf{Trunc}_d(\mathcal{T})$ played on the truncated tree of T of a well-chosen depth d, one can lift any Nash equilibrium $(\sigma_i)_{i \in \Pi}$ of $\mathsf{Trunc}_d(\mathcal{T})$ to a Nash equilibrium $(\tau_i)_{i \in \Pi}$ of $\mathcal{G}$. The proof of Proposition 8 states that the type of $(\tau_i)_{i \in \Pi}$ is equal to $\mathsf{Visit}(\alpha)$. We give in [4, Section C] an example that shows that, with this approach, it is impossible to preserve the type of the lifted Nash equilibrium $(\sigma_i)_{i \in \Pi}$.

3.2 Nash Equilibrium with Finite Memory

In this section we study the kind of strategies we can impose for a Nash equilibrium in a quantitative multiplayer reachability game. We show that given a Nash equilibrium, we can construct another Nash equilibrium with the same type such that all its strategies are finite-memory. We then answer to Problem 2 for Nash equilibria.

Theorem 12. *Let $(\sigma_i)_{i \in \Pi}$ be a Nash equilibrium in a quantitative multiplayer reachability game $\mathcal{G}$. Then there exists a finite-memory Nash equilibrium of the same type in $\mathcal{G}$.*

The proof is based on two steps. The first step constructs from $(\sigma_i)_{i \in \Pi}$ another Nash equilibrium $(\tau_i)_{i \in \Pi}$ with the same type such that the play $\langle (\tau_i)_{i \in \Pi} \rangle$ is of the form $\alpha\beta^\omega$ with $\mathsf{Visit}(\alpha) = \mathsf{Type}((\sigma_i)_{i \in \Pi})$. This is possible by first eliminating unnecessary cycles in the play $\langle (\sigma_i)_{i \in \Pi} \rangle$ and then locating a prefix $\alpha\beta$ such that β is a cycle that can be infinitely repeated.

The second step transforms the Nash equilibrium $(\tau_i)_{i \in \Pi}$ into a finite-memory one. For that, we consider the strategy profile $(\tau_i)_{i \in \Pi}$ limited to the tree T truncated at a well-chosen depth.

The detailed proof of Theorem 12 can be found in [4, Section D].

3.3 Existence of a Secure Equilibrium

In this section we positively answer to Problem 1 for secure equilibria in *two-player* games.

Theorem 13. *In every quantitative two-player reachability game, there exists a finite-memory secure equilibrium.*

The proof of this theorem is based on the same ideas as for the proof of Theorem 5 (existence of a Nash equilibrium). By Kuhn's theorem (Theorem 6), there exists a secure equilibrium in the game $\mathsf{Trunc}_d(\mathcal{T})$ played on the finite tree $\mathsf{Trunc}_d(T)$, for any depth d. By choosing an adequate depth d, we are able to extend this secure equilibrium to a secure equilibrium in the infinite tree T, and thus in $\mathcal{G}$. The details of the proof are given in [4, Section E].

4 Conclusion and Perspectives

In this paper, we proved the existence of finite-memory Nash (resp. secure) equilibria for quantitative multiplayer (resp. two-player) reachability games played on finite graphs. We do believe that our results remain true when the model is enriched by allowing positive weights on edges (instead of weight 1 on each edge). Indeed the idea is to replace any edge with a weight $c \geq 1$ by a path of length c composed of c new edges, and use the results proved in this article.

There are several interesting directions for further research. First, we intend to investigate the existence of secure equilibria in the n-player framework. Secondly, we would like to check whether our results remain true when the model is enriched by allowing a n-tuple of non-negative weights on edges (one weight by player). Then, we will also investigate deeper the size of the memory needed in the equilibria. This could be a first step towards a study of the complexity of computing equilibria with certain requirements, in the spirit of [9]. We also intend to look for existence results for *subgame perfect equilibria*. Finally we would like to address these questions for other objectives such as Büchi or request-response.

Acknowledgments. The authors are grateful to Jean-François Raskin and Hugo Gimbert for useful discussions.

References

1. Alur, R., Kanade, A., Weiss, G.: Ranking automata and games for prioritized requirements. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 240–253. Springer, Heidelberg (2008)
2. Bloem, R., Chatterjee, K., Henzinger, T., Jobstmann, B.: Better quality in synthesis through quantitative objectives. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 140–156. Springer, Heidelberg (2009)
3. Boros, E., Gurvich, V.: Why chess and back gammon can be solved in pure positional uniformly optimal strategies. Rutcor Research Report 21-2009, Rutgers University (2009)

4. Brihaye, T., Bruyère, V., De Pril, J.: Equilibria in Quantitative Reachability Games. Technical Report 122 (2010), http://www.ulb.ac.be/di/ssd/cfv/TechReps/TechRep_CFV_2010_122.pdf
5. Chatterjee, K., Henzinger, T., Jurdziński, M.: Games with secure equilibria. Theoretical Computer Science 365(1-2), 67–82 (2006)
6. Chatterjee, K., Henzinger, T.A.: Finitary winning in omega-regular games. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 257–271. Springer, Heidelberg (2006)
7. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (2000)
8. Grädel, E., Thomas, W., Wilke, T.: Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
9. Grädel, E., Ummels, M.: Solution concepts and algorithms for infinite multiplayer games. In: Apt, K., van Rooij, R. (eds.) New Perspectives on Games and Interaction, Texts in Logic and Games, vol. 4, pp. 151–178. Amsterdam University Press (2008)
10. Hopcroft, J.E., Ullman, J.D.: Introduction to automata theory, languages, and computation. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Co., Reading (1979)
11. Horn, F., Thomas, W., Wallmeier, N.: Optimal strategy synthesis in request-response games. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 361–373. Springer, Heidelberg (2008)
12. Kuhn, H.: Extensive games and the problem of information. Classics in Game Theory, 46–68 (1953)
13. Nash, J.: Equilibrium points in n-person games. Proceedings of the National Academy of Sciences of the United States of America 36(1), 48–49 (1950)
14. Osborne, M., Rubinstein, A.: A course in game theory. MIT Press, Cambridge (1994)
15. Thomas, W.: On the synthesis of strategies in infinite games. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995)
16. Zimmermann, M.: Time-optimal winning strategies for poset games. In: Maneth, S. (ed.) CIAA 2009. LNCS, vol. 5642, pp. 217–226. Springer, Heidelberg (2009)

Quotient Complexity of Closed Languages*

Janusz Brzozowski[1], Galina Jirásková[2], and Chenglong Zou[1]

[1] David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada N2L 3G1
{brzozo@,c2zou@student.math.}uwaterloo.ca
[2] Mathematical Institute, Slovak Academy of Sciences, Grešákova 6, 040 01 Košice, Slovakia
jiraskov@saske.sk

Abstract. A language L is prefix-closed if, whenever a word w is in L, then every prefix of w is also in L. We define suffix-, factor-, and subword-closed languages in an analogous way, where by subword we mean subsequence. We study the quotient complexity (usually called state complexity) of operations on prefix-, suffix-, factor-, and subword-closed languages. We find tight upper bounds on the complexity of the subword-closure of arbitrary languages, and on the complexity of boolean operations, concatenation, star, and reversal in each of the four classes of closed languages. We show that repeated application of positive closure and complement to a closed language results in at most four distinct languages, while Kleene closure and complement gives at most eight.

Keywords: automaton, closed, factor, language, prefix, quotient, regular operation, state complexity, subword, suffix, upper bound.

1 Introduction

The *state complexity of a regular language* L is the number of states in the minimal deterministic finite automaton (dfa) recognizing L. The *state complexity of an operation* in a subclass $\mathcal{C}$ of regular languages is defined as the worst-case size of the minimal dfa accepting the language resulting from the operation, taken as a function of the quotient complexities of the operands in $\mathcal{C}$.

The first results for the state complexity of reversal are due to Mirkin [21] (1966), and of union, concatenation, and star, to Maslov [20] (1970). For a general discussion of state complexity see [5,27] and the reference lists in those papers. In 1994 the complexity of concatenation, star, left and right quotients, reversal, intersection, and union in regular languages was examined in detail in [28]. The complexity of operations was also considered in several subclasses of regular languages: unary [23,28], finite [27], cofinite [3], prefix-free [14], suffix-free [13], and ideal [7]. These studies show that the complexity can be significantly lower in a subclass than in the general case. Here we examine state complexity in the classes of prefix-, suffix-, factor-, and subword-closed regular languages.

* This work was supported by the Natural Sciences and Engineering Research Council of Canada grant OGP0000871 and by VEGA grant 2/0111/09.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 84–95, 2010.

There are several reasons for considering closed languages. Subword-closed languages were studied in 1969 [12], and also in 1973 [25]. Suffix-closed languages were considered in 1974 [11], and later in [10,15,26]. Factor-closed languages, also called *factorial*, have received some attention, for example, in [2,19]. Subword-closed languages were studied in [22]. The state complexities of the prefix-, suffix-, and factor-closure of a language were examined in [17]. Prefix-closed languages play a role in predictable semiautomata [8]. All four classes of closed languages were examined in [1], and decision problems for closed languages were studied in [9]. A language is a *left ideal* (respectively, *right, two-sided*, *all-sided ideal*) if $L = \Sigma^* L$, (respectively, $L = L\Sigma^*$, $L = \Sigma^* L \Sigma^*$, and $L = \Sigma^* \sqcup\!\sqcup L$, where $\Sigma^* \sqcup\!\sqcup L$ is the shuffle of Σ^* with L). Closed languages are related to ideal languages as follows [1]: A non-empty language is a right (left, two-sided, all-sided) ideal if and only its complement is a prefix (suffix, factor, subword)-closed language. Closed languages are defined by binary relations "is a prefix of" (respectively, "is a suffix of", "is a factor of", "is a subword of") [1], and are special cases of convex languages [1,25]. The fact that the four classes of closed languages are related to each other permits us to obtain many complexity results using similar methods.

2 Quotient Complexity

If Σ is a non-empty finite *alphabet*, then Σ^* is the free monoid generated by Σ. A *word* is any element of Σ^*, and ε is the *empty word*. A *language* over Σ is any subset of Σ^*. The cardinality of a set S is denoted by $|S|$. If $w = uxv$ for some u, v, x in Σ^*, then u is a *prefix* of w, v is a *suffix* of w, and x is a *factor* of w. If $w = w_0 a_1 w_1 \cdots a_n w_n$, where $a_1, \ldots, a_n \in \Sigma$, and $w_0, \ldots, w_n \in \Sigma^*$, then the word $a_1 \cdots a_n$ is a *subword* of w.

A language L is *prefix-closed* if $w \in L$ implies that every prefix of w is also in L. In a similar way, we define *suffix-*, *factor-*, and *subword-closed* languages. A language is *closed* if it is prefix-, suffix-, factor-, or subword-closed.

The following set operations are defined on languages: *complement* ($\overline{L} = \Sigma^* \setminus L$), *union* ($K \cup L$), *intersection* ($K \cap L$), *difference* ($K \setminus L$), and *symmetric difference* ($K \oplus L$). A general *boolean operation* with two arguments is denoted by $K \circ L$. We also define the *product*, usually called *concatenation* or *catenation*, ($KL = \{w \in \Sigma^* \mid w = uv, u \in K, v \in L\}$), (Kleene) *star* ($L^* = \bigcup_{i \geq 0} L^i$), and *positive closure* ($L^+ = \bigcup_{i \geq 1} L^i$). The *reverse* w^R of a word w in Σ^* is defined as follows: $\varepsilon^R = \varepsilon$, and $(wa)^R = aw^R$. The *reverse* of a language L is denoted by L^R and is defined as $L^R = \{w^R \mid w \in L\}$.

Regular languages over an alphabet Σ are languages that can be obtained from the *set of basic languages* $\{\emptyset, \{\varepsilon\}\} \cup \{\{a\} \mid a \in \Sigma\}$, using a finite number of operations of union, product, and star. Such languages are usually denoted by regular expressions. If E is a regular expression, then $\mathcal{L}(E)$ is the language denoted by that expression. For example, $E = (\varepsilon \cup a)^* b$ denotes the language $\mathcal{L}(E) = (\{\varepsilon\} \cup \{a\})^* \{b\}$. We usually do not distinguish notationally between regular languages and regular expressions; the meaning is clear from the context.

A *deterministic finite automaton* (dfa) is a quintuple $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$, where Q is a set of *states*, Σ is the *alphabet*, $\delta : Q \times \Sigma \to Q$ is the *transition function*, q_0 is the *initial state*, and F is the set of *final* or *accepting states*. A *nondeterministic finite automaton* (nfa) is a quintuple $\mathcal{N} = (Q, \Sigma, \eta, Q_0, F)$, where Q, Σ, and F are as in a dfa, $\eta : Q \times \Sigma \to 2^Q$ is the *transition function* and $Q_0 \subseteq Q$ is the *set of initial states*. If η also allows ε, that is, $\eta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$, we call $\mathcal{N}$ an *ε-nfa*.

Our approach to quotient complexity follows closely that of [5]. Since state complexity is a property of a language, it is more appropriately defined in language-theoretic terms. The *left quotient*, or simply *quotient,* of a language L by a word w is the language $L_w = \{x \in \Sigma^* \mid wx \in L\}$. The *quotient complexity* of L is the number of distinct quotients of L, and is denoted by $\kappa(L)$.

Quotients of regular languages [4,5] can be computed as follows: First, the *ε-function* L^ε of a regular language L is $L^\varepsilon = \emptyset$ if $\varepsilon \notin L$ and $L^\varepsilon = \varepsilon$ if $\varepsilon \in L$. The quotient by a letter a in Σ is computed by induction: $b_a = \emptyset$ if $b \in \{\emptyset, \varepsilon\}$ or $b \in \Sigma$ and $b \neq a$, and $b_a = \varepsilon$ if $b = a$; $(\overline{L})_a = \overline{L_a}$; $(K \cup L)_a = K_a \cup L_a$; $(KL)_a = K_a L \cup K^\varepsilon L_a$; $(L^*)_a = L_a L^*$. The quotient by a word w in Σ^* is computed by induction on the length of w: $L_\varepsilon = L$ and $L_{wa} = (L_w)_a$. A quotient L_w is *accepting* if $\varepsilon \in L_w$; otherwise it is *rejecting*.

The *quotient automaton* of a regular language L is $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{L_w \mid w \in \Sigma^*\}$, $\delta(L_w, a) = L_{wa}$, $q_0 = L_\varepsilon = L$, and $F = \{L_w \mid (L_w)^\varepsilon = \varepsilon\}$. This is a minimal dfa for L; so quotient complexity of L equals the state complexity of L. However, there are some advantages to using quotients [5]. To simplify the notation, we write $(L_w)^\varepsilon$ as L_w^ε. Whenever convenient, we use the formulas given in the next proposition to establish upper bounds on quotient complexity.

Proposition 1 ([4,5]). *If K and L are regular languages, then*

$$(\overline{L})_w = \overline{L_w}; \quad (K \circ L)_w = K_w \circ L_w. \tag{1}$$

$$(KL)_w = K_w L \cup K^\varepsilon L_w \cup \Big(\bigcup_{\substack{w=uv \\ u,v \in \Sigma^+}} K_u^\varepsilon L_v \Big). \tag{2}$$

$$(L^*)_\varepsilon = \varepsilon \cup LL^*, \quad (L^*)_w = \Big(L_w \cup \bigcup_{\substack{w=uv \\ u,v \in \Sigma^+}} (L^*)_u^\varepsilon L_v\Big) L^* \ \text{ for } w \in \Sigma^+. \tag{3}$$

3 Closure Operations

Let $\trianglelefteq$ be a partial order on Σ^*; the *$\trianglelefteq$-closure* of a language L is the language ${}_{\trianglelefteq}L = \{x \in \Sigma^* \mid x \trianglelefteq w \text{ for some } w \in L\}$. We use $\leq$, $\preceq$, $\sqsubseteq$, $\Subset$ for the relations "is a prefix of", "is a suffix of", "is a factor of", "is a subword of", respectively.

The worst-case quotient complexity for closure was studied by Kao, Rampersad, and Shallit [17]. For suffix-closure, the bound $2^n - 1$ holds in case L does not have the empty quotient. We add the case where L has the empty quotient; here the bound is 2^{n-1}. Subword-closure was previously studied by Okhotin [22], but tight upper bounds were not established. Our next theorem solves this problem. For the sake of completeness, we provide all proofs.

Theorem 1 (Closure Operations). *Let $n \geq 2$. Let L be a regular language over an alphabet Σ with $\kappa(L) = n$. Let ${}_{\leq}L$, ${}_{\preceq}L$, ${}_{\sqsubseteq}L$, ${}_{\Subset}L$ be the prefix-, suffix-, factor-, and subword-closure of L, respectively. Then*
1. $\kappa({}_{\leq}L) \leq n$, and the bound is tight if $|\Sigma| \geq 1$;
2. $\kappa({}_{\preceq}L) \leq 2^n - 1$ if L does not have empty quotient and $\kappa({}_{\preceq}L) \leq 2^{n-1}$ otherwise, and both bounds are tight if $|\Sigma| \geq 2$;
3. $\kappa({}_{\sqsubseteq}L) \leq 2^{n-1}$, and the bound is tight if $|\Sigma| \geq 2$;
4. $\kappa({}_{\Subset}L) \leq 2^{n-2} + 1$, and the bound is tight if $|\Sigma| \geq n - 2$.

Proof. 1. Given a language L recognized by dfa $\mathcal{D}$, to get the dfa for its prefix-closure ${}_{\leq}L$, we need only make each non-empty state accepting. Thus $\kappa({}_{\leq}L) \leq n$. For tightness, consider the language $L = \{a^i \mid i \leq n-2\}$. We have $\kappa({}_{\leq}L) = n$.

2. Having a quotient automaton of a language L, we can construct an nfa for its suffix-closure by making each non-empty state initial. The equivalent dfa has at most $2^n - 1$ states if L does not have the empty quotient (the empty set of states cannot be reached), and at most 2^{n-1} states otherwise.

To prove tightness, consider the language L defined by the quotient automaton shown in Fig. 1. Construct an nfa for the suffix-closure of L, by making all states initial. We show that the corresponding subset automaton has $2^n - 1$ reachable and pairwise inequivalent states. We prove reachability by induction on the size

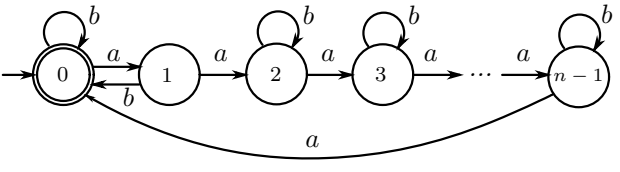


Fig. 1. Quotient automaton of a language L which does not have $\emptyset$

of subsets. The basis, $|S| = n$, holds since $\{0, 1, \ldots, n-1\}$ is the initial state. Assume that each set of size k is reachable, and let S be a set of size $k-1$. If S contains 0 but does not contain 1, then it can be reached from the set $S \cup \{1\}$ of size k by b. If S contains both 0 and 1, then there is an i such that $i \in S$ and $i+1 \notin S$. Then S can be reached from $\{(s-i) \bmod n \mid s \in S\}$ by a^i. The latter set contains 0 and does not contain 1, and so is reachable. If a non-empty S does not contain 0, then it can be reached from $\{s - \min S \mid s \in S\}$, which contains 0, by $a^{\min S}$. To prove inequivalence notice that the word a^{n-i} is accepted by the nfa only from state i for all $i = 0, 1, \ldots, n-1$. It turns out that all the states in the subset automaton are pairwise inequivalent.

Now consider the case where a language has the empty quotient. Let L be defined by the dfa of Fig. 2. Remove state $n-1$ and all transitions going to it, and then construct an nfa as above. The proof of reachability of all non-empty subsets of $\{0, 1, \ldots, n-2\}$ is the same as above. The empty set is reached from $\{0\}$ by b. For inequivalence, the word $(ab)^n$ is accepted only from state 0, and the word $a^{n-1-i}(ab)^n$ is accepted only from state i for $i = 1, 2, \ldots, n-2$.

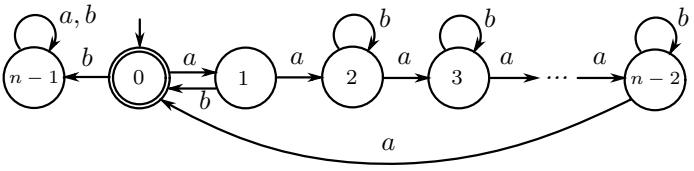


Fig. 2. Quotient automaton of a language L which has $\emptyset$

3. Suppose we have the quotient automaton of a language L. To find an nfa for the factor closure ${}_{\sqsubseteq}L$, we make all non-empty states of the quotient automaton both accepting and initial, and delete the empty state. Hence the bound is 2^{n-1}. The language L defined by the quotient automaton of Fig. 2 meets the bound.

4. To get an ε-nfa for the subword-closure ${}_{\Subset}L$ from the quotient automaton of L, we remove the empty state (if there is no empty state, then ${}_{\Subset}L = \Sigma^*$), and add an ε-transition from state p to state q whenever there is a transition from p to q in the quotient automaton. Since the initial state can reach every non-empty state by ε-transitions, no other subset containing the initial state can be reached. Hence there are at most $2^{n-2}+1$ reachable subsets.

To prove tightness, if $n = 2$, let $\Sigma = \{a, b\}$; then $L = a^*$ meets the bound. If $n \geq 3$, let $\Sigma = \{a_1, \ldots, a_{n-2}\}$, and $L = \bigcup_{a_i \in \Sigma} a_i(\Sigma \setminus \{a_i\})^*$. Thus L consists of all words over Σ in which the first letter occurs exactly once. Let K be the subword-closure of L. Then $K = L \cup \{w \in \Sigma^* \mid \text{at least one letter is missing in } w\}$. For each boolean vector $b = (b_1, b_2, \ldots, b_{n-2})$, define the word $w(b) = w_1 w_2 \cdots w_{n-2}$, in which $w_i = \varepsilon$ if $b_i = 0$ and $w_i = a_i$ if $b_i = 1$. Consider ε, and each word $a_1 w(b)$. All the quotients of K by these $2^{n-2}+1$ words are distinct: For each binary vector b, we have $a_1 a_2 \cdots a_{n-2} \in K_\varepsilon \setminus K_{a_1 w(b)}$. Let b and b' be two different vectors with $b_i = 0$ and $b'_i = 1$. Then we have $a_1 a_2 \cdots a_{i-1} a_{i+1} a_{i+2} \cdots a_{n-2} \in K_{a_1 w(b)} \setminus K_{a_1 w(b')}$. Thus all quotients are distinct, and so $\kappa(K) \geq 2^{n-2}+1$. □

4 Basic Operations on Closed Languages

Now we study the quotient complexity of operations on closed languages. For regular languages, the following bounds are known [20,21,28]: mn for boolean operations, $m2^n - 2^{n-1}$ for product, $2^{n-1}+2^{n-2}$ for star, and 2^n for reversal. The bounds for closed languages are smaller in most cases. The bounds for boolean operations and reversal follow from the results on ideal languages [7].

Theorem 2 (Boolean Operations). *Let K and L be prefix-closed (or factor-closed, or subword-closed) languages with $\kappa(K) = m$ and $\kappa(L) = n$. Then*
1. $\kappa(K \cap L) \leq mn - (m + n - 2)$,
2. $\kappa(K \cup L), \kappa(K \oplus L) \leq mn$,
3. $\kappa(K \setminus L) \leq mn - (n - 1)$.
For suffix-closed languages, $\kappa(K \circ L) \leq mn$. All bounds are tight if $|\Sigma| \geq 2$ except for the union and difference of suffix-closed languages where $|\Sigma| \geq 4$ is required.

Proof. The complement of a prefix-closed (suffix-, factor-, or subword-closed) language is a right (respectively, left, two-sided, all-sided) ideal. We get all the results using De Morgan's laws and the results from [7]. □

Remark 1. If L is prefix-closed, then either $L = \Sigma^*$ or L has the empty quotient. Moreover, each quotient of L is either accepting or empty.

Remark 2. For a suffix-closed language L, if v is a suffix of w then $L_w \subseteq L_v$. In particular, $L_w \subseteq L_\varepsilon = L$ for each word w in Σ^*.

Theorem 3 (Product). *Let $m, n \geq 2$. Let K and L be closed languages with $\kappa(K) = m$, $\kappa(L) = n$, and let k be the number of accepting quotients of K.*
1. If K and L are prefix-closed, then $\kappa(KL) \leq (m+1) \cdot 2^{n-2}$.
2. If K and L are suffix-closed, then $\kappa(KL) \leq (m-k)n + k$.
3. If K and L are both factor- or both subword-closed, then $\kappa(KL) \leq m+n-1$.
The first two bounds are tight if $|\Sigma| \geq 3$, and the third bound is tight if $|\Sigma| \geq 2$.
If $\kappa(K) = 1$ or $\kappa(L) = 1$, then $\kappa(KL) = 1$.

Proof. If $m = 1$, then $K = \emptyset$ or $K = \Sigma^*$; so $KL = \emptyset$ or $KL = \Sigma^*$, for if $L \neq \emptyset$, then $\varepsilon \in L$. Thus $\kappa(KL) = 1$. The case $n = 1$ is similar. Now let $m, n \geq 2$.

1. If K and L are prefix-closed, then $\varepsilon \in K$ and by Remark 1 both languages have the empty quotient. The quotient $(KL)_w$ is given by Equation (2). If K_w is accepting, then L is always in the union, and there are 2^{n-2} non-empty subsets of non-empty quotients of L that can be added. Since there are $m-1$ accepting quotients of K, there are $(m-1)2^{n-2}$ such quotients of KL. If K_w is rejecting, then 2^{n-1} subsets of non-empty quotients of L can be added.

For tightness, consider prefix-closed languages K and L defined by the quotient automata of Fig. 3 (if $n = 2$, then $L = \{a, c\}^*$). Construct an ε-nfa for

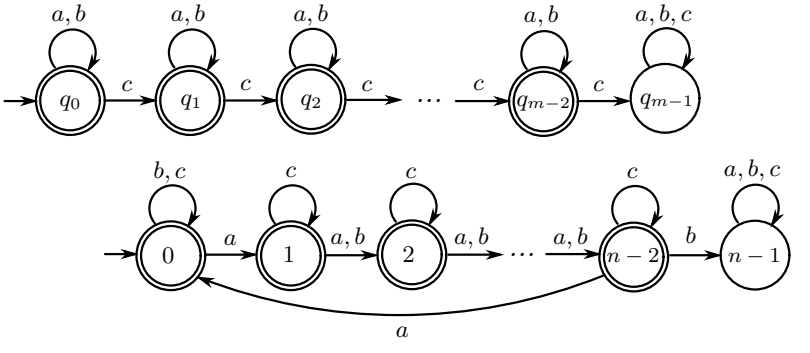


Fig. 3. Quotient automata of prefix-closed languages K and L

the language KL from these quotient automata by adding an ε-transition from states $q_0, q_1, \ldots, q_{m-2}$ to state 0. The initial state of the nfa is q_0, and the accepting states are $0, 1, \ldots, n-2$. We show that there are $(m+1) \cdot 2^{n-2}$ reachable and pairwise inequivalent states in the corresponding subset automaton.

State $\{q_0, 0\}$ is the initial state, and each state $\{q_0, 0, i_1, i_2, \ldots, i_k\}$, where $1 \le i_1 < i_2 < \cdots < i_k \le n-2$, is reached from state $\{q_0, 0, i_2 - i_1, \ldots, i_k - i_1\}$ by ab^{i_1-1}. For each subset S of $\{0, 1, \ldots, n-2\}$ containing 0, each state $\{q_i\} \cup S$ with $1 \le i \le m-1$ is reached from $\{q_0\} \cup S$ by c^i. If a non-empty S does not contain 0, then $\{q_{m-1}\} \cup S$ is reached from $\{q_{m-1}\} \cup \{s - \min S \mid s \in S\}$, which contains 0, by $a^{\min S}$. State $\{q_{m-1}, n-1\}$ is reached from $\{q_{m-1}, n-2\}$ by b.

To prove inequivalence, notice that the word b^n is accepted by the quotient automaton for L only from state 0, and the word $a^{n-1-i}b^n$ only from state i $(1 \le i \le n-2)$. It turns out that two different states $\{q_{m-1}\} \cup S$ and $\{q_{m-1}\} \cup T$ are inequivalent. It follows that states $\{q_i\} \cup S$ and $\{q_i\} \cup T$ are inequivalent as well. States $\{q_i\} \cup S$ and $\{q_j\} \cup T$ with $i < j$ can be distinguished by $c^{m-1-j}b^n a b^n$.

2. If K and L are suffix-closed, then, by Remark 2, for each word w in Σ^* and u, v in Σ^+, we have $(KL)_w = K_w L \cup K^\varepsilon L_w \cup (\bigcup_{w=uv} K_u^\varepsilon L_v) = K_w L \cup L_x$ for some suffix x of w. If K_w is a rejecting quotient, there are at most $(m-k)n$ such quotients. If K_w is accepting, then $\varepsilon \in K_w$, and since $L_x \subseteq L_\varepsilon = L \subseteq K_w L$, we have $(KL)_w = K_w L$. There are at most k such quotients. Therefore there are at most $(m-k)n + k$ quotients in total.

To prove tightness, let K and L be ternary suffix-closed languages defined by the quotient automata of Fig. 4. Consider the words $\varepsilon = a^0 b^0$, and $a^i b^j$ with

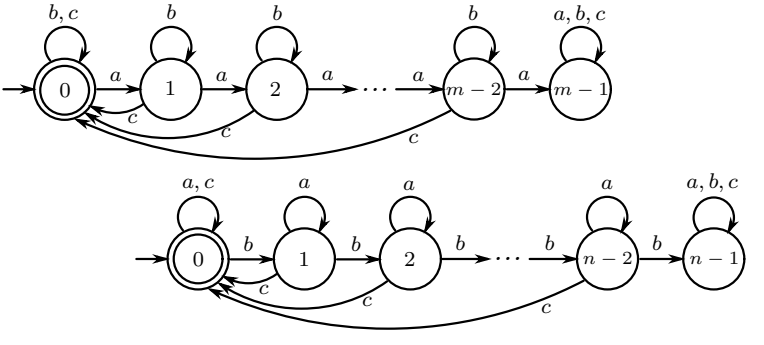


Fig. 4. Quotient automata of suffix-closed languages K and L

$1 \le i \le m-1$ and $0 \le j \le n-1$. Let us show that all the quotients of KL by these words are distinct. Let $(i, j) \ne (k, \ell)$, and let $x = a^i b^j$ and $y = a^k b^\ell$. If $i < k$, take $z = a^{m-1-k} b^n c$. Then xz is in KL, while yz is not, and so $z \in (KL)_x \setminus (KL)_y$. If $i = k$ and $j < \ell$, take $z = a^m b^{n-1-\ell} c$. We again have $z \in (KL)_x \setminus (KL)_y$.

Notice that, if the quotients K_{a^i} with $0 \le i \le k-1$ are accepting, then the resulting product has quotient complexity $(m-k)n + k$.

3. It suffices to derive the bound for factor-closed languages, since every subword-closed language is also factor-closed. Since factor-closed languages are suffix-closed, $\kappa(KL) \le (m-k)n + k$. The language K has at most one rejecting quotient, because it is prefix-closed. Thus, $k = m-1$ and $\kappa(KL) \le m + n - 1$.

For tightness, let $K = \{w \in \{a,b\}^* \mid a^{m-1}$ is not a subword of $w\}$ and $L = \{w \in \{a,b\}^* \mid b^{n-1}$ is not a subword of $w\}$. The languages are subword-closed and $\kappa(K) = m$ and $\kappa(L) = n$. Consider the word $w = a^{m-1}b^{n-1}$. This word is not in the product KL. However, removing any non-empty subword from w results in a word in KL. Therefore, $\kappa(KL) \geq m + n - 1$. □

Theorem 4 (Star). *Let $n \geq 2$. Let L be a closed language with $\kappa(L) = n$.*
1. If L is prefix-closed, then $\kappa(L^*) \leq 2^{n-2} + 1$.
2. If L is suffix-closed, then $\kappa(L^*) = n$ if $L = L^*$, and $\kappa(L^*) \leq n - 1$ if $L \neq L^*$.
3. If L is factor- or subword-closed, then $\kappa(L^*) \leq 2$.
The first bound is tight if $|\Sigma| \geq 3$, and all the other bounds are tight if $|\Sigma| \geq 2$. If $\kappa(L) = 1$, then $\kappa(L^*) \leq 2$.

Proof. 1. For every non-empty word w, the quotient $(L^*)_w$ is given by Equation (3). If L is prefix-closed, then so is L^* and $(L^*)_w$. Thus, if $(L^*)_w$ is non-empty, then it contains ε. Hence $(L^*)_w \supseteq L^* \supseteq L$. Since $\emptyset$ and L are always contained in every non-empty quotient of L^*, there are at most 2^{n-2} non-empty quotients of L^*. Since there is at most one empty quotient, there are at most $2^{n-2} + 1$ quotients in total. The quotient $(L^*)_\varepsilon$ has already been counted, since L is closed and $\varepsilon \in L$ implies $(L^*)_\varepsilon = LL^*$, which has the form of Equation (3).

If $n = 1$ and $n = 2$, the bound 2 is met by $L = \emptyset$ and $L = \varepsilon$, respectively. Now let $n \geq 3$ and let L be the prefix-closed language defined by the dfa of Fig. 5; transitions not depicted in the figure go to state $n - 1$. Construct an ε-nfa for

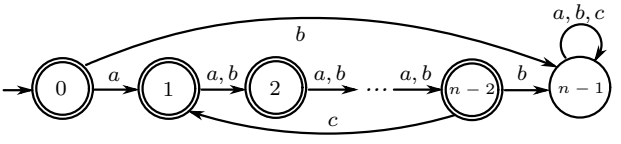


Fig. 5. Quotient automaton of prefix-closed language L

L^* by removing state $n - 1$ and adding an ε-transition from all the remaining states to the initial state. Let us show that $2^{n-2} + 1$ states are reachable and pairwise inequivalent in the corresponding subset automaton.

We first prove that each subset of $\{0, 1, \ldots, n-2\}$ containing state 0 is reachable. The proof is by induction on the size of the subsets. The basis, $|S| = 1$, holds since $\{0\}$ is the initial state of the subset automaton. Assume that each set of size k containing 0 is reachable, and let $S = \{0, i_1, i_2, \ldots, i_k\}$, where $0 < i_1 < i_2 < \cdots < i_k \leq n - 2$, be a set of size $k + 1$. Then S is reached from the set $\{0, i_2 - i_1, \ldots, i_k - i_1\}$ of size k by ab^{i_1-1}. Since the latter set is reachable by the induction hypothesis, the set S is reachable as well. The empty set can be reached from $\{0\}$ by b, and we have $2^{n-2} + 1$ reachable states. To prove inequivalence notice that b^{n-3} is accepted by the nfa only from state 1, and each word $b^{n-2-i}cb^{n-3}$ $(2 \leq i \leq n - 2)$, only from state i.

2. For a non-empty suffix-closed language L, the quotient $(L^*)_\varepsilon$ is LL^*, which is of the same form as the quotients by a non-empty word w in Equation (3),

$(L^*)_w = (L_w \cup L_{v_1} \cup \cdots \cup L_{v_k})L^*$, where the v_i are suffixes of w, and v_k is the shortest. By Remark 2, $(L^*)_w = L_{v_k}L^*$. There are at most n such quotients. If $L \neq L^*$ for a non-empty suffix-closed language L, then there must be two words x, y in L such that $xy \notin L$. Hence $y \in L_\varepsilon \setminus L_x$, and so $L_\varepsilon \neq L_x$. However, since $\varepsilon \in L_x$ and L^* is suffix-closed, we have $(L^*)_\varepsilon = L^* \subseteq L_xL^* \subseteq (L^*)_x \subseteq (L^*)_\varepsilon$, and so $(L^*)_\varepsilon = (L^*)_x$. It turns out that $\kappa(L^*) \leq n-1$.

For $n = 1$, $L = \emptyset$ and for $n = 2$, $L = \varepsilon$ meet the bound 2. Let $n \geq 3$. If $L = (a \cup ba^{n-2})^*$, then L is suffix-closed, $\kappa(L) = n$, and $L^* = L$. If $L = \varepsilon \cup \bigcup_{i=0}^{n-3} a^ib$, then L is suffix-closed, $\kappa(L) = n$, $L^* = (\bigcup_{i=0}^{n-3} a^ib)^*$, and $\kappa(L^*) = n-1$.

3. If each letter in Σ appears in some word of a factor-closed language L, then $L^* = \Sigma^*$ and $\kappa(L^*) = 1$. Otherwise, $\kappa(L^*) = 2$. The bound is met by subword-closed language $L = \{w \in \{a,b\}^* \mid w = a^i \text{ and } 0 \leq i \leq n-2\}$. □

Since the operation of reversal commutes with complementation, the next theorem follows from the results on ideal languages [7].

Theorem 5 (Reversal). *Let $n \geq 2$. Let L be a closed language with $\kappa(L) = n$.*
1. If L is prefix-closed, then $\kappa(L^R) \leq 2^{n-1}$. The bound is tight if $|\Sigma| \geq 2$.
2. If L is suffix-closed, then $\kappa(L^R) \leq 2^{n-1}+1$. The bound is tight if $|\Sigma| \geq 3$.
3. If L is factor-closed, then $\kappa(L^R) \leq 2^{n-2}+1$. The bound is tight if $|\Sigma| \geq 3$.
4. If L is subword-closed, then $\kappa(L^R) \leq 2^{n-2}+1$. The bound is tight if $|\Sigma| \geq 2n$.
If $\kappa(L) = 1$, then $\kappa(L^R) = 1$. □

Unary Languages: Unary languages have special properties because the product of unary languages is commutative. The classes of prefix-closed, suffix-closed, factor-closed, and subword-closed unary languages all coincide. If a unary closed language L is finite, then either it is empty and so $\kappa(L) = 1$, or has the form $\{a^i \mid i \leq n-2\}$ and then $\kappa(L) = n$. If L is infinite, then $L = a^*$ and $\kappa(L) = 1$. The bounds for unary languages are in Tables 1 and 2 on page 94.

5 Kuratowski Algebras Generated by Closed Regular Languages

A theorem of Kuratowski [18] states that, given a topological space, at most 14 distinct sets can be produced by repeatedly applying the operations of closure and complement to a given set. A closure operation on a set S is an operation $\Box : 2^S \to 2^S$ satisfying the following conditions for any subsets X, Y of S: (1) $X \subseteq X^\Box$, (2) $X \subseteq Y$ implies $X^\Box \subseteq Y^\Box$, (3) $X^{\Box\Box} \subseteq X^\Box$.

Kuratowski's theorem was studied in the setting of formal languages in [6]. Positive closure and Kleene closure (star) are both closure operations. It then follows that at most 10 distinct languages can be produced by repeatedly applying the operations of positive closure and complement to a given language, and at most 14 distinct languages can be produced with Kleene closure instead of positive closure. We consider here the case where the given language is closed and regular, and give upper bounds on the quotient complexity of the resulting languages. We denote the complement of a language L by L^-, the positive closure of the complement of L by L^{-+}, etc.

We begin with positive closure. Let L be a $\trianglelefteq$-closed language not equal to Σ^*. Then L^- is an ideal, and $L^{-+} = L^-$. In addition, L^+ is also $\trianglelefteq$-closed, so $L^{+-+} = L^{+-}$. Hence there are at most 4 distinct languages that can be produced with positive closure and complementation.

Theorem 6. *The worst-case complexities in the 4-element algebra generated by a closed language L with $\kappa(L) = n$ under positive closure and complement are as follows: $\kappa(L) = \kappa(L^-) = n$, $\kappa(L^+) = \kappa(L^{+-}) = f(n)$, where $f(n)$ is $2^{n-2}+1$ for prefix-closed languages, $n-1$ for suffix-closed languages, and 2 for factor- and subword-closed languages. There exist closed languages that meet these bounds.*

Proof. Since $L^+ = L^*$ for a non-empty closed language we have $\kappa(L^+) = \kappa(L^*)$, and the upper bounds $f(n)$ follow from our results on the quotient complexity of the star operation; in the case of suffix-closed languages, to get a 4-element algebra we need $L \neq L^*$. All the languages that we have used in Theorem 4 to prove tightness can be used as examples meeting the bound $f(n)$. □

The case of Kleene closure is similar. Let L be a non-empty $\trianglelefteq$-closed language that is not equal to Σ^*. Then the language L^- is an ideal and L^- does not contain ε. Thus $L^{-*} = L^- \cup \varepsilon$ and $L^{-*-} = L \setminus \varepsilon$, which gives at most four languages thus far. Now $L^* = (L \setminus \varepsilon)^*$, and the language L^* is also $\trianglelefteq$-closed. By the previous reasoning, we have at most four additional languages, giving a total of eight languages as the upper bound. The 8-element algebras are of the form $(L, L^-, L^{-*} = L^- \cup \varepsilon, L^{-*-} = L \setminus \varepsilon, L^*, L^{*-}, L^{*-*} = L^{*-} \cup \varepsilon, L^{*-*-} = L^* \setminus \varepsilon)$.

Theorem 7. *The worst-case quotient complexities in the 8-element algebra generated by a closed language L with $\kappa(L) = n$ under star and complement are as follows: $\kappa(L) = \kappa(L^-) = n$, $\kappa(L^*) = \kappa(L^{*-}) = f(n)$, $\kappa(L^{*-*}) = \kappa(L^{*-*-}) = f(n)+1$, $\kappa(L^{-*}) = \kappa(L^{-*-}) = n+1$, where $f(n)$ is $2^{n-2}+1$ for prefix-closed languages, $n-1$ for suffix-closed languages, and 2 for factor-and subword-closed languages. Moreover, there exist closed languages that meet these bounds.*

Proof. Since $L^{-*-} = L \setminus \varepsilon$ and $L^{*-*-} = L^* \setminus \varepsilon$ we have $\kappa(L^{-*-}) \leq n+1$ and $\kappa(L^{*-*-}) \leq f(n)+1$. In the case of suffix-closed languages, since L must be distinct from L^*, we have $f(n) = n-1$ by Theorem 4.

1. Let L be the prefix-closed language defined by the quotient automaton in Fig. 5 on page 91; then L meets the upper bound on star. Add a loop with a new letter d in each state and denote the resulting language by K. Then K is a prefix-closed language with $\kappa(K) = n$ and $\kappa(K \setminus \varepsilon) = n+1$. Next we have $\kappa(K^*) = \kappa(L^*) = 2^{n-2}+1$ and $\kappa(K^* \setminus \varepsilon) = 2^{n-2}+2$.

2. Let $L = b^* \cup \bigcup_{i=1}^{n-3} b^* a^i b$. Then L is a suffix-closed language with $\kappa(L) = n$ and $\kappa(L \setminus \varepsilon) = n+1$. Next $\kappa(L^*) = n-1$ and $\kappa(L^* \setminus \varepsilon) = n$.

3. Let $L = \{w \in \{a,b,c\}^* \mid w = b^* a^i \text{ and } 0 \leq i \leq n-2\}$. Then L is a subword-closed language with $\kappa(L) = n$ and $\kappa(L \setminus \varepsilon) = n+1$. Next $L^* = \{a,b\}^*$, and so $\kappa(L^*) = 2$ and $\kappa(L^* \setminus \varepsilon) = 3$. □

Table 1. Bounds on quotient complexity of boolean operations

	$K \cup L$	$\lvert\Sigma\rvert$	$K \cap L$	$\lvert\Sigma\rvert$	$K \setminus L$	$\lvert\Sigma\rvert$	$K \oplus L$	$\lvert\Sigma\rvert$
unary closed	$\max(m,n)$	1	$\min(m,n)$	1	m	1	$\max(m,n)$	1
prefix-, factor-, subword-closed	mn	2	$mn-m-n+2$	2	$mn-n+1$	2	mn	2
suffix-closed	mn	4	mn	2	mn	4	mn	2
regular	mn	2	mn	2	mn	2	mn	2

Table 2. Bounds on quotient complexity of closure, product, star and reversal

	$\trianglelefteq L$	$\lvert\Sigma\rvert$	KL	$\lvert\Sigma\rvert$	K^*	$\lvert\Sigma\rvert$	K^R	$\lvert\Sigma\rvert$
unary closed	n	1	$m+n-2$	1	2	1	n	1
prefix–closed	n	1	$(m+1)2^{n-2}$	3	$2^{n-2}+1$	3	2^{n-1}	2
suffix-closed	2^n-1	2	$(m-k)n+k$	3	n	2	$2^{n-1}+1$	3
factor-closed	2^{n-1}	2	$m+n-1$	2	2	2	$2^{n-2}+1$	3
subword-closed	$2^{n-2}+1$	$n-2$	$m+n-1$	2	2	2	$2^{n-2}+1$	$2n$
regular	–	–	$m2^n-k2^{n-1}$	2	$2^{n-1}+2^{n-k-1}$	2	2^n	2

6 Conclusions

Tables 1 and 2 summarize our complexity results. The complexities for regular languages are from [16,20,21,28], except those for difference and symmetric difference, which are from [5]. The bounds for boolean operations and reversal of closed languages are direct consequences of the results in [7]. In Table 2, k is the number of accepting quotients of K; the results for prefix-, suffix-, and factor-closure are from [17]. The tables also show the size of the alphabet of the witness languages. In all cases when the size of the alphabet is more than two, we do not know whether the bounds are tight for a smaller alphabet.

References

1. Ang, T., Brzozowski, J.: Languages convex with respect to binary relations, and their closure properties. Acta Cybernet 19, 445–464 (2009)
2. Avgustinovich, S.V., Frid, A.E.: A unique decomposition theorem for factorial languages. Internat. J. Algebra Comput. 15, 149–160 (2005)
3. Bassino, F., Giambruno, L., Nicaud, C.: Complexity of operations on cofinite languages. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 222–233. Springer, Heidelberg (2010)
4. Brzozowski, J.: Derivatives of regular expressions. J. ACM 11, 481–494 (1964)
5. Brzozowski, J.: Quotient complexity of regular languages. In: Dassow, J., Pighizzini, G., Truthe, B. (eds.) 11th International Workshop on Descriptional Complexity of Formal Systems, DCFS 2009, pp. 25–42. Otto-von-Guericke-Universität, Magdeburg (2009), http://arxiv.org/abs/0907.4547

6. Brzozowski, J., Grant, E., Shallit, J.: Closures in formal languages and Kuratowski's theorem. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 125–144. Springer, Heidelberg (2009)
7. Brzozowski, J., Jirásková, G., Li, B.: Quotient complexity of ideal languages. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 208–221. Springer, Heidelberg (2010) Full paper, http://arxiv.org/abs/0908.2083
8. Brzozowski, J., Santean, N.: Predictable semiautomata. Theoret. Comput. Sci. 410, 3236–3249 (2009)
9. Brzozowski, J., Shallit, J., Xu, Z.: Decision procedures for convex languages. In: Dediu, A., Ionescu, A., Martin-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 247–258. Springer, Heidelberg (2009)
10. Galil, Z., Simon, J.: A note on multiple-entry finite automata. J. Comput. System Sci. 12, 350–351 (1976)
11. Gill, A., Kou, L.T.: Multiple-entry finite automata. J. Comput. System Sci. 9, 1–19 (1974)
12. Haines, L.H.: On free monoids partially ordered by embedding. J. Combin. Theory 6, 94–98 (1969)
13. Han, Y.-S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. Theoret. Comput. Sci. 410, 2537–2548 (2009)
14. Han, Y.-S., Salomaa, K., Wood, D.: Operational state complexity of prefix-free regular languages. In: Automata, Formal Languages, and Related Topics, pp. 99–115. University of Szeged, Hungary (2009)
15. Holzer, M., Salomaa, K., Yu, S.: On the state complexity of k-entry deterministic finite automata. J. Autom. Lang. Comb. 6, 453–466 (2001)
16. Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation. Internat. J. Found. Comput. Sci. 16, 511–529 (2005)
17. Kao, J.-Y., Rampersad, N., Shallit, J.: On NFAs where all states are final, initial, or both. Theoret. Comput. Sci. 410, 5010–5021 (2009)
18. Kuratowski, C.: Sur l'opération $\overline{A}$ de l'analysis situs. Fund. Math. 3, 182–199 (1922)
19. de Luca, A., Varricchio, S.: Some combinatorial properties of factorial languages. In: Capocelli, R. (ed.) Sequences, pp. 258–266. Springer, Heidelberg (1990)
20. Maslov, A.N.: Estimates of the number of states of finite automata. Dokl. Akad. Nauk SSSR 194, 1266–1268 (1970) (in Russian); English translation: Soviet Math. Dokl. 11, 1373–1375 (1970)
21. Mirkin, B.G.: On dual automata. Kibernetika (Kiev) 2, 7–10 (1966) (in Russian); English translation: Cybernetics 2, 6–9 (1966)
22. Okhotin, A.: On the state complexity of scattered substrings and superstrings. Turku Centre for Computer Science Technical Report No. 849 (2007)
23. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function. Internat. J. Found. Comput. Sci. 13, 145–159 (2002)
24. Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages. Theoret. Comput. Sci. 320, 315–329 (2004)
25. Thierrin, G.: Convex languages. In: Nivat, M. (ed.) Automata, Languages and Programming, pp. 481–492. North-Holland, Amsterdam (1973)
26. Veloso, P.A.S., Gill, A.: Some remarks on multiple-entry finite automata. J. Comput. System Sci. 18, 304–306 (1979)
27. Yu., S.: State complexity of regular languages. J. Autom. Lang. Comb. 6, 221–234 (2001)
28. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoret. Comput. Sci. 125, 315–328 (1994)

Right-Sequential Functions on Infinite Words

Olivier Carton

LIAFA, Université Paris Diderot & CNRS
http://www.liafa.jussieu.fr/~carton

Abstract. In this paper, we introduce a notion of a right-sequential function on infinite words. The main result is that any rational function is the composition of a right-sequential function and a left-sequential function. This extends a classical result of Elgot and Mezei on finite words to infinite words. We also show that our class of right-sequential functions includes the normalization of real numbers in some base and the truth value of linear temporal logic. Finally, we apply the decomposition theorem to show that automatic sequences are preserved by rational letter-to-letter functions.

1 Introduction

A classical result, due to Elgot and Mezei [7], on rational functions of finite words states that any such function can be described as the composition of a left-sequential function and a right-sequential one. The aim of this paper is to establish the same result when dealing with rational functions of infinite words. For this, we introduce a natural notion of a right-sequential function for infinite words.

Rational functions are functions between free monoids realized by finite transducers. Transducers are finite nondeterministic automata with edges labeled by pairs of finite words (an input and an output label). They are very useful in various areas like coding [8], computer arithmetic [9], language processing (see for instance [12] and [10]), and in program analysis [6]. Transducers that have a deterministic input automaton are called left-sequential transducers [14] since they deterministically read their input from left to right. Rational functions that can be realized by left-sequential transducers are called left-sequential functions. They play an important role since they allow sequential encoding. We refer the reader to [3] and [13] for complete introductions to transducers.

In the case of finite words, the result of Elgot and Mezei [7] states that any rational function is the composition of a left-sequential function and a right-sequential function. In that case, right-sequential functions are those realized by transducers with a co-deterministic input automaton, that is, an automaton that becomes deterministic if all its transitions are reversed. This result is important since it has both theoretical and practical applications. From a theoretical point of view, it allows us to show that some words or languages are preserved by rational functions by just considering sequential functions which are usually

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 96–106, 2010.

much easier to handle. From a practical point of view, it allows us to implement a rational transduction in a two-stage process.

In this paper, we extend the result of Elgot and Mezei to infinite words. The main problem is to come up with a right notion of a right-sequential transducer. The intuitive idea of such a machine is not clear because it should read each infinite word from right to left, starting at the infinite tail. The notion of a left-sequential transducer is the same for finite and for infinite words, since determinism makes sense for both finite and infinite words. Co-determinism is, however, not sufficient to define a suitable notion of a right-sequential transducer on infinite words (see Example 3). Our definition of a right-sequential transducer is based on prophetic automata introduced in [4]. These automata are defined by a global property rather than a local one such as the co-determinism.

To show that our notion of a right-sequential function is relevant, we give examples of such functions taken from different fields. Our first example comes from computer arithmetic. We show that normalization of real numbers is a right-sequential function. This extends the classical result on integers represented in some base by finite words [9]. Our second example comes from the field of verification. We show that the function that maps a model of linear temporal logic to its word of truth values is also a right-sequential function.

Finally, we apply the main theorem to automatic sequences. It allows us to prove that automatic sequences are preserved by rational letter-to-letter functions. This extends a classical result that these sequences are preserved by left-sequential functions [1].

The paper is organized as follows. Section 2 is devoted to basic notions of automata and transducers. Left- and right-sequential transducers are defined in Section 3. Examples of right-sequential functions are given in Section 4. The main theorem of decomposition of rational function is stated in Section 5. The application to automatic sequences is given in Section 6.

2 Automata and Transducers

In the sequel, A and B denote finite alphabets. Sets of finite and infinite words over A are respectively A^* and A^ω. The empty word is denoted by ε.

A *Büchi automaton* (see [11, p. 25] for a complete introduction) over A is an automaton (Q, A, E, I, F) where Q is a finite set of *states*, $E \subseteq Q \times A \times Q$ is a finite set of *transitions*, $I \subseteq Q$ is the set of *initial* states and $F \subseteq Q$ is the set of final states. A transition (p, a, q) from p to q is denoted $p \xrightarrow{a} q$. An *infinite path* γ in the automaton is an infinite sequence

$$q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \cdots$$

of consecutive transitions. Its *label* is the infinite concatenation $x = a_0a_1a_2\cdots$ of the labels of its transitions. The path is *initial* if its first state q_0 is initial and it is *final* if it visits a final state infinitely often. A path is *accepting* if it is both initial and final. An infinite word is *accepted* if it is the label of an accepting path.

A *Büchi transducer* is a Büchi automaton over $A^* \times B^*$. The label of each transition is then a pair (u, v) of words that is merely written $u|v$. A transition (p, u, v, q) is then denoted by $p \xrightarrow{u|v} q$. The words u and v are called the *input* and *output label* of the transition. The *label* of an infinite path

$$q_0 \xrightarrow{u_0|v_0} q_1 \xrightarrow{u_1|v_1} q_2 \xrightarrow{u_2|v_2} q_3 \cdots$$

is the pair (x, y) where the *input label* x is $u_0u_1u_2\cdots$ and the *output label* y is $v_0v_1v_2\cdots$. Since the labels of the transitions might be empty, the labels of a path are either finite or infinite. A path is accepting if it is initial and final and if both its input and output labels are infinite. The relation *realized* by a Büchi transducer is the set of pairs (x, y) labelling an accepting path. A partial function f from A^ω to B^ω is realized by a transducer $\mathcal{T}$ if its graph $\{(x, f(x)) \mid x \in \mathrm{dom}(f)\}$ is realized by $\mathcal{T}$. A relation or a partial function is said to be *rational* if it is realized by some Büchi transducer.

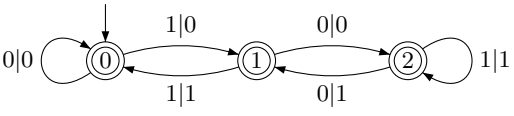


Fig. 1. Transducer for division by 3 in base 2

Example 1. Let A and B be the alphabet $\{0, 1\}$. The transducer depicted in Fig. 1 realizes division by 3. If the input word is the infinite binary representation of a real number α, the output word is the binary representation of $\alpha/3$.

A *transducer with an initial output function* is a transducer with an additional function ι from the set I of initial states to the set B^* of finite words over B. The purpose of this function is to prefix the output of each accepting path with some finite word $\iota(q_0)$, called the *initial output*, depending on the first state q_0. The output label of an accepting path

$$q_0 \xrightarrow{u_0|v_0} q_1 \xrightarrow{u_1|v_1} q_2 \xrightarrow{u_2|v_2} q_3 \cdots$$

is the word $\iota(q_0)v_0v_1v_2\cdots$. In the figures, the initial output $\iota(q)$ is written as a label on the small incoming arrow marking the initial state q. Nothing is written when $\iota(q) = \varepsilon$.

Initial output functions are, in general, not needed. It is always possible to remove the initial output function by adding a new initial state and some transitions. In this paper, we consider a special class of transducers, namely right-sequential transducers, for which initial output functions turn out to be useful.

A transducer is said to be *real-time* if the input label of each transition is a single letter. The transducer of Example 1 is real-time. The *input automaton* of a real-time transducer is the automaton obtained by ignoring the output label of each transition. Each transition $p \xrightarrow{a|v} q$ of the transducer is replaced in the input automaton by the transition $p \xrightarrow{a} q$. A transducer is said to be

letter-to-letter if both the input and the output labels of each transition are letters. The transducer of Example 1 is actually letter-to-letter. By extension, a partial function is said to be *letter-to-letter* if it is realized by a letter-to-letter transducer.

3 Sequential Transducers

We first recall the usual definition of a left-sequential transducer, and next we define the notion of a right-sequential transducer.

As usual, an automaton is *deterministic* if it has a single initial state and for any pair $(p, a) \in Q \times A$, there is at most one transition $p \xrightarrow{a} q$. A transducer is *left-sequential* if it is real-time and its input automaton is deterministic. A partial function $f : A^\omega \to B^\omega$ is *left-sequential* if it is realized by a left-sequential transducer. The transducer given in Example 1 is left-sequential.

We now introduce the key definition that allows us to define right-sequential transducers and functions. A Büchi automaton over a finite alphabet A is called *prophetic* if any infinite word over A is the label of exactly one final path. Let us rephrase this definition. For any state q, let X_q be the set of words accepted by the automaton $\mathcal{A}_q$ obtained by replacing the set I of initial states in $\mathcal{A}$ by the singleton $\{q\}$. The automaton $\mathcal{A}$ is prophetic if the family $(X_q)_{q\in Q}$ is a partition of the set A^ω of all infinite words over A. This means that $A^\omega = \bigcup_{q\in Q} X_q$ and $X_p \cap X_q = \emptyset$ whenever $p \neq q$. Here we follow the terminology of [11, p. 122]. These automata were introduced as *complete and unambiguous Büchi automata* in [4], where it is shown that any rational set of infinite words is accepted by such an automaton. To illustrate this definition, we give below an example of a prophetic automaton.

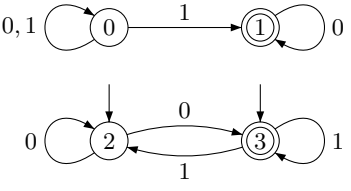


Fig. 2. A prophetic automaton

Example 2. Consider the automaton $\mathcal{A}$ depicted in Fig. 2. The sets X_0, X_1, X_2 and X_3 are respectively equal to $(0{+}1)^*10^\omega$, 0^ω, $0(0^*1)^\omega$ and $1(0^*1)^\omega$. Since these four sets do form a partition of $(0 + 1)^\omega$, the automaton $\mathcal{A}$ is prophetic. Since the initial states are states 2 and 3, this automaton accepts $X_2 + X_3 = (0^*1)^\omega$.

We recall here a property of prophetic automata that is proved in [4]. Any prophetic automaton is co-deterministic and complete. This means that for any pair (q, a) in $Q \times A$, there is exactly one state p such that $p \xrightarrow{a} q$ is a transition. This property is, however, not sufficient to insure that the automaton is prophetic. This is shown by the following example.

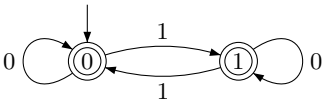


Fig. 3. A non-prophetic automaton

Example 3. The automaton depicted in Fig. 3 is co-deterministic and complete but it is not prophetic. The word 0^ω is the label of the two final paths $(0 \xrightarrow{0} 0)^\omega$ and $(1 \xrightarrow{0} 1)^\omega$.

Prophetic automata capture the idea of reading infinite words from right to left, that is, from the infinite tail to the first letter. The unique final path labeled by an infinite word $x' = ax$ is always made of a transition $p \xrightarrow{a} q$ preceding the unique final path labeled by x.

We now recall a second property of prophetic automata. The final path labeled by a periodic word $x = u^\omega$ is always periodic (see [4, Proposition 8]).

A transducer is called *right-sequential* if it is real-time and its input automaton is prophetic. A partial function from A^ω to B^ω is called *purely right-sequential* if it is realized by a right-sequential transducer. Is is called *right-sequential* if it is realized by a right-sequential transducer with an initial output function. The next example shows that the former class is strictly contained in the latter one. We follow the terminology of [13] rather than the seminal one of Schützenberger. More emphasis is put on sequential functions (called subsequential in Schützenberger's terminology) which form the most natural class as shown by the following example.

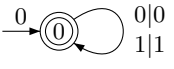


Fig. 4. A right-sequential transducer

Example 4. The transducer depicted in Fig. 4 realizes the function f from $\{0,1\}^\omega$ to $\{0,1\}^\omega$ which maps each infinite word x to $0x$. The leading 0 is added by the initial output function ι defined by $\iota(0) = 0$. Since the input automaton of this transducer is obviously prophetic, the function f is right-sequential. This function is not purely right-sequential since the image of a periodic word must be periodic. This is due to the fact that the final path labeled by a periodic word in a prophetic automaton is periodic.

Example 5. The transducer depicted in Fig. 5 realizes the shift function from $\{0,1\}^\omega$ to $\{0,1\}^\omega$ which maps any infinite word $a_0a_1a_2\cdots$ to $a_1a_2a_3\cdots$.

4 Examples of Right-Sequential Functions

The purpose of this section is to show that some natural functions are right-sequential.

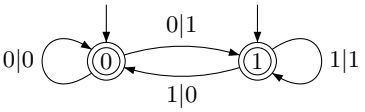


Fig. 5. A purely right-sequential transducer

4.1 Normalization of Real Numbers

It is well known [9] that, when integers are represented in a base, multiplication and division by a fixed integer are rational functions. Furthermore, multiplication is right-sequential whereas division is left-sequential. This captures the fact that multiplication must be algorithmically performed from right to left, whereas division must be performed from left to right. The left-sequential transducer that realizes division for integers also realizes division for real numbers. The transducer given in Example 1 for division by 3 works for integers as well as for real numbers. In this section, we show that multiplication of real numbers by a fixed integer is a right-sequential function. This result follows from a more general result about normalization.

Let $b \geq 2$ be a fixed base. A *b-representation* of a real number α in $[0, 1)$ is an infinite word $d_0 d_1 d_2 \cdots$ of digits in $B = \{0, \ldots, b-1\}$ such that $\alpha = \sum_{i \geq 0} d_i b^{-i-1}$. Such a representation is not always unique; for example, both infinite words 10^ω and 01^ω are 2-representations of $1/2$. The *normal b-representation* of α is the unique representation which does not end with $(b-1)^\omega$.

Let A be the alphabet $\{0, \ldots, k\}$ where $k \geq b-1$. Each infinite word $a_0 a_1 a_2 \cdots$ over A has a numerical value $\alpha = \sum_{i \geq 0} a_i b^{-i-1}$. This real number α has a normal representation in base b. Since α might be greater than 1, this normal representation has the form $d_{-n} \cdots d_{-1} \cdot d_0 d_1 d_2 \cdots$ where $d_{-n} \cdots d_{-1}$ is the b-representation of the integer part of α and $d_0 d_1 d_2 \cdots$ is the normal representation of its fractional part. By *b-normalization* over the alphabet $A = \{0, ..., k\}$, we mean the function which maps each infinite words $a_0 a_1 a_2 \cdots$ over A to the normal b-representation of its numerical value.

Proposition 1. *For any integer $k \geq b-1$, b-normalization over the alphabet $\{0, \ldots, k\}$ is a right-sequential function.*

Proof. Let q be the integer $\lfloor k/(b-1) \rfloor$ and let Q be the set $\{0, \ldots, q\} \times \{0, 1\}$. We define a Büchi transducer $\mathcal{T}$ whose state set is Q. All states are initial and the initial output function ι is defined by $\iota(p, e) = \langle p \rangle_b$ where $\langle p \rangle_b$ is the b-representation of the integer p. The set of final states is $\{0, \ldots, q\} \times \{0\}$. The set E of transitions is given by

$$E = \{(p, \delta_{r,b-1}) \xrightarrow{\ell|r} (p', e') \mid \ell + p' = pb + r\}$$

where the Kronecker symbol $\delta_{i,j}$ is defined, as usual, by $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ otherwise.

For simplicity, the result has been stated and proved for the normalization over the alphabet $\{0, \ldots, k\}$ but it also holds for larger alphabets containing negative digits.

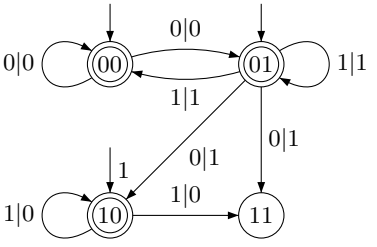


Fig. 6. Normalization with $b = 2$ and $k = 1$

Example 6. The simplest case of normalization is for $b = 2$ and $k = 1$. In that case, the transducer just replaces each tail 01^ω by the tail 10^ω. The transducer depicted in Fig. 6 realizes this transformation. State $(1, 1)$ is useless but it has been kept in the figure for completeness. It can be verified that this transducer is right-sequential, by using the definition of prophetic automata based on a partition of A^ω.

Corollary 1. *Multiplication of real numbers represented in some base b by a fixed integer is a right-sequential function.*

Since the transducer of Example 1 realizes division by 3, a transducer realizing multiplication by 3 can be obtained by exchanging the input and the output labels of each transition. The resulting transducer is still real-time but it is neither left-sequential nor right-sequential.

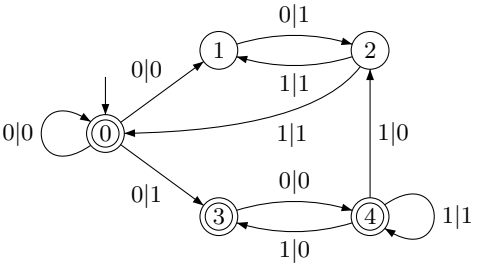


Fig. 7. Transducer for multiplication by 3 in base 2

Example 7. The transducer depicted in Fig. 7 is right sequential. It realizes multiplication by 3.

Let p be a fixed integer. If the infinite word $d_0d_1d_2\cdots$ is a b-representation of α, then the infinite word $(pd_0)(pd_1)(pd_2)\cdots$ is obviously a representation of $p\alpha$. It suffices then to apply b-normalization over the alphabet $\{0, \cdots, p(b-1)\}$ to get the normal b-representation of $p\alpha$.

Corollary 2. *Addition of real numbers is a right-sequential function.*

If $d_0d_1d_2\cdots$ and $d'_0d'_1d'_2\cdots$ are b-representations of α and α', then $(d_0+d'_0)(d_1+d'_1)(d_2+d'_2)\cdots$ is obviously a representation of $\alpha+\alpha'$. It suffices then to apply b-normalization over the alphabet $\{0,\ldots,2b-2\}$ to get the normal b-representation of $\alpha+\alpha'$.

4.2 Temporal Logic

We consider here future linear temporal logic with discrete time. We assume that a set AP of atomic propositions is fixed. The formulas are built from these atomic propositions using the Boolean connectives and the operators $\bigcirc$ (Next) and $\mathtt{U}$ (Until).

$$\varphi := \mathrm{AP} \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \,\mathtt{U}\, \varphi$$

A *model* is an infinite word over the alphabet 2^{AP}. For $k \geq 0$, the letter x_k of the model $x = x_0x_1x_2\cdots$ is the set of valid atomic propositions at time k in the model x. The semantics is then inductively defined as usual.

- $x, k \models p$ if $p \in x_k$;
- $x, k \models \neg\varphi$ if $x, k \not\models \varphi$;
- $x, k \models \varphi \vee \psi$ if $x, k \models \varphi$ or $x, k \models \varphi$;
- $x, k \models \bigcirc\varphi$ if $x, k+1 \models \varphi$;
- $x, k \models \varphi \,\mathtt{U}\, \psi$ if there exists $j \geq k$ such that $x, j \models \psi$ and for any $k \leq i < j$, one has $x, i \models \varphi$.

Let φ be a formula and let $x = x_0x_1x_2\cdots$ be a model. The *word of truth value* of x is the infinite word $y_0y_1y_2\cdots$ where $y_i = 1$ if $x, i \models \varphi$ and $y_i = 0$ otherwise. This word is denoted by $\varphi(x)$.

Proposition 2. *Let φ be a formula. The function that maps any model x to its word of truth value $\varphi(x)$ is a purely right-sequential function.*

This proposition gives another proof that LTL is decidable but it does not yield a polynomial time algorithm in the size of the formula. There is an exponential blow-up of the number of states of the constructed transducer.

Proof. Each Boolean connective and each operator Next and Until can be viewed as an operator on words of truth value. For instance, the word of truth value $(\varphi \,\mathtt{U}\, \psi)(x)$ only depends on the words of truth value $\varphi(x)$ and $\psi(x)$. It is then sufficient to show that each of these operators can be realized by a right-sequential transducer. The transducer for a formula φ is then built using the composition of the transducers. It is easy to show that purely right-sequential transducers can be composed.

The transducers for the Boolean connectives are trivial. The Next operator is essentially the shift function. A purely right-sequential transducer realizing this

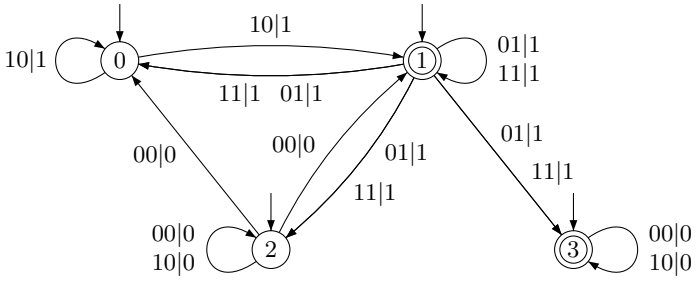


Fig. 8. Transducer for the Until operator

function has been given in Example 5. The most interesting transducer is the one realizing the Until operator.

The transducer depicted in Fig. 8 realizes the Until operator. It takes as input two words $\varphi(x)$ and $\psi(x)$ and output the word $(\varphi \ \mathtt{U} \ \psi)(x)$. The label of each transition has the form $ab|c$ where a and b are symbols from $\varphi(x)$ and $\psi(x)$ and c is a symbol of $(\varphi \ \mathtt{U} \ \psi)(x)$. It can be verified that this transducer is purely right-sequential.

5 Decomposition of Rational Functions

In this section, we state and we prove the main result of the paper. It has been proved by Elgot and Mezei [7] that each rational function of finite words is the composition of a left-sequential function and a right-sequential function. We refer the reader to [3, p. 125] or [13] for more structured presentations of this result. Here we extend this result to infinite words.

Theorem 1. *Any rational partial function $f : A^\omega \to B^\omega$ can be decomposed $f = g \circ h$ where $g : C^\omega \to B^\omega$ is a left-sequential partial function and $h : A^\omega \to C^\omega$ is a purely right-sequential partial function. The function h can always be chosen to be letter-to-letter. Moreover, if the function f is letter-to-letter, the function g can be chosen to be letter-to-letter.*

As in [3,13], the proof is mainly based on a adaptation of a construction due to Schützenberger [14]. Roughly speaking, the main idea in [3,13] is to consider the synchronized product of a real-time transducer with the automaton obtained by the subset construction applied to the input automaton of the transducer. Some transitions of this product must then be removed in a clever way to make it unambiguous. The proof of our result follows these lines, but the subset construction is replaced by another construction that yields an equivalent prophetic automaton from any Büchi automaton. This construction is borrowed from [4].

6 An Application to Automatic Sequences

Automatic sequences are sequences of letters from a finite alphabet that are generated by an automaton. They are also the pointwise images of fixed points

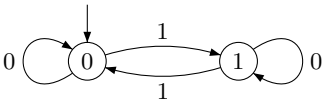


Fig. 9. Automaton for the Thue-Morse word

of uniform morphisms. Much of the interest in these sequences stems from their numerous connections with number theory. The most famous result in this area is Christol's theorem, which states that the formal power series $\sum_{n\geq 0} a_n X^n$ is algebraic over $\mathbb{F}_k(X)$ if and only if the sequence $(a_n)_{n\geq 0}$ is k-automatic. We refer the reader to [1] for a complete introduction to automatic sequences.

For the sake of completeness, we recall the definition of automatic sequences. Let $k \geq 2$ be an integer and let B be the alphabet $\{0, \ldots, k-1\}$. The representation in base k of an integer n is denoted $\langle n \rangle_k$. An infinite word $x = x_0 x_1 x_2 \cdots$ over A is *k-automatic* if there is a deterministic automaton $\mathcal{A} = (Q, B, E, \{i\}, F)$ and a function $\lambda : Q \to A$ such that for each integer n, $x_n = \lambda(i \cdot \langle n \rangle_k)$ where $i \cdot w$ denotes the state reached after reading the finite word w from the initial state i. In other words, the n-th element of the sequence is determined by the state reached after reading the k-representation of n in the automaton $\mathcal{A}$.

Example 8. The Thue-Morse word is the infinite word $x = x_0 x_1 x_2 \cdots$ over $A = \{0, 1\}$ defined by $x_n = 0$ if n has an even number of 1's in its 2-representation and $x_n = 1$ otherwise. This word is 2-automatic since the automaton depicted in Fig. 9 satisfies $x_n = 0 \cdot \langle n \rangle_2$ for each $n \geq 0$. The function λ is the identity.

Automatic sequences are closed under many transformations. It is for instance stated in [1, Theorem 6.9.2] that automatic sequences are closed under letter-to-letter left-sequential transducer. What is called a transducer in [1] is actually a left-sequential transducer in our terminology (see definition p. 140). We show here that this result can be extended to any functional letter-to-letter transducer.

Theorem 2. *Let f be a function realized by a letter-to-letter transducer. If the infinite word x is k-automatic, the infinite word $f(x)$ is also k-automatic.*

7 Conclusion

As a conclusion, let us sketch a few problems that are raised by our work.

In the case of finite words, left- and right-sequential functions play a symmetrical role. It follows from Elgot and Mezei's result that a rational function of finite words is either the composition of a left-sequential function with a right-sequential function or the composition of a right-sequential function with a left-sequential function. In this paper, we have only proved that a rational function of infinite words is the composition of a left-sequential function with a right-sequential function. The other result does not follow from symmetry arguments and it is still open.

Sequential functions of finite words have a quasi-topological characterization due to Choffrut [5]. This characterization gives a necessary and sufficient condition for a transducer, called the *twinning property*, to realize a sequential function. This characterization has been extended to left-sequential functions of infinite words in [2] with a variant of the twinning property. Moreover, the twinning property and its variant can be checked in polynomial time [15]. It would be nice to have similar characterizations for right-sequential functions of infinite words.

References

1. Allouche, J.-P., Shallit, J.: Automatic Sequences. Cambridge University Press, Cambridge (2003)
2. Béal, M.-P., Carton, O.: Determinization of transducers over infinite words: the general case. TOCS 37(4), 483–502 (2004)
3. Berstel, J.: Transductions and Context-Free Languages. B.G. Teubner (1979)
4. Carton, O., Michel, M.: Unambiguous Büchi automata. Theo. Comput. Sci. 297, 37–81 (2003)
5. Choffrut, C.: Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. Theo. Comput. Sci. 5, 325–337 (1977)
6. Cohen, A., Collard, J.-F.: Instance-wise reaching definition analysis for recursive programs using context-free transductions. In: Parallel Architectures and Compilation Techniques, pp. 332–340. IEEE Computer Society Press, Los Alamitos (1998)
7. Elgot, C.C., Mezei, J.E.: On relations defined by generalized finite automata. IBM J. of Res. and Dev. 9, 47–68 (1965)
8. Lind, D., Marcus, B.: An Introduction to Symbolic Dynamics and Coding. Cambridge University Press, Cambridge (1995)
9. Lothaire, M.: Algebraic Combinatorics on Words, ch. 7, pp. 230–268. Cambridge University Press, Cambridge (2002)
10. Mohri, M.: On some applications of finite-state automata theory to natural languages processing. Journal of Natural Language Engineering 2, 1–20 (1996)
11. Perrin, D., Pin, J.-É.: Infinite Words. Elsevier, Amsterdam (2004)
12. Roche, E., Schabes, Y.: Finite-State Language Processing, ch. 7. MIT Press, Cambridge (1997)
13. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press, Cambridge (2009)
14. Schützenberger, M.-P.: Sur les relations rationnelles entre monoïdes libres. Theo. Comput. Sci. 4, 47–57 (1976)
15. Weber, A., Klemm, R.: Economy of description for single-valued transducers. Inform. and Comput. 118, 327–340 (1995)

Kernelization

(Invited Talk)

Fedor V. Fomin

Department of Informatics, Univeristy of Bergen, 5020 Bergen, Norway
fomin@ii.uib.no

Preprocessing (data reduction or kernelization) as a strategy of coping with hard problems is universally used in almost every implementation. The history of preprocessing, like applying reduction rules simplifying truth functions, can be traced back to the 1950's [6]. A natural question in this regard is how to measure the quality of preprocessing rules proposed for a specific problem. For a long time the mathematical analysis of polynomial time preprocessing algorithms was neglected. The basic reason for this anomaly was that if we start with an instance I of an NP-hard problem and can show that in polynomial time we can replace this with an equivalent instance I' with $|I'| < |I|$ then that would imply P=NP in classical complexity.

The situation changed drastically with advent of Parameterized Complexity [3]. The philosophy of parameterized complexity is that beyond overall input size, key secondary measurements fundamentally affect the computational complexity of problems and govern the opportunities for designing efficient algorithms. Combining tools from parameterized complexity and classical complexity it has become possible to derive upper and lower bounds on the sizes of reduced instances, or so called *kernels*. In parameterized complexity each problem instance comes with a parameter k and the parameterized problem is said to admit a *polynomial kernel* if there is a polynomial time algorithm (the degree of polynomial is independent of k), called a *kernelization* algorithm, that reduces the input instance down to an instance with size bounded by a polynomial $p(k)$ in k, while preserving the answer. This reduced instance is called a $p(k)$ *kernel* for the problem. If $p(k) = O(k)$, then we call it a linear kernel and if$p(k) = k^{O1)}$, then a polynomial kernel.

The classical example goes back to the work of Nemhauser and Trotter [5], who gave a polynomial time algorithm that for a given graph G and integer k, reduces G to a graph K on $2k$ vertices (and thus of size $O(k^2)$) such that G has a vertex cover of size k if and only if K does. In other words, there is an $O(k^2)$ kernel for k-Vertex Cover. Other examples of kernels include a linear kernel for k-Dominating Set on planar graphs [1] and an $O(k^2)$ kernel for k-Feedback Vertex Set [7].

In this talk we give an overview of recent results and techniques for obtaining linear and polynomial kernels on planar graphs and more generally, on graphs excluding some fixed graph as a minor. The talk is based on joint works with Hans Bodlaender, Daniel Lokshtanov, Elko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos [2,4].

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 107–108, 2010.

References

1. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial-time data reduction for dominating set. J. ACM 51, 363–384 (2004)
2. Bodlaender, H., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M.: (meta) Kernelization. In: Proceedings of the 50th Annual Symposium on Foundations of Computer Science (FOCS 2009), pp. 629–638. IEEE, Los Alamitos (2009)
3. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1998)
4. Fomin, F.V., Lokshtanov, D., Saurabh, S., Thilikos, D.M.: Bidimensionality and kernels. In: Proceedings of the 21th ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 503–510 (2010)
5. Nemhauser, G.L., Trotter, L.E.: Vertex packings: Structural properties and algorithms. Math. Program. 8, 232–248 (1975)
6. Quine, W.V.: The problem of simplifying truth functions. Amer. Math. Monthly 59, 521–531 (1952)
7. Thomassé, S.: A quadratic kernel for feedback vertex set. In: Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms (SODA 2009), pp. 115–119 (2009)

Zigzags in Turing Machines*

Anahí Gajardo[1] and Pierre Guillon[2]

[1] Departamento de Ingeniería Matemática, Universidad de Concepción, Casilla 160-C, Concepción, Chile
anahi@ing-mat.udec.cl

[2] DIM - CMM, UMI CNRS 2807, Universidad de Chile, Av. Blanco Encalada 2120, Santiago, Chile
pguillon@dim.uchile.cl

Abstract. We study one-head machines through symbolic and topological dynamics. In particular, a subshift is associated to the subshift, and we are interested in its complexity in terms of realtime recognition. We emphasize the class of one-head machines whose subshift can be recognized by a deterministic pushdown automaton. We prove that this class corresponds to particular restrictions on the head movement, and to equicontinuity in associated dynamical systems.

Keywords: Turing machines, discrete dynamical systems, subshifts, formal languages.

We study the dynamics of a system consisting in a finite automaton (the head) that can write and move over an infinite tape, like a Turing machine. We use the approach of symbolic and topological dynamics. Our interest is to understand its properties and limitations, and how dynamical properties are related to computational complexity.

This approach was initiated by Kůrka in [1] with two different topologies: one focused on the machine head, and the other on the tape. The first approach was further developed in [2,3]. More recently, in [4,5], a third kind of dynamical system was associated to Turing machines, taking advantage of the following specificity: changes happen only in the head position whilst the rest of the configuration remains unaltered. The whole evolution can therefore be described by the sequence of states taken by the head and the symbols that it reads. This observation actually yields a factor map between Kůrka's first dynamical system and a one-sided subshift.

In [4], it has been proved that machines with a sofic subshift correspond to machines whose head makes only bounded cycles. We prove here a similar characterization of machines with a shift that can be recognized by a deterministic pushdown automaton. Moreover, we establish links between these two properties and equicontinuity in all three spaces.

* This work has been supported by ECOS-Sud Project and CONICYT FONDECYT #1090568.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 109–119, 2010.

In the first section, we recall the definitions and fundamental results. The second section is devoted to defining the different dynamical systems associated to one-head machines, and to stating basic results about equicontinuity within these systems. In the last section, we define the class of bounded-zigzag machines and state our main results.

1 Preliminaries

Consider a finite alphabet A, and $\mathbb{M}$ to stand either for $\mathbb{N}$ or for $\mathbb{Z}$. For a finite word $u \in A^*$, we will note $|u|$ its length, and index its letters from 0 to $|u|-1$, unless specified otherwise. We denote $A^{\leq m}$ the set of words on A of length at most $m \in \mathbb{N}$. If $i, j \in \mathbb{Z}$ and $i \leq j$, $[\![i, j]\!]$ will denote the closed interval of integers $i, \dots, j$, $[\![i, j[\![= [\![i, j-1]\!]$, etc. A point $x \in A^{\mathbb{M}}$ will be called *configuration*. For a configuration or a word x, we define $x_{[\![i,j]\!]} = x_i \dots x_j$. $A \sqcup B$ will denote the disjoint union of two sets A and B.

1.1 Topological Dynamics

A *dynamical system* (DS) is a pair (X, F) where X is a metric space and F a continuous self-map of X. Sometimes the space will be implicit.

The orbit of a point $x \in X$ is the set of the $F^t(x)$ for all *iteration* $t > 0$. A point x is called *preperiodic* if there exist two naturals q, p such that $F^{q+p}(x) = F^q(x)$. If q and p are minimal, then q is called the *transient* and p the *period*. When $t = 0$, x is called *periodic*.

A point $x \in X$ is *isolated* if there is an $\varepsilon > 0$ such that the ball of radius ε and center x contains only x. A point $x \in X$ is *equicontinuous* for F if, for any $\varepsilon > 0$, there exists some $\delta > 0$ such that, for any $y \in X$ with $d(x, y) < \delta$, we have that, for all $t \in \mathbb{N}$, $d(F^t(x), F^t(y)) < \varepsilon$. The DS (X, F) is *equicontinuous* if, for any $\varepsilon > 0$, there exists some $\delta > 0$ such that, for any $x, y \in X$ with $d(x, y) < \delta$, we have that, for all $t \in \mathbb{N}$, $d(F^t(x), F^t(y)) < \varepsilon$. When X is compact, this is equivalent to having only equicontinuous points. The DS (X, F) is *almost equicontinuous* if it has a residual set of equicontinuous points.

A DS (X, F) is a *factor* of a DS (Y, G) if $\phi G = F\phi$ for some continuous onto map $\phi : Y \to X$, then called a *factor map*.

1.2 Subshifts

We can endow the space $A^{\mathbb{M}}$ of configurations with the product of the discrete topology of A. It is based on the cylinders $[u]_i = \left\{ x \in A^{\mathbb{M}} \middle| \, x_{[\![i,i+k[\![} = u \right\}$, where $i \in \mathbb{M}$, $k \in \mathbb{N}$ and $u \in A^k$; this notation shall be extended to semi-infinite words. If $\mathbb{M} = \mathbb{Z}$, $u \in A^{2r+1}$ and $r \in \mathbb{N}$, we note $[u] = [u]_{-r}$.

This topology corresponds to the metric $d : x, y \mapsto 2^{-\min_{x_i \neq y_i} |i|}$. In other words, $d(x, y) \leq 2^{-i} \Leftrightarrow x_{[\![-i,i]\!]} = y_{[\![-i,i]\!]}$; two points are "close to each other" if they coincide "around position 0". It is easy to extend this metric to spaces $A^{\mathbb{M}} \times Q$ and $A^{\mathbb{M}} \times Q \times \mathbb{Z}$. In that setting, $A^{\mathbb{M}}$ and $A^{\mathbb{M}} \times Q$ are compact, but $A^{\mathbb{M}} \times Q \times \mathbb{Z}$ is not.

The *shift* map is the function $\sigma : A^{\mathbb{M}} \to A^{\mathbb{M}}$ defined by $\sigma(x)_i = x_{i+1}$. A *subshift* Σ is a closed subset of $A^{\mathbb{M}}$ which is also invariant by σ. It can be seen as a compact DS where the map is σ.

A subshift Σ is characterized by its *language*, containing all finite patterns that appear in some of its configurations: $\mathcal{L}(\Sigma) = \left\{ z_{[\![i,j]\!]} \middle| z \in \Sigma \text{ and } i, j \in \mathbb{M} \right\}$. We denote $\mathcal{L}_n(\Sigma) = \mathcal{L}(\Sigma) \cap A^n$. If the language $\mathcal{L}(\Sigma)$ is regular, then we say that Σ is *sofic*. Equivalently, a sofic subshift can be seen as the set of labels of infinite paths in some finite arc-labeled graph; this graph basically corresponds to the finite automaton that recognizes its language, without initial nor terminal state.

Any subshift can also be defined from a set of forbidden finite patterns $\mathcal{F} \subset A^*$ by $\Sigma = \left\{ z \in A^{\mathbb{M}} \middle| \forall i, j \in \mathbb{M}, z_{[\![i,j]\!]} \notin \mathcal{F} \right\}$. If $\mathcal{F}$ can be chosen to be finite, then Σ is a subshift *of finite type* (SFT).

A DS F on $A^{\mathbb{M}}$ is completely determined by the family of its factor subshifts, *i.e.* the factors which are also subshifts in some alphabet. Up to some letter renaming, all factor subshifts of F are of the form $(\mathcal{P}(F^j(x)))_{j\in\mathbb{N}}$, where $\mathcal{P}$ is a finite partition of X into closed open sets, and $\mathcal{P}(y)$ denotes the unique element of this partition which contains $y \in X$.

1.3 Deterministic Pushdown Automata

Definition 1. *A* deterministic pushdown automaton *(DPDA) is a tuple* $(A, \Omega, \Gamma, \bot, \lambda, o_0, F)$ *where* A *is the* input alphabet*,* Ω *is the* set of states *,* Γ *is the* stack alphabet*,* $\bot \in \Gamma$ *is the* stack bottom*,* o_0 *is the* initial state*,* $F \subset \Omega$ *is the* subset of terminal states *and* $\lambda : A \times \Omega \times \Gamma \to \Omega \times \Gamma^{\leq 2}$ *is the* transition function *such that: if* $\lambda(a, o, \bot) = (o', \mu)$*, then* μ *contains exactly one* $\bot$*, which is on its end, and if* $\lambda(a, o, \beta) = (o', \mu)$ *with* $\beta \neq \bot$*, then* μ *does not contain any* $\bot$*.*

An (infinite) arc-labeled graph G *is associated to the automaton. Its set of vertices is* $\Omega \times (\Gamma \setminus \{\bot\})^*\bot$*, and there exists an arc from* (e, μ) *to* (f, ν) *labeled* a *if and only if* $\nu = \rho\mu_{[\![1,|\mu|-1]\!]}$ *and* $\lambda(a, e, \mu_0) = (f, \rho)$*. The word* μ *is called the* stack content*.*

The language L *recognized by the automaton consists of all words* w *in* A^* *such that there exists a finite path in* G *with label* w*, starting on vertex* $(o_0, \bot)$ *and ending in some vertex* (o, μ) *with* $o \in F$*. A subshift is recognized by the automaton if its language is recognized by the automaton.*

2 Turing Machines

In this article, a Turing Machine (TM) is a triple (A, Q, δ), where A and Q are the finite *tape alphabet* and *set of state*, and $\delta : A \times Q \to A \times Q \times \{-1, 1\}$ the *rule*. We do not particularize any halting state. We can see the TM as evolving on a bi-infinite tape. The phase space is $X = A^{\mathbb{Z}} \times Q \times \mathbb{Z}$. Any element of X is called a configuration and represents the state of the tape, the state of the head and its position. We consider here the topology introduced in Section 1.1. Thus, the

farther the head is from the center, the less important become the read symbols, but the head state and position remain important. On this (non-compact) space, $T : X \to X$ by $T(x, q, i) = (x_{]-\infty,i[}ax_{]i,\infty[}, p, i + d)$ if $\delta(x_i, q) = (a, p, d)$ gives the corresponding DS. We can extend the shift function to TM configurations by $\sigma : (x, q, i) \mapsto (\sigma(x), q, i - 1)$, and it clearly commutes with T.

We can represent the head state and position by adding a "mark" on the tape. If we want a compact space, this corresponds to the following phase space:

$$X_H = \left\{ x \in (A \sqcup (A \times Q))^{\mathbb{Z}} \middle| \, |\{ i \in \mathbb{Z} | \, x_i \in A \times Q\}| \leq 1 \right\}$$

where the head position is implicitly given by the only cell with a symbol in $(A \times Q)$, and the function $T_H : X_H \longrightarrow X_H$ is defined by $T_H(x_{]-\infty,i[}(b, q)x_{]i,\infty[}) = y_{]-\infty,i+d[}(y_{i+d}, p)y_{]i+d,\infty[}$, where $y = x_{]-\infty,i[}ax_{]i,\infty[}$ and $\delta(b, q) = (a, p, d)$, and $T_H(x) = x$ if x does not contain any symbol in $A \times Q$. With the topology of X_H as a subshift of $(A \sqcup (A \times Q))^{\mathbb{Z}}$, the head state and movement are less important when the head is far from 0. This model corresponds to the TM *with moving head* defined by Kůrka in [1], which highlights the tape configuration. It is a particular case of cellular automaton, *i.e.* based on some uniformly-applied local rule. We can intuitively see a continuous injection $\Phi : X \to X_H$ such that $\Phi T = T_H \Phi$ and $\Phi\sigma = \sigma\Phi$.

Focusing on the movements and states of the head, [1] also defines the system *with moving tape* $T_T : X_T \to X_T$ on the (compact) space $X_T = A^{\mathbb{Z}} \times Q$ by $T_T(x, q) = (\sigma^d(x_{]-\infty,0[}ax_{]0,\infty[}), p)$ if $\delta(x_0, q) = (a, p, d)$. Here the head is assumed to be always at position 0, and the tape is shifted at each step according to the rule. There is a continuous non-injective surjection $\Psi : X \to X_T$ such that $\Psi T = T_T \Psi$.

Finally, we can have a vision centered on the head and which emphasizes only the relevant part of the configuration, as in [4,5]. The system S_T is the one-sided subshift on alphabet $Q \times A$, which is the image of the factor map $\tau_T : X_T \to S_T$ defined by $\tau_T(x, q)_t = (y_0, p)$ if $(y, p) = T_T^t(x, q)$. In other words, it represents the sequence of pairs corresponding to the successive states of the head and the letters that it reads.

$$\begin{array}{ccccccc} X_H & \xleftarrow{\Phi} & X & \xrightarrow{\Psi} & X_T & \xrightarrow{\tau_T} & S_T \\ \downarrow{\scriptstyle T_H} & & \downarrow{\scriptstyle T} & & \downarrow{\scriptstyle T_T} & & \downarrow{\scriptstyle \sigma} \\ X_H & \xleftarrow{\Phi} & X & \xrightarrow{\Psi} & X_T & \xrightarrow{\tau_T} & S_T \end{array}$$

Similarly, we will note S_H the one-sided subshift on alphabet $Q \sqcup (A \times Q)$ which is the image of the factor map $\tau_H : X_H \to S_H$ defined by $\tau_H(x)_t = T_H^t(x)_0$. Unlike S_T, this subshift does not always contain the relevant information, since the head can be completely absent.

2.1 Equicontinuous Configurations

Topological notions can actually formalize various types of head movements. One first example is equicontinuity of the DS T_T. It is strongly related with

periodicity, as the next remark establishes. This is natural since the symbol that the head reads in X_T is always at position 0. Hence, if the head visits an infinite number of cells, say to the right, any perturbation on the initial configuration will get to position 0, and thus will become largely significant for this topology. We conclude the following.

Remark 1. *Let $x \in X$ be a configuration and T a machine over X. The following statements are equivalent:*

1. *The head position on x is bounded.*
2. *x is **preperiodic** for T.*
3. *$\Phi(x)$ is **preperiodic** for T_H.*
4. *$\Psi(x)$ is **equicontinuous** for T_T.*
5. *$\tau_T\Psi(x)$ is **preperiodic** and **isolated** –i.e. **equicontinuous**– in S_T.*

Moreover, if one of the above occurs, then $\Psi(x)$ is preperiodic for T_T, x is equicontinuous for T and $\Phi(x)$ is equicontinuous for T_H. The set of equicontinuous configurations for T_T is a union of cylinders of X_T.

If $\Psi(x)$ is preperiodic for T_T, then $\tau_T\Psi(x)$ is also periodic (for σ), but x need not be periodic for T. For example, a machine that simply moves to the left on every symbol will produce a periodic point for T_T if the initial configuration x is spatially periodic. From the previous remark, such a point is not equicontinuous, and $\tau_T\Psi(x)$ is a non-isolated periodic point in S_T, because any perturbation of x will produce a neighbor of $\tau_T\Psi(x)$ in S_T. Periodic points for T generate isolated periodic points in S_T because, once the system falls in the periodic behavior, its future is fixed.

Preperiodicity in T also implies equicontinuity in T_H, but T_H may have other equicontinuous points. The previously mentioned machine which always go to the left produces equicontinuous points for T_H which are not equicontinuous nor preperiodic for T_T.

The following proposition states that the equicontinuity of preperiodic configurations is transmitted to factor subshifts of T_H, which will be helpful in the sequel.

Proposition 1. *If $z \in S_H$ is a preperiodic word involving the machine head infinitely often, then it is isolated.*

Proof. We can assume that z is periodic, and then include the transient evolution in a larger ball. Let $p \in \mathbb{N} \setminus \{0\}$ be the period of z; let us prove that the ball $U = [z_{[\![0,|Q||A|^{p+1}(p+1)^2]\!]}]_0$ of S_H is equal to $\{z\}$. Let $z' \in U$ and $x \in \tau_H^{-1}(z')$. It can be seen that the head computing over z' always remains between the positions $\lfloor -p/2 \rfloor$ and $\lfloor p/2 \rfloor$, which correspond to at most $|Q|\,|A|^{p+1}\,(p+1)$ distinct finite patterns. Hence there are $i < j \leq |Q|\,|A|^{p+1}\,(p+1)$ such that $T^i(x) = T^j(x)$; as a consequence $\sigma^i(z')$ is $(j-i)$-periodic. Together with $\sigma^i(z)$, they are both $(j-i)p$-periodic and coincide on their first $(j-i)p$ letters, since $(j-i)p \leq |Q|\,|A|^{p+1}\,(p+1)^2 - i$. As a conclusion, $z' = z$. □

2.2 Preperiodic Machines

When all the configurations are uniformly preperiodic, we say that the system is preperiodic, *i.e.* there exist q, p such that $T^{q+p} = T^q$. In the present case, global preperiodicity of each of the considered systems comes directly from local preperiodicity of T; and it is equivalent to global equicontinuity of each of the systems as the next theorem establishes.

Theorem 1. *Considering a machine, the following statements are equivalent:*

1. *The head position is (uniformly) bounded.*
2. *Any configuration of X (or X_H, X_T) is* **preperiodic**.
3. *T (or T_H, T_T, S_T, S_H) is* **preperiodic**.
4. *T (or T_H, T_T, S_T, S_H) is* **equicontinuous**.
5. *S_T (or S_H) is* **finite**.

Proof. We give only a sketch of the main implications.

- It is quite obvious from the commutation diagrams that the preperiodicity of T, T_H and T_T are equivalent, and they imply those of S_T and S_H. They also imply, from Remark 1, that the head position is bounded.
- Clearly, the equicontinuity of T and T_H are equivalent.
- It is known from cellular automata theory that the equicontinuity of T_H, its preperiodicity, that of all its configuration and the finiteness of S_H are equivalent.
- If the head position on all configurations is bounded, then from Remark 1 they are all equicontinuous for T_T. X_T being compact, T_T is equicontinuous.
- It is obvious that S_T is finite if and only if the head reads a bounded part of the initial configuration. □

2.3 Sofic Machines

Now we allow computations where the head can go arbitrarily "far", but without ever making "large" movements back.

Definition 2. *We say that a machine makes a* right-cycle *(*left-cycle*) of width $N \in \mathbb{N}$ over a configuration $x \in A^{\mathbb{Z}} \times Q \times \mathbb{Z}$ and a cell $i \in \mathbb{Z}$ if there exist time steps $0 = t_0 < t_1 < t_2$ such that the head position is i at time 0 and t_2, and is $i + N$ $(i - N)$ at time t_1.*

In this section, we consider machines whose cycles have bounded width, *i.e.* there exists an integer N such that the machine cannot make any cycle wider than N. These machines have been studied in [5,4], where it was proved that they are exactly the machines for which S_T is sofic.

Theorem 2. *Considering a machine, the following statements are equivalent:*

1. *S_T is* **sofic**.
2. *All configurations of X_H that contain the head are* **equicontinuous**.

Proof

1⇒2 From [4], we know that there exists an integer N such that the machine cannot make any cycle wider than $N \in \mathbb{N}$, and let $x \in X_H$ a configuration containing the head within $[\![-k, k]\!]$, for some $k \in \mathbb{N}$. Let us show that if $y \in [x_{[\![-k-N,k+N]\!]}]$, then for every $t \in \mathbb{N}$ we have $T_H^t(y) \in [T_H^t(x)_{[\![-k,k]\!]}]$. Let us remark that while the head is inside $[\![-k-N, k+N]\!]$, we necessarily have $T_H^t(y) \in [T_H^t(x)_{[\![-k,k]\!]}]$. Let us suppose that there exists $j \in \mathbb{N}$ such that the head is outside $[\![-k-N, k+N]\!]$ at time j and let us take this j minimal. Then the heads of $T_H^j(x)$ and $T_H^j(y)$ are outside $[\![-k-N, k+N]\!]$. At some moment, the head has gone from k to $k+N$ (or from $-k-N$); if it comes back to $[\![-k, k]\!]$, it would make a cycle. Therefore, the head cannot come back to $[\![-k, k]\!]$, and this is true both for x and y, and we have the result.

2⇒1 Conversely, assume that the head can do arbitrarily wide right-cycles in cell 0, *i.e.* for each $j \in \mathbb{N}$ there exists a cylinder $[u^j]_0$ of X_H with $u^j \in (A \times Q)A^{n_j}$, with $n_j > j$, such that over each configuration of $[u^j]_0$, the head starts at 0, it visits the whole interval $[\![0, n_j]\!]$ and comes back to cell 0. Let us take some configuration c^j in each cylinder $[u^j]_0$. By compactness, the sequence $(c^j)_{j\in\mathbb{N}}$ admits an adhering value c, on which the head necessarily goes infinitely far to the right without ever coming back to cell 0. By construction, for any N, there is some $j \in \mathbb{N}$ such that the configuration $c^j_{[\![-N,N]\!]} = c_{[\![-N,N]\!]}$. But there exists a time $t \in \mathbb{N}$ such that $T_H^t(c^j)$ has the head in cell 0, whilst $T_H^t(c)$ has not; hence c is not equicontinuous. □

From [5], any of the former properties implies that any configuration is either preperiodic or gives rise to a movement of the head arbitrarily far in some direction, but the converse is not true. Any configuration of S_H is preperiodic, hence this subshift is numerable.

3 Bounded-Zigzag Machines

Whilst the sofic machines did not allow any large cycle, we can wonder what happens when allowing a single one, or a finite number of these. The first step is to allow one cycle of arbitrary width but to forbid two overlapped unbounded cycles (zigzags). We remark that two independent cycles, each on a different direction, are allowed in this case.

Definition 3. *We say that a machine makes a* right-zigzag *(resp.,* left-zigzag*) of width $N \in \mathbb{N}$ over a configuration $x \in A^{\mathbb{Z}} \times Q \times \mathbb{Z}$ and a cell $i \in \mathbb{Z}$, if there exist time steps $0 = t_0 < t_1 < t_2$ such that the machine position is i at times t_0 and t_2, and $i+N$ (resp., $i-N$) at time t_1. We say that a machine is* bounded-zigzag *if the maximal width of the zigzags that it can make is finite.*

3.1 Complexity of S_T

While bounded cycle machines have a sofic shift S_T, the bounded-zigzag machines have a subshift recognized by a deterministic pushdown automata. The

words of S_T contain information about the tape symbols and the head state. From this data, it is possible to deduce the tape symbol of the visited cells and the relative position of the head at each time step. In order to recognize S_T, we can register this information and check its coherence at each time step. When the width of the cycles is bounded, we only need to register a finite part of the tape (bounded-cycle machines have a subshift that can be recognized by a finite state automaton).

When only one "wide" cycle can be done, we can register the information in a stack, from which it can be read exactly once (and is lost forever once read). This corresponds to the fact that the cells registered in the stack cannot be visited any more and zigzags cannot be allowed. The complete proof can be found in [6].

Theorem 3. *A machine T is bounded-zigzag if and only if S_T is recognized by some deterministic pushdown automaton.*

3.2 Complexity of S_H

If we now adopt a point of view fixed on the tape $-S_H-$ rather than the head, a cycle in the subshift corresponds to a waiting time during which cell 0 does not change. We can adapt the previously built DPDA so that it recognizes exactly these waiting words between two visits of the head. The key point here is that these languages are unary, and unary context-free languages are regular (see for example [7]), and thus they can be recognized with a finite automaton.

When the machine is bounded-zigzag, the head can make at most one long cycle by side. The rest of the time, the head is either moving closer to or farther from cell 0, or staying in some finite window around cell 0. All of these behaviors can be recognized by a finite automaton, thus the language of S_H is regular. Therefore, we obtain a surprising reduction in language complexity when changing the point of view: if S_T is recognized by some DPDA, then S_H is sofic. The complete proof can be found in [6]. Note that, up to a rescaling of the tape alphabet, all factor subshifts can be reduced to the case of S_H.

Theorem 4. *For any bounded-zigzag machine, all the factor subshifts of T_H are sofic.*

The converse of this theorem is false: we can construct a machine with a tape with n levels, where the head vertically shifts down the content of each level while moving right. It rebounds when it finds a wall in the lowest level (which is erased in the same way), and does the same in the opposite direction. We can see that the machine can make arbitrarily wide n-zigzags, each of independent length, in such a way that the factor subshifts of T_H are sofic.

Nevertheless, we can prove that this kind of construction is possible only with a bounded n. Let us introduce this formally.

Definition 4. *We say that a machine makes an* n-cycle *of width $N \in \mathbb{N}$ over configuration $x \in A^{\mathbb{Z}} \times Q \times \mathbb{Z}$ and cell $i \in \mathbb{Z}$, if there exist $2n+1$ time steps $0 = t_0 < t_1 < \ldots < t_{2n}$ such that the head is in position i at time t_{2q} and outside $[\![-N, N]\!]$ at time t_{2q+1}, for each $q \in [\![0, n]\!]$. We say that the machine is*

n-bounded-cycle if there is some N such that the head cannot make n-cycles of width larger than N.

When S_T is sofic, the machine is 1-bounded cycle. Considering some machine T, we denote $\psi_N(x) \in \mathbb{N} \sqcup \{+\infty\}$ the maximum n such that the machine can make an n-cycle of width N over configuration x. Clearly, T is n-bounded cycle if and only if for some $N \in \mathbb{N}$, ψ_N is bounded by $n-1$.

Let us call $\Phi_i(x)$ the set of time steps for which the head has position $i \in \mathbb{Z}$ when computing over configuration x. This set is linked to cycles by the following intuitive observation.

Proposition 2. *If T is an n-bounded-cycle machine, then there exists $p \in \mathbb{N}$ such that for any cell $i \in \mathbb{Z}$ and any non-preperiodic configuration $x \in X$, $|\Phi_i(x)| \leq p$.*

Proof. Let $n, N \in \mathbb{N}$ be such that $max\{\psi_N(x) | x \in X\} = n-1$, and $x \in X$ such that $|\Phi_0(x)| > p = 2n|A|^{2N+1}$ – the case $i \neq 0$ can be obtained by shifting. Consider $\{t_0, \dots, t_p\} \subset \Phi_0(x)$ with $t_0 < t_1 < \dots < t_p$. If we consider an $(n-1)$-cycle over x in cell 0, we can see that there exist $t_{k_1} < t_{k_2} < \dots < t_{k_{n-1)}}$ such that for any $i \in [\![1, n-1]\!]$, the head goes beyond N or $-N$ between time steps t_{k_i} and t_{k_i+1}, but not between (possibly equal) times t_{k_i+1} and $t_{k_{i+1}}$. This means that t_{k_i} is the last time that the head is in 0 before going beyond $[\![-N, N]\!]$. Let $k_0 = -1$ and $k_n = p$, in such a way that $[\![0, p]\!] = \bigcup_{i=0}^{n} I_i$, where $I_i = [\![k_i + 1, k_{i+1}]\!]$ for $0 \leq i \leq n$. There are $n+1$ such intervals, so one of them, say I_i, has at least $|A|^{2N+1}$ elements; this is all the more the case for $[\![t_{k_i+1}, t_{k_{i+1}}]\!] \supset \{t_{k_j} | k_i < j \leq k_{i+1}\}$. Hence, between time steps t_{k_i+1} and $t_{k_{i+1}}$ there are at least $|A|^{2N+1}$ consecutive time steps in $\Phi_0(x)$ such that the head stays within the interval of cells $[\![-N, N]\!]$. As a result, there are $i, j \in [\![t_{k_i+1}, t_{k_{i+1}}]\!]$ with $i < j$ and $T^i(x) = T^j(x)$, which implies that x is preperiodic. □

Theorem 5. *If S_H is sofic, then T is n-bounded-cycle for some n.*

Proof. Assume that $S_H = \tau_H(X_H)$ is recognized by some finite automaton with N states, and that there exists some configuration $x \in X$ on which the machine makes some N-cycle of width N. Let $t_0, \dots, t_{2N}$ be as in the definition of N-cycles, and $u = \tau_H(x)_{[\![0,t_{2N}[\![}$. Let $o_0 \dots o_{t_{2N}+1}$ be the corresponding path of the finite automaton. We can see that there are $i < j < N$ such that $o_{t_{2i}} = o_{t_{2j}}$, hence there is some periodic infinite word $z \in \tau_H(X_H)$ corresponding to the path w that repeats the cycle $(o_{t_{2i}} \dots o_{t_{2j}})$. From Proposition 1, z is isolated. As a consequence, w is the only path to start from $o_{t_{2i}}$. Therefore, its vertices are all different, and $t_{2j} - t_{2i} \leq N$, but in this case the head does not have the time to go beyond $[\![-N, N]\!]$ between these two iterations, which is a contradiction. We have proved that T is N-bounded-cycle. □

Here, too, the converse is false, since it is easy to build a machine doing a given number of arbitrarily wide rebounds on specific wall characters before stopping. The language of such a machine cannot be regular because the time intervals between two rebounds are not independent.

3.3 Almost Equicontinuity

We have already seen that in sofic machines, almost all configurations of X_H are equicontinuous. It is still so when allowing n-cycles, though in this case there are some configurations with head which are not equicontinuous – recall that Theorem 2 is an equivalence.

Theorem 6. *If T is an n-bounded-cycle machine for some n, then T_H is almost equicontinuous.*

Proof. By compactness of the space, it is enough to prove that for any cylinder $[u]$ and any $k \in \mathbb{N}$, there exist some $x \in [u]$ and some $m \in \mathbb{N}$ such that for any $y \in [x_{[\![-m,m]\!]}]$ and any $t \in \mathbb{N}$, $T_H^t(y) \in [T_H^t(x)_{[\![-k,k]\!]}]$. Let $N \in \mathbb{N}$ be as in the definition of n-bounded-cycle machine, $[u]$ a cylinder of X_H and $k \in \mathbb{N}$. If $[u]$ contains some preperiodic configuration with the head, then we can easily find m thanks to Remark 1. Otherwise, let us consider some configuration $x \in [u]$ (with the head) maximizing $|\Phi_{-k}(x) \sqcup \Phi_k(x)|$, which is finite thanks to Proposition 2. Let $m \in \mathbb{Z}$ be such that $m \geq k$ and the interval $[\![-m, m]\!]$ contains all the cells visited, when computing from x, up to time step $t = \max(\Phi_{-k}(x) \sqcup \Phi_k(x))$. Then we can see that any configuration $y \in [x_{[\![-m,m]\!]}]$ has the same evolution as x until this time step, and that after that, its head cannot visit cell $-k$ nor k, otherwise it would contradict the maximality of x. We can deduce that the head of x (then also y) is outside $[\![-k, k]\!]$ after iteration t, otherwise it would be trapped between $-k$ and k and would become periodic. We observe, then, that the cells of $[\![-k, k]\!]$ evolve exactly in the same way for configurations x and y. □

The converse is untrue: imagine a machine whose head rebounds between two walls, each time shifting them to the left. Every configuration where the head starts enclosed between two walls is equicontinuous. Any finite pattern can be extended by adding walls to enclose the head, therefore equicontinuous points are dense, but the head can make an arbitrary number of arbitrarily wide cycles.

4 Conclusion

The complexity of the Turing machine will always be very hard to understand. In our attempt to treat this issue through the theories of topological and symbolic dynamics, we have found interesting relations between:

- The head movements that can be observed during the computation;
- The density of equicontinuous points;
- The language complexity of the associated subshifts S_T and S_H.

These relations introduce a new point of view on how computation is performed. In addition to generalizing them to more machines, the next step would be to study Turing machines as computing model by introducing a halting state, and to link all of these considerations to the result itself of the computation, and eventually the temporal or spatial complexity of the computation.

References

1. Kůrka, P.: On topological dynamics of Turing machines. Theoret. Comput. Sci. 174(1-2), 203–216 (1997)
2. Blondel, V.D., Cassaigne, J., Nichitiu, C.: On the presence of periodic configurations in Turing machines and in counter machines. Theoret. Comput. Sci. 289, 573–590 (2002)
3. Oprocha, P.: On entropy and Turing machine with moving tape dynamical model. Nonlinearity 19, 2475–2487 (2006)
4. Gajardo, A., Mazoyer, J.: One head machines from a symbolic approach. Theoret. Comput. Sci. 370, 34–47 (2007)
5. Gajardo, A.: Sofic one head machines. In: Durand, B. (ed.) Journées Automates Cellulaires, pp. 54–64 (2008)
6. Gajardo, A., Guillon, P.: Zigzags in turing machines (February 2010), http://fr.arxiv.org/abs/1003.0588
7. Ginsburg, S., Rice, H.: Two families of languages related to algol. J. ACM 9(3), 350–371 (1962)

Frameworks for Logically Classifying Polynomial-Time Optimisation Problems

James Gate and Iain A. Stewart

School of Engineering and Computing Sciences, Durham University,
Science Labs, South Road, Durham DH1 3LE, U.K.
{j.s.gate,i.a.stewart}@durham.ac.uk

Abstract. We show that a logical framework, based around a fragment of existential second-order logic formerly proposed by others so as to capture the class of polynomially-bounded P-optimisation problems, cannot hope to do so, under the assumption that $\mathsf{P} \neq \mathsf{NP}$. We do this by exhibiting polynomially-bounded maximisation and minimisation problems that can be expressed in the framework but whose decision versions are NP-complete. We propose an alternative logical framework, based around inflationary fixed-point logic, and show that we can capture the above classes of optimisation problems. We use the inductive depth of an inflationary fixed-point as a means to describe the objective functions of the instances of our optimisation problems.

1 Introduction

The theory of computational complexity is primarily concerned with the classification of decision problems, and although many (NP-complete) decision problems are actually decision versions of more natural optimisation problems, the classification of optimisation problems does not fit naturally into many of the available standard classification frameworks. While there do exist criteria against which we can classify optimisation problems, such as according to their approximation properties [11], it was not until Papadimitriou and Yannakakis [18] proposed the use of existential second-order logic as a means for classification that a natural and robust framework became available. The classification of optimisation problems within this logical framework was subsequently significantly clarified by Panconesi and Ranjan [17] and Kolaitis and Thakur [13,14] (we briefly explain Kolaitis and Thakur's work later).

The optimisation problems considered in the papers above are (polynomially-bounded) NP-optimisation problems. The class of (polynomially-bounded) P-optimisation problems is an important sub-class of optimisation problems. Typical of P-optimisation problems are the maximum unit flow problem, the maximum 2-satisfiability problem, the minimum shortest-path problem and the minimum cut problem. In [16], Manyem attempted to logically capture the class of polynomially-bounded P-optimisation problems by utilizing a fragment of existential second-order logic (where the first-order part of any formula is a

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 120–131, 2010.

universally-quantified Horn formula) which is known to capture the class of decision problems P (on ordered structures; this characterization was due to Grädel [6]). As we demonstrate here, proceeding as Manyem suggests results in failure (assuming that $\mathsf{P} \neq \mathsf{NP}$), for there are polynomially-bounded optimisation problems that can be expressed within his logical framework but whose associated decision problems are NP-complete (by definition, an NP-optimisation problem is a P-optimisation problem if its associated decision problem is in P). However, we present a new framework based around inflationary fixed-point logic and where we use the inductive depth of an inflationary fixed-point as a mechanism by which to compute the values of the objective functions of the instances of our optimisation problems. We show that the classes of polynomially-bounded P-maximisation problems and polynomially-bounded P-minimisation problems can be captured within our framework.

2 A Framework for Classification

In this section, we define classes of (non-deterministic polynomial-time) optimisation problems and provide logical frameworks for the classification of such problems. These classes and frameworks come from [13,14,17,18]. Furthermore, we present some classification results from [13,14]. In addition, we refine definitions and notions from [1,16]. In particular, we define classes of (deterministic) polynomial-time optimisation problems and we explain how the logical framework presented in [1,16], together with the subsequent analysis, was somewhat incongruous.

2.1 P-Optimisation Problems

We begin by defining what we mean by a polynomial-time optimisation problem, or P-optimisation problem for short.

Definition 1. *A* maximisation problem (*resp.* minimisation problem) $\mathcal{Q}$ *is a* 4*-tuple* $(\mathcal{I}, \mathcal{F}, f, opt)$ *where*:

1. $\mathcal{I}$ *is the set of instances of* $\mathcal{Q}$*, with* $\mathcal{I}$ *recognisable in polynomial-time*;
2. $\mathcal{F}$ *is the set of feasible solutions to some instance of* $\mathcal{I}$*, where we denote by* $\mathcal{F}(I)$ *the set of feasible solutions to instance* I;
3. $f : \mathcal{I} \times \mathcal{F} \to \mathbb{N} \cup \{\perp\}$ *is the objective function, and is such that*:
   - $f(I, J) = \perp$ *if, and only if,* $J \notin \mathcal{F}(I)$;
   - *there is a polynomial* p_f *such that* $f(I, J)$ *is computable in time* $p_f(|I|)$;
4. *For any instance* $I \in \mathcal{I}$*, if* $\mathcal{F}(I)$ *is non-empty then* $opt(I) = max\{f(I, J) : J \in \mathcal{F}(I)\}$ *(resp.* $opt(I) = min\{f(I, J) : J \in \mathcal{F}(I)\}$*), and if* $\mathcal{F}(I)$ *is empty then* $opt(I) = \perp$.

The class of optimisation problems *consists of the class of maximisation problems in union with the class of minimisation problems. The maximisation* (*resp. minimisation*) *problem* $\mathcal{Q}$ *is a* P-maximisation problem (*resp.* P-minimisation problem) *if*:

5. *The problem of deciding whether a given instance $I \in \mathcal{I}$ and a given integer m are such that there exists a feasible solution $J \in \mathcal{F}(I)$ such that $f(I,J) \geq m$ (resp. $f(I,J) \leq m$) can be solved in polynomial-time.*

The class of P-optimisation problems P_{opt} *is the class of* P*-maximisation problems* P_{max} *in union with the class of* P*-minimisation problems* P_{min}*. The classes of* NP-optimisation problems NP_{opt}*,* NP-maximisation problems NP_{max}*, and* NP-minimisation problems NP_{min} *are defined analogously except that in condition 5 'non-deterministic polynomial-time' replaces 'polynomial-time'. We refer to the problem in condition 5 as the* decision version *of* $\mathcal{Q}$*.* □

Importantly, for us the *solution* of an optimisation problem $\mathcal{Q} = (\mathcal{I}, \mathcal{F}, f, opt)$ is an algorithm that given any instance I of the problem, produces as output the value $opt(I)$ and *not* (necessarily) an optimal feasible solution from $\mathcal{F}(I)$ (if there is one). In fact, this algorithm need not even work with representations of feasible solutions; all it has to do is to come up with the optimal value. Note that all problems $\mathcal{Q} = (\mathcal{I}, \mathcal{F}, f, opt)$ in P_{opt} can be solved in polynomial-time, for: given any instance I of size n and any feasible solution $J \in \mathcal{F}(I)$, by definition $f(I,J)$ is $O(2^{p(n)})$, where p is some polynomial; and repeating the algorithm in condition 5 in tandem with a binary search yields a polynomial-time algorithm that computes $opt(I)$.

Remark 1. Note that Definition 1 implies that all feasible solutions to some instance can be taken to have size bounded by some polynomial in the size of the instance. Given that our notion of a solution of an optimisation problem is such that a numeric value should be found and not a witnessing feasible solution, there is no real need to discuss the computational nature of a set of feasible solutions corresponding to some instance. In particular, Definition 1 says nothing about the complexity of deciding whether some potential feasible solution is indeed an actual feasible solution to some instance. It turns out that most (instances of) natural optimisation problems have easily recognizable sets of feasible solutions.

Remark 2. The reader will have noted that according to Definition 1, *every* optimisation problem is in fact an NP-optimisation problem, and so condition 5 is redundant when defining an NP-optimisation problem. However, we have included it as it appears in analogous definitions in [13,14]; for in these definitions the objective function is defined to be computable in time polynomial in the size of the input, i.e., the instance and a feasible solution, rather than in the size of the instance. Our notion of an optimisation problem is such that every feasible solution to some instance necessarily has size bounded by some polynomial in the size of the instance, whereas in [13,14] there is scope for considering optimisation problems whose instances have feasible solutions of size exponential in the size of the instance.

Remark 3. We should point out that Manyem's definition of a P-optimisation problem in [16], and subsequently in [1], is slightly different from that in Definition 1, for Manyem had an extra condition, namely that:

6′. The problem of computing an optimal solution for a given instance I of $\mathcal{Q}$ can be solved in time polynomial in $|I|$.

We have dropped this condition as we feel that the condition is not intrinsic to our notion of the solution of an optimisation problem. We wish our P-optimisation problems to be analogous to the NP-optimisation problems of [13,17,18] where no such condition exists. Indeed, imposing such a condition in the context of NP is somewhat problematic as not only would we be asking for a non-deterministic polynomial-time transducer but checking optimality would provide difficulties too. Dropping Manyem's additional condition provides for a more appropriate analysis. (Note that the conditions 1 – 5 and 6′, above, mirror Manyem's conditions (i) – (vi) in [16].)

2.2 Polynomially-Bounded P-Optimisation Problems

The classes defined in the following definition play an important role.

Definition 2. *An optimisation problem $\mathcal{Q}$ is* polynomially-bounded *if there is a polynomial q such that for every instance I of $\mathcal{Q}$, $opt(I) \leq q(|I|)$. We denote the class of polynomially-bounded* P*-optimisation problems by P_{opt}^{PB}, the (sub-)class of polynomially-bounded* P*-maximisation problems by P_{max}^{PB}, and the (sub-)class of polynomially-bounded* P*-minimisation problems by P_{min}^{PB}. There are analogous definitions of NP_{opt}^{PB}, NP_{max}^{PB}, and NP_{min}^{PB}.* □

We now mention some examples of optimisation problems.

Example 1. Consider the maximum 2-satisfiability problem MAX2SAT = $(\mathcal{I}, \mathcal{F}, f, opt)$, where:

- $\mathcal{I}$ is the set of conjunctive normal form formulae φ where every clause has 2 literals;
- $\mathcal{F}(\varphi)$ is the set of truth assignments on the Boolean variables involved in φ;
- $f(I, J)$, for some instance I and for some feasible solution $J \in \mathcal{F}(I)$, is the number of clauses of I made *true* under the truth assignment J.

It is well-known that the decision version of MAX2SAT is NP-complete (see, e.g., [5]). Hence, MAX2SAT is in NP_{max}^{PB} (and not in P_{max} unless P = NP). If we define MAXHORN2SAT just as was MAX2SAT except that all instances are in addition Horn formulae then thanks to a result in [10] where the decision version of MAXHORN2SAT was shown to be NP-complete, we obtain that MAXHORN2SAT is in NP_{max}^{PB} and unlikely to be in P_{max}^{PB} (in fact, the analogous problem MINHORN2SAT is also NP-complete [12]). However, if we define the problem MAX2SAT($\leq$2) by restricting instances of MAX2SAT so that every variable appears in at most 2 clauses then as the decision version of MAX2SAT($\leq$2) can be solved in linear time [19], MAX2SAT($\leq$2) is in P_{max}^{PB}.

Example 2. Consider the minimum shortest-path problem MINSP = $(\mathcal{I}, \mathcal{F}, f,$ $opt)$, where:

- $\mathcal{I}$ is the set of triples (G, s, t), with G a digraph and s and t two distinct vertices of G;
- $\mathcal{F}((G, s, t))$ is the set of all possible paths in G from s to t;
- $f(I, J)$, for some instance I and for some feasible solution $J \in \mathcal{F}(I)$, is the length of the path J.

It is well-known that the decision version of MINSP is in P (see, e.g., [2]); thus, MINSP $\in \mathsf{P}^{PB}_{min}$.

Henceforth, we define the maximum or minimum of the empty set as $\perp$.

2.3 Using Logic to Classify NP-Optimization Problems

We begin with some basic definitions. For us, a *signature* σ is a finite tuple of relation symbols $R_1, R_2, \ldots, R_r$, where each R_i has arity a_i, and constant symbols $C_1, C_2, \ldots, C_c$. A *finite structure over* σ, or σ*-structure*, $\mathcal{A}$ of size n, where $n \geq 2$, consists of a *domain*, or *universe*, $\{0, 1, \ldots, n-1\}$ and a relation R_i of arity a_i (resp. constant C_j), for every relation symbol R_i (resp. constant symbol C_j) in σ (it causes no confusion that we do not differentiate between relations and relation symbols, and constants and constant symbols). We denote both the size and the domain of a structure $\mathcal{A}$ as $|\mathcal{A}|$ (again, this causes no confusion). Let σ and τ be signatures with no symbols in common. Suppose that $\mathcal{A}$ is a σ-structure and $\mathcal{B}$ is a τ-structure with $|\mathcal{A}| = |\mathcal{B}|$. The $\sigma \cup \tau$-structure $(\mathcal{A}, \mathcal{B})$ has domain that of $\mathcal{A}$ (and $\mathcal{B}$) with relations and constants corresponding to symbols from σ (resp. τ) inherited from $\mathcal{A}$ (resp. $\mathcal{B}$). If $\tau = \langle S_1, S_2, \ldots, S_t \rangle$, where each S_i is a relation symbol, then we sometimes denote $(\mathcal{A}, \mathcal{B})$ by $(\mathcal{A}, S_1, S_2, \ldots, S_t)$. A *problem* is an isomorphism-closed set of finite structures over some fixed signature; so, a problem refers to a decision problem (as opposed to an optimisation problem). Of particular interest to us is a *successor relation*; that is, a binary relation over some domain of size n where this relation is of the form

$$\{(a_0, a_1), (a_1, a_2), \ldots, (a_{n-2}, a_{n-1}) : \text{ all } a_i\text{'s are distinct}\}.$$

We assume that the reader is familiar with using first-order logic FO and second-order logic SO to define problems.

Henceforth, all instances of some optimisation problem are finite structures $\mathcal{A}$ over some fixed signature, σ say, and we say that such an optimisation problem is *over* σ. We make no assumptions as regards the feasible solutions of some instance although in practice they tend to be structures over some (fixed) signature. Such a framework fully captures all of the optimisation problems from the previous section.

In [13], Kolaitis and Thakur characterized the classes of polynomially-bounded NP-maximization problems NP^{PB}_{max} and polynomially-bounded NP-minimization problems NP^{PB}_{min}.

Theorem 1 (Kolaitis and Thakur [13]). *Let $\mathcal{Q} = (\mathcal{I}, \mathcal{F}, f, opt)$ be a maximisation problem over σ. The following are equivalent.*

1. *$\mathcal{Q}$ is a polynomially-bounded* NP*-maximization problem, i.e., $\mathcal{Q} \in \mathsf{NP}^{PB}_{max}$.*
2. *There exists a signature τ, consisting solely of relation symbols and disjoint from σ, and a first-order formula $\varphi(\mathbf{x})$ over $\sigma \cup \tau$, where $\mathbf{x}$ is a k-tuple of variables, for some k, such that for every instance $\mathcal{A} \in \mathcal{I}$:*

$$opt(\mathcal{A}) = \max_{\mathcal{B}}\{|\{\mathbf{u} \in |\mathcal{A}|^k : (\mathcal{A}, \mathcal{B}) \models \varphi(\mathbf{u})\}|\},$$

where $\mathcal{B}$ ranges over all τ-structures of size $|\mathcal{A}|$.

Moreover, if one of the above conditions holds then the formula φ, above, can be taken to be a Π_2-formula. □

Theorem 2 (Kolaitis and Thakur [13]). *Let $\mathcal{Q} = (\mathcal{I}, \mathcal{F}, f, opt)$ be a minimisation problem over σ. The following are equivalent.*

1. *$\mathcal{Q}$ is a polynomially-bounded* NP*-minimization problem, i.e., $\mathcal{Q} \in \mathsf{NP}^{PB}_{min}$.*
2. *There exists a signature τ, consisting solely of relation symbols and disjoint from σ, and a first-order formula $\varphi(\mathbf{x})$ over $\sigma \cup \tau$, where $\mathbf{x}$ is a k-tuple of variables, for some k, such that for every instance $\mathcal{A} \in \mathcal{I}$:*

$$opt(\mathcal{A}) = \min_{\mathcal{B}}\{|\{\mathbf{u} \in |\mathcal{A}|^k : (\mathcal{A}, \mathcal{B}) \models \varphi(\mathbf{u})\}|\},$$

where $\mathcal{B}$ ranges over all τ-structures of size $|\mathcal{A}|$.

Moreover, if one of the above conditions holds then the formula φ, above, can be taken to be a Σ_2-formula. □

The class of (logically-defined) maximisation problems defined in Theorem 1 is called MAX Π_2 and the class of minimisation problems defined in Theorem 2 is called MIN Σ_2, with the notation derived from the syntax of the defining first-order formula. By imposing suitable restrictions upon the formula φ in Theorems 1 and 2, one obtains classes such as MAX Π_i, MAX Σ_i, MIN Π_i, and MIN Σ_i, for $i \geq 0$. Obviously, NP^{PB}_{max} = MAX Π_2 = MAX Π_i and NP^{PB}_{min} = MIN Σ_2 = MIN Σ_i, for all $i \geq 2$. In fact, Kolaitis and Thakur also proved the following result.

Theorem 3 (Kolaitis and Thakur [13])

- *MAX $\Sigma_0 \subset$ MAX $\Sigma_1 \subset$ MAX $\Pi_1 =$ MAX $\Sigma_2 \subset$ MAX $\Pi_2 = \mathsf{NP}^{PB}_{max}$.*
- *MIN $\Sigma_0 =$ MIN $\Sigma_1 \subset$ MIN $\Pi_1 =$ MIN $\Sigma_2 =$ MIN $\Pi_2 = \mathsf{NP}^{PB}_{min}$.* □

Kolaitis and Thakur went on in [14] to vary their logical framework slightly. They defined the following classes of optimisation problems. Note that for a relation X, we write $|X|$ to denote the number of tuples in X.

Definition 3. *Let $\mathcal{Q} = (\mathcal{I}, \mathcal{F}, f, opt)$ be a maximisation problem over σ and let $i \geq 1$. The optimisation problem $\mathcal{Q}$ is in MAX FΠ_i if, and only if, there exists a Π_i (first-order) sentence φ over $\sigma \cup \tau$, where $\tau = \langle S_1, S_2, \ldots, S_t \rangle$ and where each S_j is a relation symbol not appearing in σ, with the property that for every instance $\mathcal{A}$ of $\mathcal{I}$:*

$$opt(\mathcal{A}) = \max_{\mathcal{B}}\{|S_1| : (\mathcal{A}, \mathcal{B}) \models \varphi\},$$

where $\mathcal{B}$ ranges over all τ-structures of size $|\mathcal{A}|$. □

The classes MAX FΣ_i, MIN FΠ_i, and MIN FΣ_i, for $i \geq 1$, are defined analogously.

Kolaitis and Thakur [14] showed that these new classes of optimisation problems are closely related to the classes discussed earlier.

Theorem 4 (Kolaitis and Thakur [14])

$$\left.\begin{array}{l} MAX\ \Sigma_0 \\ MAX\ F\Sigma_1 \end{array}\right\} \subset MAX\ \Sigma_1 \subset MAX\ \Pi_1 = MAX\ F\Pi_1 = MAX\ \Sigma_2 = MAX\ F\Sigma_2 \subset MAX\ \Pi_2 = MAX\ F\Pi_2 = \mathsf{NP}^{PB}_{max}.$$

$$\left.\begin{array}{l} MIN\ \Sigma_0 = MIN\ \Sigma_1 = MIN\ F\Pi_1 \\ MIN\ F\Sigma_1 \end{array}\right\} \subset MIN\ F\Sigma_2 \subset MIN\ F\Pi_2 = MIN\ \Pi_1 = MIN\ \Sigma_2 = MIN\ \Pi_2 = \mathsf{NP}^{PB}_{min}.$$ □

The bracketing used in the statement of Theorem 4 is to denote that the classes are incomparable.

2.4 Manyem's Framework for P-Optimization Problems

Inspired by the work of Kolaitis and Thakur, in [16] Manyem (and subsequently with Bueno in [1]) attempted to provide a suitable logical framework to characterize the classes of polynomially-bounded P-maximisation problems and polynomially-bounded P-minimisation problems. Whereas Kolaitis and Thakur's logical framework had been derived from Fagin's seminal characterization of NP as the class of problems definable in existential second-order logic [4], Bueno and Manyem tried to take Grädel's characterization of P [6] as the class of problems definable in a particular fragment of existential second-order logic as their inspiration. We shall now describe Grädel's result.

We say that a quantifier-free first-order formula over $\sigma \cup \tau$, where τ consists entirely of relation symbols and is disjoint from σ, is a *Horn formula over* (σ, τ) if it is a conjunction of clauses where each clause contains at most one positive atom involving a symbol from τ. The logic $\exists$SO-Horn is the fragment of existential second-order logic consisting of all formulae over some signature σ of the form:

$$\exists S_1 \exists S_2 \ldots \exists S_t \forall y_1 \forall y_2 \ldots \forall y_m \varphi,$$

where each S_i is a relation symbol not in the underlying signature σ, each y_j is a (first-order) variable, and φ is a Horn formula over (σ, τ), where $\tau = \langle S_1, S_2, \ldots, S_t \rangle$. We say that a logic $\mathcal{L}$ *describes*, or *captures*, a class of (decision) problems $\mathcal{C}$ in the presence of a *built-in successor relation*, or *on ordered structures*, if the following are equivalent:

- The problem Ω, over the signature σ, is in $\mathcal{C}$;
- There is a sentence Φ of $\mathcal{L}$ over the signature $\sigma \cup \langle succ, min, max \rangle$, where *succ* is a binary relation symbol not in σ and *min* and *max* are constant symbols not in σ, such that:

- For every σ-structure $\mathcal{A}$, as to whether the expansion of $\mathcal{A}$ by a successor relation *succ* and constants *min* and *max*, so that *min* (resp. *max*) is the minimal (resp. maximal) element of the linear order described by *succ*, satisfies Φ is independent of the particular successor relation chosen (that is, Φ is *order invariant*);
- For every σ-structure $\mathcal{A}$, $\mathcal{A} \in \Omega$ if, and only if, the expansion of $\mathcal{A}$ by some successor relation *succ* and corresponding constants *min* and *max* satisfies Φ.

For more on built-in successor relations, we refer the reader to [3,7,9,15].

Theorem 5 (Grädel [6]). *A problem is in* P *if, and only if, it can be defined by a sentence of* $\exists$*SO-Horn in the presence of a built-in successor relation.* □

In [16], Manyem gave a definition of a P-optimisation problem and a logical framework within which to try and capture the classes of polynomially-bounded P-maximisation problems and polynomially-bounded P-minimisation problems. As we have already mentioned, his definition of a P-optimisation problem was at variance with the definition to be expected should one proceed analogously to related research on (NP) optimisation problems, mentioned above. In Theorem 3 of [16] he claims that every polynomially-bounded P-maximisation problem $\mathcal{Q} = (\mathcal{I}, \mathcal{F}, f, opt)$ over σ (according to his definition) is such that there exists a signature τ consisting of only relation symbols and disjoint from σ and a Horn formula $\varphi(\mathbf{y})$ over (σ, τ), where $\mathbf{y}$ is the tuple of free variables of φ, such that for every instance $\mathcal{A} \in \mathcal{I}$:

$$opt(\mathcal{A}) = \max_{\mathcal{B}} |\{\mathbf{u} : (\mathcal{A}, \mathcal{B}, \mathbf{u}) \models \forall x_1 \forall x_2 \ldots \forall x_k \varphi(\mathbf{y})\}|,$$

with $\mathcal{B}$ ranging over all τ-structures with domain $|\mathcal{A}|$ and $\mathbf{u}$ detailing values for the variables of $\mathbf{y}$. Manyem made a similar claim relating to polynomially-bounded P-minimisation problems in Theorem 10 of [16]. Manyem allowed for the use of a built-in successor relation in the formula φ, above, but did not explain how $\varphi(\mathbf{y})$ might be order-invariant; consequently, he left open the possibility that $|\{\mathbf{u} : (\mathcal{A}, \mathcal{B}, \mathbf{u}) \models \forall x_1 \forall x_2 \ldots \forall x_k \varphi(\mathbf{y})\}|$ might vary depending upon the particular successor relation chosen. Manyem made no claims as regards the converse direction; that is, whether optimisation problems definable in the above logical form are necessarily polynomially-bounded P-optimisation problems.

3 The Failure of Manyem's Framework

We show how any framework defined in accordance with that proposed by Manyem will not suffice to characterize the classes of polynomially-bounded P-maximisation problems and polynomially-bounded P-minimisation problems.

Theorem 6. *There exists a polynomially-bounded maximisation problem* $\mathcal{Q} = (\mathcal{I}, \mathcal{F}, f, opt)$ *such that*:

- *$\mathcal{Q}$ is over $\sigma = \langle H, Z \rangle$, where H is a relation symbol of arity 4 and Z is a constant symbol, with $\mathcal{I}$ the set of σ-structures*;
- *$\tau = \langle P, T \rangle$, where P and T are both relation symbols of arity 1, so that the set of feasible solutions $\mathcal{F}(\mathcal{A})$ to some instance $\mathcal{A} \in \mathcal{I}$ is the set of τ-structures with domain $|\mathcal{A}|$*;
- *φ is a Horn formula over (σ, τ) with free variables x_1, x_2, x_3, y, so that given some instance $\mathcal{A} \in \mathcal{I}$ and some feasible solution $\mathcal{B} \in \mathcal{F}(\mathcal{I})$,*

$$f(\mathcal{A}, \mathcal{B}) = |\{u : (\mathcal{A}, \mathcal{B}, u) \models \forall x_1 \forall x_2 \forall x_3 \varphi(y)\}|;$$

- *The decision version of $\mathcal{Q}$ is* NP*-complete.*

Theorem 7. *There exists a polynomially-bounded minimisation problem $\mathcal{Q} = (\mathcal{I}, \mathcal{F}, f, opt)$ such that*:

- *$\mathcal{Q}$ is over $\sigma = \langle H, Z \rangle$, where H is a relation symbol of arity 4 and Z is a constant symbol, with $\mathcal{I}$ the set of σ-structures*;
- *$\tau = \langle P, T \rangle$, where P and T are both relation symbols of arity 1, so that the set of feasible solutions $\mathcal{F}(\mathcal{A})$ to some instance $\mathcal{A} \in \mathcal{I}$ is the set of τ-structures with domain $|\mathcal{A}|$*;
- *φ is a Horn formula over (σ, τ) with free variables x_1, x_2, x_3, y, so that given some instance $\mathcal{A} \in \mathcal{I}$ and some feasible solution $\mathcal{B} \in \mathcal{F}(\mathcal{I})$,*

$$f(\mathcal{A}, \mathcal{B}) = |\{u : (\mathcal{A}, \mathcal{B}, u) \models \forall x_1 \forall x_2 \forall x_3 \varphi(y)\}|;$$

- *The decision version of $\mathcal{Q}$ is* NP*-complete.*

An immediate consequence of Theorems 6 and 7 is that any framework based around Grädel's characterisation of P using restricted (Horn) formulae of existential second-order logic (as advocated by Manyem) will not characterize the class of polynomially-bounded P-maximisation problems nor the class of polynomially-bounded P-minimisation problems (assuming that $\mathsf{P} \neq \mathsf{NP}$).

Note also that when working with polynomially-bounded P-minimisation problems, there is a pronounced difference between the original framework proposed by Kolaitis and Thakur [13], where in order to obtain the objective function value we count the number of elements satisfying some formula, and the amended one [14], where we count the cardinality of a witnessing relation (Manyem chose to adopt the former framework when he strove for a logical classification of P-optimisation problems in [16]). Whilst MIN F$\Pi_1 \subset$ MINΠ_1, with MIN FΠ_1 still containing NP-hard optimisation problems (like VERTEX COVER), if we have some optimisation problem $\mathcal{Q} = (\mathcal{I}, \mathcal{F}, f, opt)$, over σ, where for every instance $\mathcal{A} \in \mathcal{I}$ for which $\mathcal{F}(\mathcal{A})$ is non-empty,

$$opt(\mathcal{A}) = \max_{\mathcal{B}} \{|B_0| : (\mathcal{A}, \mathcal{B}) \models \forall x_1 \forall x_2 \ldots \forall x_k \varphi\},$$

with φ a Horn sentence over (σ, τ), $\mathcal{B}$ ranging over all τ-structures with domain $|\mathcal{A}|$ and B_0 a specific relation from $\mathcal{B}$, then $\mathcal{Q}$ is indeed a polynomially-bounded

P-optimisation problem. To see this, simply 'expand' the sentence φ in any instance $\mathcal{A}$ in order to obtain a collection of Horn formulae. If there exists a witnessing set of relations $\mathcal{B}$ then there is a unique 'minimal' set of relations $\mathcal{B}$, computable in polynomial-time. The cardinality of the relation B_0 from this set of relations $\mathcal{B}$ yields the value $opt(\mathcal{A})$. Thus, if we were to adopt the amended framework from [14] and Manyem's approach to obtaining a logical characterization of polynomially-bounded P-minimisation problems then it is feasible that we might be able to do so (though we would have to ensure that all defining formulae were order-invariant, as we explained at the end of Section 2.4). Even this revised framework would fail for polynomially-bounded P-maximisation problems, though, as is demonstrated by the proof of Theorem 6 (as always, we assume that $\mathsf{P} \neq \mathsf{NP}$).

4 Logically Capturing P^{PB}_{max} and P^{PB}_{min}

In this section we provide logical characterizations of the classes of polynomially-bounded P-maximisation problems and polynomially-bounded P-minimisation problems. The logic we use is not a fragment of existential second-order logic but the well-known inflationary fixed-point logic FO(IFP). We use the inductive process of building fixed-points in order to provide values for the objective functions of our optimisation problems. The reader is referred to any of [3,7,9,15] for details as regards FO(IFP). Pertinent to this paper is the following result where we denote the logic FO(IFP) in the presence of a built-in successor relation by $\mathrm{FO}_s(\mathrm{IFP})$.

Theorem 8 (Immerman [8], Vardi [20]). *A problem is in* P *if, and only if, it can be defined by a sentence of $FO_s(IFP)$. Moreover, any sentence of $FO_s(IFP)$ is logically equivalent to one of the form $[IFP_{R,\mathbf{x}}\varphi](\mathbf{max})$, where φ is quantifier-free first-order and* **max** *is a tuple every component of which is the constant symbol max.*

Let φ be a formula of $\mathrm{FO}_s(\mathrm{IFP})$ over the signature $\sigma \cup \langle R \rangle$, where R is a k-ary relation symbol, so that the free variables of φ are those of the k-tuple of variables $\mathbf{x}$. We write $depth_{(\mathcal{A},\mathbf{u})}([\mathrm{IFP}_{R,\mathbf{x}}\varphi])$ to denote the inductive depth of the inflationary fixed-point of $\varphi(R,\mathbf{x})$ in $(\mathcal{A},\mathbf{u})$. We say that $\varphi(R,\mathbf{x})$ is *depth-invariant* if the inductive depth of the inflationary fixed-point of $\varphi(R,\mathbf{x})$ in any σ-structure is independent of the actual underlying successor relation.

Theorem 9. *Let $\mathcal{Q} = (\mathcal{I},\mathcal{F},f,opt)$ be a maximisation problem over σ. The following are equivalent.*

1. *$\mathcal{Q}$ is a polynomially-bounded* P*-maximisation problem (i.e., $\mathcal{Q} \in \mathsf{P}^{PB}_{max}$).*
2. *There exists some depth-invariant formula $\varphi(R,\mathbf{x})$ of $FO_s(IFP)$ over $\sigma \cup \langle R \rangle$, where R is a k-ary relation symbol and the free variables of φ are those of the k-tuple $\mathbf{x}$, such that for any $\mathcal{A} \in \mathcal{I}$:*
   - *if $\mathcal{F}(\mathcal{A})$ is non-empty then $\mathcal{A} \models [IFP_{R,\mathbf{x}}\varphi](\mathbf{min})$ and the optimal value $opt(\mathcal{A})$ is given by $depth_{\mathcal{A}}([IFP_{R,\mathbf{x}}\varphi]) - 1$;*
   - *If $\mathcal{F}(\mathcal{A})$ is empty then $\mathcal{A} \not\models [IFP_{R,\mathbf{x}}\varphi](\mathbf{min})$.*

Theorem 10. *Let $\mathcal{Q} = (\mathcal{I}, \mathcal{F}, f, opt)$ be a minimisation problem over σ. The following are equivalent.*

1. *$\mathcal{Q}$ is a polynomially-bounded P-minimisation problem (i.e., $\mathcal{Q} \in \mathsf{P}^{PB}_{min}$).*
2. *There exists some depth-invariant formula $\varphi(R, \mathbf{x})$ of $FO_s(IFP)$ over $\sigma \cup \langle R \rangle$, where R is a k-ary relation symbol and the free variables of φ are those of the k-tuple $\mathbf{x}$, such that for any $\mathcal{A} \in \mathcal{I}$:*
   - *if $\mathcal{F}(\mathcal{A})$ is non-empty then $\mathcal{A} \models [IFP_{R,\mathbf{x}}\varphi](\mathbf{min})$ and the optimal value $opt(\mathcal{A})$ is given by $|\mathcal{A}|^k - depth_{\mathcal{A}}([IFP_{R,\mathbf{x}}\varphi]) + 1$;*
   - *if $\mathcal{F}(\mathcal{A})$ is empty then $\mathcal{A} \not\models [IFP_{R,\mathbf{x}}\varphi](\mathbf{min})$.*

Note that the actual numeric formulae giving the value of an optimal solution in Theorems 9 and 10 in terms of the depth of a fixed-point construction are to some extent unimportant. All that matters is that they are efficiently computable, which both formulae are. In consequence, we obtain logical characterizations of the classes P^{PB}_{max} and P^{PB}_{min}.

5 Conclusions

In this paper we have clarified the applicability of logical frameworks in relation to capturing classes of polynomially-bounded NP-optimisation problems. We have seen: that Manyem's framework does not (and will not) suffice; that there are additional differences between the two frameworks proposed by Kolaitis and Thakur when one restricts so as to consider P-optimisation problems; and that there does exist an alternative logical framework capturing polynomially-bounded P-optimisation problems.

We suggest the following as directions for further research. Whilst Manyem's attempt to capture classes of polynomial-time optimisation problems using fragments of existential second-order logic with the first-order quantifier-free part restricted to be a conjunction of Horn clauses has gone awry, it would be interesting to continue this investigation in relation to the hierarchy results from Theorem 4. That is, what happens at the lower end of this hierarchy when we restrict the first-order quantifier-free part of formulae (with or without successor) to be a conjunction of Horn clauses, or even Krom clauses (a Krom clause is a clause with exactly 2 literals)?

Finally, there is no doubt that polynomial-time optimisation problems are not as abundant as NP-optimisation problems, nor do they straddle the P versus NP divide as do NP-optimisation problems. Nevertheless, a more wide-ranging investigation as to the relationship between, for example, P-optimisation problems and optimisation problems that can be solved in NC or NL and into alternative means of logically defining P-optimisation problems is warranted.

Acknowledgement. The authors are indebted to Prabhu Manyem for many clarifying conversations as regards his work, undertaken whilst he spent a sabbatical stay in Durham.

References

1. Bueno, O., Manyem, P.: Polynomial-time maximisation classes: syntactic hierarchy. Fundamenta Informaticae 84(1), 111–133 (2008)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1990)
3. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory. In: Monographs in Mathematics. Springer, Heidelberg (1999)
4. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. In: Complexity and Computation, SIAM-AMS Proceedings, vol. 7, pp. 43–73 (1974)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York (1979)
6. Grädel, E.: Capturing complexity classes by fragments of second-order logic. Theoretical Computer Science 101(1), 35–57 (1992)
7. Grädel, E., Kolaitis, P.G., Libkin, L., Marx, M., Spencer, J., Vardi, M.Y., Venema, Y., Weinstein, S.: Finite Model Theory and Its Applications. In: Texts in Theoretical Computer Science. Springer, Heidelberg (2007)
8. Immerman, N.: Relational queries computable in polynomial time. Information and Control 68(1-3), 86–104 (1986)
9. Immerman, N.: Descriptive Complexity. In: Graduate Texts in Computer Science. Springer, Heidelberg (1999)
10. Jaumard, B., Simeone, B.: On the complexity of the maximum satisfiability problem for horn formulas. Information Processing Letters 26(1), 1–4 (1987)
11. Johnson, D.: Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences 9(3), 256–278 (1974)
12. Kohli, R., Krishnamurti, R., Mirchandani, P.: The minimum satisfiability problem. SIAM Journal of Discrete Mathematics 7(2), 275–283 (1994)
13. Kolaitis, P.G., Thakur, M.N.: Logical definability of NP optimization problems. Information and Computation 115(2), 321–353 (1994)
14. Kolaitis, P.G., Thakur, M.N.: Approximation properties of NP minimization classes. Journal of Computer and System Sciences 50(3), 391–411 (1995)
15. Libkin, L.: Elements of Finite Model Theory. In: Texts in Theoretical Computer Science. Springer, Heidelberg (2004)
16. Manyem, P.: Syntactic characterizations of polynomial time optimization classes. Chicago Journal of Theoretical Computer Science (3), 1–23 (2008)
17. Panconesi, A., Ranjan, D.: Quantifiers and approximation. Theoretical Computer Science 107(1), 145–163 (1993)
18. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. Journal of Computer and System Sciences 43(3), 425–440 (1991)
19. Raman, V., Ravikumar, B., Srinivasa Rao, S.: A simplified NP-complete MAXSAT problem. Information Processing Letters 65(1), 163–168 (1998)
20. Vardi, M.Y.: The complexity of relational query languages. In: Proceedings of 14th ACM Ann. Symp. on the Theory of Computing, pp. 137–146 (1982)

Validating the Knuth-Morris-Pratt Failure Function, Fast and Online

Paweł Gawrychowski, Artur Jeż, and Łukasz Jeż*

Institute of Computer Science, University of Wrocław
{gawry,aje,lje}@cs.uni.wroc.pl

Abstract. Let π'_w denote the failure function of the Knuth-Morris-Pratt algorithm for a word w. In this paper we study the following problem: given an integer array $A'[1\,..\,n]$, is there a word w over an arbitrary alphabet Σ such that $A'[i] = \pi'_w[i]$ for all i? Moreover, what is the minimum cardinality of Σ required? We give an elementary and self-contained $\mathcal{O}(n \log n)$ time algorithm for this problem, thus improving the previously known solution [8] with no polynomial time bound. Using both deeper combinatorial insight into the structure of π' and more advanced tools, we further improve the running time to $\mathcal{O}(n)$.

1 Introduction

Failure functions. The Morris-Pratt algorithm [19], first linear time pattern matching algorithm, is well known for its beautiful concept. It simulates the minimal DFA recognizing $\Sigma^* p$ (p denotes the pattern) by using a *failure function* π, known as the *border array*. The automaton's transitions are recovered, in amortized constant time, from the values of π for all prefixes of the pattern, to which the DFA's states correspond. The values of π are precomputed in a similar fashion, also in linear time.

The MP algorithm has many variants. For instance, the Knuth-Morris-Pratt algorithm [16] improves it by using an optimised failure function, namely the *strict border array* π' (or *strong failure function*). This was improved by Simon [21], and further improvements are known [13,1]. We focus on the KMP failure function for two reasons. Unlike later algorithms, it is well-known and used in practice. Furthermore, the strong border array itself is of interest as, for instance, it captures all the information about periodicity of the word. Hence it is often used in word combinatorics and numerous text algorithms, see [4,5]. On the other hand, even Simon's algorithm (i.e., the very first improvement) deals with periods of pattern prefixes augmented by a single text symbol rather than pure periods of pattern prefixes.

Problem statement. We investigate the following problem: given an integer array $A'[1\,..\,n]$, is there a word w over an arbitrary alphabet Σ s.t. $A'[i] =$

* Supported by MNiSW grants number N N206 1723 33, 2007–2010 and N N206 4906 38 2010–2011.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 132–143, 2010.

$\pi'_w[i]$ for all i, where π'_w denotes the failure function of the Knuth-Morris-Pratt algorithm for w. If so, what is the minimum cardinality of the alphabet Σ over which such a word exists?

Pursuing these questions is motivated by the fact that in word combinatorics one is often interested only in values of π'_w rather than w itself. For instance, the logarithmic upper bound on delay of KMP follows from properties of the strict border array [16]. Thus it makes sense to ask if there is a word w admitting $\pi'_w = A'$ for a given array A'.

We are interested in an *online* algorithm, i.e., one that receives the input array values one by one, and is required to output the answer after reading each single value.

Previous results. To our best knowledge, this problem was investigated only for a slightly different variant of π', namely a function g that can be expressed as $g[n] = \pi'[n-1] + 1$, for which an offline validation algorithm is known [8]. Unfortunately, Duval et al. [8] provided no upper bound on the running time of their algorithm, but they did observe that on certain input arrays it runs in $\Omega(n^2)$ time.

Our results. We give a simple $\mathcal{O}(n \log n)$ online algorithm for strong border array validation, which uses the linear offline bijective transformation between π and π'. Our algorithm is also applicable to g validation with no changes, thus giving the first provably polynomial algorithm for the problem considered by Duval et al. [8]. Note that aforementioned bijection between π and π' cannot be applied to g considered by Duval et al. [8], as it essentially uses the unavailable value $\pi[n] = \pi'[n]$.

Then we improve this construction to an optimal linear online algorithm VALIDATE-π'. The improved algorithm heavily relies on both more sophisticated data structures, such as dynamic suffix trees supporting LCA queries, and deeper insight into the combinatorial properties of π' function.

Related results. The study of validating arrays related to string algorithms and word combinatorics was started by Franěk et al. [11], who gave an offline algorithm for border array validation. This result was improved over time, in particular a simple linear online algorithm for π validation is known [9].

The border array validation problem was also studied in the more general setting of *parametrised border array* validation [14], where parametrised border array is a border array for text in which a permutation of letters of alphabet is allowed. A linear time algorithm for a restricted variant of this problem is known, with the general case still not settled [14].

Recently a linear online algorithm for a closely related *prefix array* validation was given [2].

Validation of border arrays is used by algorithms generating all valid border arrays [7,11,18].

2 Preliminaries

For $w \in \Sigma^*$, we denote its length by $|w|$ or n, when w is clear from the context. For $v, w \in \Sigma^*$, by vw we denote the concatenation of v and w. We say that u is a *prefix* of w if there is $v \in \Sigma^*$ such that $w = uv$. Similarly, we call v a *suffix* of w if there is $u \in \Sigma^*$ such that $w = uv$. A word v that is both a prefix and a suffix of w is called a *border* of w. By $w[i]$ we denote the i-th letter of w and by $w[i\,..\,j]$ we denote the *subword* $w[i]w[i+1]\dots w[j]$ of w. We call a prefix (respectively: suffix, border) v of the word w *proper* if $v \neq w$, i.e., it is shorter than w itself.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$w[i]$	a	a	b	a	a	b	a	a	a	b	a	a	b	a	a	c
$\pi_w[i]$	0	1	0	1	2	3	4	5	2	3	4	5	6	7	8	0
$\pi'_w[i]$	−1	1	−1	−1	1	−1	−1	5	1	−1	−1	1	−1	−1	8	0

Fig. 1. Functions π and π' for *aabaabaaabaac*

For a word w its *failure function* π_w is defined as follows: $\pi_w[i]$ is the length of the longest proper border of $w[1\,..\,i]$ for $i = 1, 2 \dots, n$. By $\pi_w^{(k)}$ we denote the k-fold composition of π_w with itself, i.e., $\pi_w^{(0)}[i] := i$ and $\pi_w^{(k+1)}[i] := \pi_w[\pi_w^{(k)}[i]]$. This convention applies to other functions as well. We omit the subscript w in π_w, whenever it is unambiguous. Note that every border of $w[1\,..\,i]$ has length $\pi_w^{(k)}[i]$ for some integer $k \geq 0$.

The *strong failure function* π' is defined as follows: $\pi'_w[n] := \pi_w[n]$, and for $i < n$, $\pi'[i]$ is the length of the longest (proper) border of $w[1\,..\,i]$ such that $w[\pi'_w[i]+1] \neq w[i+1]$. If no such border exists, $\pi'[i] = -1$.

Below we present a bijection between π_w and π'_w. Values of this function, as well as its inverse, can be computed in linear time. The correctness as well as the procedure itself are a consequence of the following observation[1].

$$w[i+1] = w[\pi[i]+1] \iff \pi[i+1] = \pi[i]+1 \iff \pi'[i] < \pi[i] \iff \pi'[i] = \pi'[\pi[i]]. \quad (1)$$

COMPUTE-π'-FROM-$\pi(\pi)$

```
π'[0] ← −1
π'[n] ← π[n]
for i ← 1 to n − 1 do
    if π[i + 1] = π[i] + 1 then
        π'[i] ← π'[π[i]]
    else π'[i] ← π[i]
```

COMPUTE-π-FROM-$\pi'(\pi')$

```
π[n] ← π'[n]
for i ← n − 1 downto 1 do
    π[i] ← max{π'[i], π[i + 1] − 1}
```

[1] While we believe that (1) and both transformations are folklore, we did not find them explicitly stated anywhere, though.

3 Online Strict Border Array Validation

Since there is a bijection between valid border arrays and strict border arrays, it is natural to proceed as follows. Assume the input forms a valid strict border array, obtain the corresponding border array, and validate the result using the known border array validation algorithm. Unfortunately, the bijection in question starts the calculations from the last entry of the array, so it is not suitable for an online algorithm. We refine this approach, by noting that while there can be many border arrays consistent with $A'[1\,..\,i]$, they differ only on a certain suffix. Our algorithm identifies this suffix and finds the maximal (in a sense explained below) function A consistent with A'. Then it validates the fixed prefix of A (as a border array) using the known linear-time online algorithm by Duval et al. [9].

Consistent functions. We say that $A[1\,..\,n+1]$ is *consistent* with $A'[1\,..\,n]$ iff there is a word $w[1\,..\,n+1]$ such that

(i1) $A[1\,..\,n+1] = \pi_w[1\,..\,n+1]$,
(i2) $A'[1\,..\,n] = \pi'_w[1\,..\,n]$.

Among functions consistent with A' there exists the maximal one, i.e., $A[1\,..\,n+1]$ such that

(i3) every $B[1\,..\,n+1]$ consistent with $A'[1\,..\,n]$ satisfies $B[1\,..\,n+1] \le A[1\,..\,n+1]$, where $A[1\,..\,m] \ge B[1\,..\,m]$ denotes that $A[j] \ge B[j]$ for $j = 1, \ldots, m$.

We call such A the *maximal function consistent with* A'. Our algorithm VALIDATE-π' maintains such A, proving its existence.

Slopes and their properties. Imagine the array A' as the set of points $(i, A'[i])$. Such a picture helps in understanding the idea behind the algorithm. In this setting we think of A as a collection of maximal *slopes*: a set of indices $i, i+1, \ldots, i+j$ is a slope if $A[i+k] = A[i] + k$ for $k = 1, \ldots, j$. Note that $A[i+j+1] \neq A[i+j]+1$ implies that $A[i+j] = A'[i+j]$, by (1). Let *the A-pin* be the first position on the last slope of A. We abbreviate it to the pin, if A is clear from the context. It turns out that all functions consistent with A' differ with A only on the *last slope*.

Lemma 1. *Let* $A[1\,..\,n+1] \ge B[1\,..\,n+1]$ *be both consistent with* $A'[1\,..\,n]$. *Let* i *be the A-pin. Then* $A[1\,..\,i-1] = B[1\,..\,i-1]$.

Algorithm's overview. Our algorithm, VALIDATE-π', maintains the maximal function A consistent with A' and the A-pin i. When a new value $A'[n]$ is read, these are updated as follows. If A is no longer consistent with A', the last slope of A is adjusted. This consists in setting $A[i]$ to its next maximum candidate value. If there is none, A' is invalid. Otherwise it may happen that the A-pin is invalid, i.e., that there is $j \ge i$ such that $A[j] \le A'[j]$. Again, A' is invalid if $A[j] < A'[j]$, and if $A[j] = A'[j]$, the last slope is broken into two slopes, one ending at j and the other starting at $j+1$, which is the new pin.

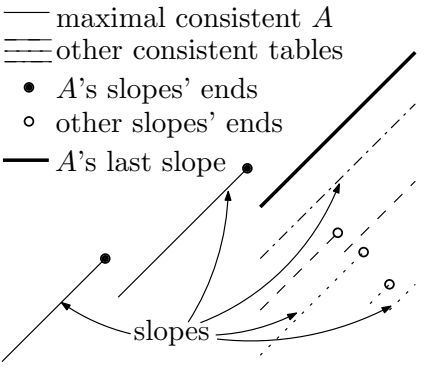


Fig. 2. Graphical illustration of slopes and maximal consistent function

Two condition are checked to decide if adjustment is needed: whether $A'[j] < A[j]$ for all $j \geq i$, and whether $A'[i\,..\,n] = A'[A[i]\,..\,A[i] + (n-i)]$.

Unfortunately, this simple combinatorial idea alone fails to produce a linear algorithm. The problem is caused by the second check: large segments of A' should be compared in amortised constant time. We exploit LCA queries on suffix trees for this task. The known online suffix tree construction algorithms [17,22] are linear only for constant size-alphabets and the only linear-time algorithm for non-constant alphabets [10] is inherently offline and thus infeasible for our purposes. To overcome this obstacle we specialise the data structures used, building the suffix tree for compressed encoding of A' and multiple suffix trees for short texts over polylogarithmic alphabet.

Data maintained. VALIDATE-π' stores:

- n, the number of values read so far,
- $A'[1\,..\,n]$, the input read so far,
- i, the A-pin,
- the maximal $A[1\,..\,n+1]$ consistent with $A'[1\,..\,n]$:
  - $A[1\,..\,i-1]$, the fixed prefix,
  - $A[i]$, the candidate value that may change.

Note that $A[i+j]$ for $j = 1, \ldots, n-i+1$ are not stored. These values are implicit, given by $A[i+j] = A[i] + j$.

Validating A. VALIDATE-π' creates a border array A, which is always valid by the construction. We run the linear-time online algorithm VALIDATE-π [9] for border array validation on A. For a valid border array $A[1\,..\,n]$ it computes all valid π-candidates for $A[n+1]$, i.e., $A[1\,..\,n+1]$ is a valid border array if and only if $A[n+1]$ is in the precomputed set.

VALIDATE-π' runs VALIDATE-π on the fixed prefix of $A[1\,..\,n]$, i.e., $A[1\,..\,i-1]$ if $A[i] > 0$, or $A[1\,..\,i]$ if $A[i] = 0$. This way the set of valid candidates for $\pi[i]$ is computed, as well as the minimum size of the alphabet Σ required by A, and the word w over Σ that admits A.

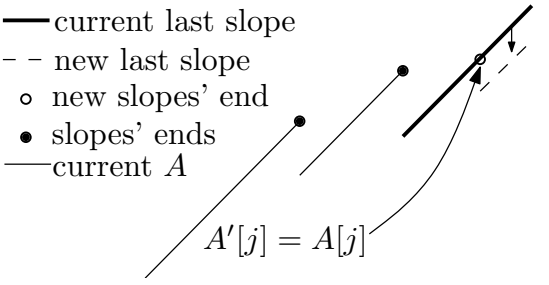


Fig. 3. Splitting the last slope

We note that the minimum alphabet size required by the fixed prefix of A matches the minimum alphabet size required by A'.

Lemma 2. *Let $A'[1\,..\,n]$ be a valid π' function, $A[1\,..\,n+1]$ be the maximal function consistent with $A'[1\,..\,n]$, and i be the A-pin. The minimum alphabet size required by $A'[1\,..\,n]$ equals the minimum alphabet size required by $A[1\,..\,i-1]$ if $A[i] > 0$, and by $A[1\,..\,i]$ if $A[i] = 0$.*

Checking the last slope. When the next value $A'[n]$ is read, and $A'[n] \neq A'[A[n]]$, then A is no longer consistent with A' or i should be changed. Hence, we adjust A on the current last slope until the following conditions hold.

$$A'[j] < A[j], \quad \text{for each } j \in [i\,..\,n]\ , \tag{2}$$

$$A'[j] = A'[A[j]], \quad \text{for each } j \in [i\,..\,n]\ . \tag{3}$$

The A-pin i is valid if and only if (2) holds, while the values of A and A' on the last slope are consistent if and only if (3) holds.

These conditions are checked by appropriate queries: (2) by the *pin check* (denoted Pin-Check), which returns any $j \in [i\,..\,n]$ such that $A'[j] > A[j]$ or, if there is no such j, the smallest $j \in [i\,..\,n]$ such that $A'[j] = A[j]$, and (3) by the *consistency check* (denoted Consistency), which checks whether $A'[i\,..\,n] = A'[A[i]\,..\,A[i]+(n-i)]$.

When a new input value $A'[n]$ is read, the last slope is updated as explained below until both (2) and (3) hold, i.e., until Pin-Check returns no index and Consistency returns **true**.

```
Validate-π'(A')
    A[1] ← 0, i ← 0, n ← 1, A'[0] ← −1
    while true do
        change ← false
        n ← n + 1
        if A'[n] ≠ A'[A[n]] then
                change ← true
        while change do
                change ← false
                j ← Pin-Check
                if j is defined then
                   run Check-slope
                if not Consistencythen
                   change ← true
                if change then
                   if A[i] = 0 then
                           A' is not valid at n
                   A[i] ← next candidate
```

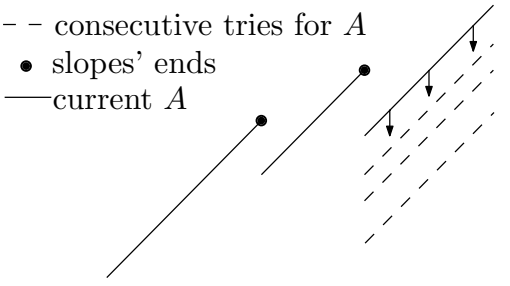


Fig. 4. Decreasing the $A[i]$

Adjusting the last slope. If the pin check returns an index j such that $A'[j] > A[j]$, then we reject the input and report an error. If $A'[j] = A[j]$ then we check (naively) whether $A'[i\,..\,j-1] = A'[A[i]\,..\,A[i]+(j-i-1)]$ holds. If it does not hold, we reject. If it does, we break the last slope in two: $[i\,..\,j]$ and $[j+1\,..\,n]$, the new last slope, see Fig 3. This consists in storing values $A[i\,..\,j]$, setting i to $j+1$, and setting $A[i]$ to the largest valid candidate value for $\pi[i]$.

```
CHECK-SLOPE
  if A'[j] > A[i − 1] + (j − i + 1) then
      error A' is not valid at n
  if A'[j] = A[j] then
      if A'[i..j − 1] ≠ A'[A[i]..A[j − 1]] then
            error A' is not valid at n
      run VALIDATE-π(A) on positions i..j
      if A[j + 1] = 0 then
            run VALIDATE-π(A) on j + 1
      for m ← i + 1 to j do
            store A[m] ← A[m − 1] + 1
      i ← j + 1
      change ← TRUE
```

If CONSISTENCY check fails, then we set the value of $A[i]$ to the next valid candidate value for $\pi[i]$, see Fig. 4. If $A[i] = 0$, then there is no candidate value and A' is rejected.

Note, that as the values $A[i+1\,..\,n+1]$ are implicit, decreasing the value of $A[i]$ decreases all the values of $A[i+1\,..\,n+1]$, see Fig. 4.

Theorem 1. VALIDATE-π' *answers if A' is a valid strict border array. If so, it supplies the maximal function A consistent with A'. Furthermore $A = \pi_w$ for a word w over an alphabet of minimum required size.*

We further note that Lemma 2 implies that the minimum size of the alphabet required for a valid strict border array is at most as large as the one required for border array. The latter is known to be $\mathcal{O}(\log n)$ [18, Th. 3.3a]. These two observations imply the following.

Corollary 1. *The minimum size of the alphabet required for a valid strict border array is $\mathcal{O}(\log n)$.*

Performing pin checks. Consider the PIN-CHECK. Note that if $A'[j'] - A'[j] > j' - j > 0$ and j is an answer to PIN-CHECK, so is j'. This observation allows to keep a collection $j_1 < j_2 < \cdots < j_\ell$ of indices such that the answer to the query is always either j_1 or **false**. Updates of this collection are done by either removal from the beginning of the list, when i becomes larger than j_1, or by consecutive removals from the end of the list, when a new $A'[n]$ is read.

Lemma 3. *The total time of answering all the pin checks is linear.*

Performing consistency checks. We need to efficiently perform two operations: appending a letter to the current text $A'[1\,..\,n]$ and checking if two fragments of the prefix read so far are the same. First we show how to implement both of them using randomisation so that the expected running time is $\mathcal{O}(\log n)$. In the next section we improve the running time to deterministic $\mathcal{O}(1)$.

We use the standard labeling technique [15], assigning unique small names to all fragments of length that are powers of two. More formally, let $name[i][j]$ be an integer from $\{1, \ldots, n\}$ such that $name[i][j] = name[i'][j]$ if and only if $A'[i..i + 2^j - 1] = A'[i'..i' + 2^j - 1]$. Then checking if any two fragments of A' are the same is easy: we only need to cover both of them with fragments which are of the same length 2^j, where 2^j is the largest power of two not exceeding their length. Then we check if the corresponding fragments of length 2^j are the same in constant time using the previously assigned names.

Appending a new letter $A'[n + 1]$ is more difficult, as we need to compute $name[n - 2^j + 2][j]$ for all j. We set $name[n + 1][0]$ to $A'[n + 1]$. Computing other names is more complicated: we need to check if a given fragment of text $A'[n - 2^j + 2...n + 1]$ occurs at some earlier position, and if so, choose the same name. To locate the previous occurences, for all $j > 0$ we keep a dictionary $M(j)$ mapping pair $(name[i][j-1], name[i + 2^{j-1}][j-1])$ to $name[i][j]$. As soon as we choose the value of $name[i][j]$, we insert a new element into $M(j)$. To check if a given fragment $A'[n - 2^j + 2...n + 1]$ occurs previously in the text, we look up the pair $(name[n - 2^j + 2][j-1], name[n - 2^{j-1} + 2][j-1])$ in $M(j)$. If there is such an element in $M(j)$, we set $name[n - 2^j + 2][j]$ equal to the corresponding name. Otherwise we set $name[n - 2^j + 2][j]$ equal to the size of $M(j)$ plus 1 (or, in other words, the smallest integer which we have not assigned as a name of fragment of length 2^j yet).

To implement the dictionaries $M(j)$, we use dynamic hashing with a worst-case constant time lookup and amortized expected constant time for updates (see [6] or a simpler variant with the same performance bounds [20]). Then the running time of the whole algorithm becomes expected $\mathcal{O}(n \log n)$, where the expectation is taken over the random choices of the algorithm.

4 Improving the Running Time to Linear

To improve the running time we only need to show how to perform consistency checks more efficiently. A natural approach is as follows: construct a suffix tree

on-line [17,22] for the input table $A'[1\,..\,n]$, together with a data structure for answering LCA queries [3]. However, A' is a text over an alphabet $\{-1, 0, \ldots, n-1\}$, i.e., of size n, so the online suffix tree construction takes $\mathcal{O}(n \log n)$ time. To get a linear time algorithm we exploit both the structure of the π' array and the relationship between subsequent consistency checks. First we note that a suffix tree for text over a polylogarithmic alphabet can be constructed in linear time.

Lemma 4. *For any constant c, the suffix tree for a text of length n over an alphabet of size $\log^c n$ can be constructed on-line in $\mathcal{O}(n)$ time. Given a vertex in the resulting tree, its child labeled by a specified letter can be retrieved in constant time.*

Proof (outline). It is enough to modify Ukkonen's algorithm [22] so that retrieval of the child of a given vertex labeled with a specified letter takes constant time. For that we can use the atomic heaps of Fredman and Willard [12], which allow constant time search and insert operations on a collection of $\mathcal{O}(\sqrt{\log n})$-elements sets. This results in a fairly complicated structure, which can be greatly simplified since in our case not only are the sets small, but the size of the universe is bounded as well.

To this end we develop a simple succinct data structure similar to a B-tree of order $\sqrt{\log n}$ and constant depth. Similarly to atomic heaps, both search and insert operation take constant time. □

Compressing A'. Lemma 4 does not apply to A', as it may hold too many different values. To overcome this obstacle we compress A' into $\mathit{Compress}(A')$, so that the resulting text is over a polylogarithmic alphabet and checking equality of two fragments of A' can be performed by looking at the corresponding fragments of $\mathit{Compress}(A')$. To compress A', we scan it from left to right. If $A'[i] = A'[i-j]$ for some $1 \le j \le \log^2 n$ we output $\#_0 j$. If $A'[i] \le \log^2 n$ we output $\#_1 A'[i]$. Otherwise output $\#_2$ and the binary encoding of $A'[i]$ followed by $\#_3$. For each i we store the position of its encoding in $\mathit{Compress}(A')$ in $\mathit{Start}[i]$.

We show that the number of different large values of π' is small, which allows bounding the size of $\mathit{Compress}(A')$ by $\mathcal{O}(n)$.

Lemma 5. *Let $k \ge 0$ and consider a segment of 2^k consecutive entries in the π' array. At most 48 different values from the interval $[2^k, 2^{k+1})$ occur in such a segment.*

Corollary 2. *$\mathit{Compress}(A')$ consists of $\mathcal{O}(n)$ symbols over an alphabet of $\mathcal{O}(\log^2 n)$ size.*

As the alphabet of $\mathit{Compress}(A')$ is of polylogarithmic size, the suffix tree for $\mathit{Compress}(A')$ can be constructed in linear time by Lemma 4.

Subchecks. Consider Consistency check: is $A'[j\,..\,j+k] = A'[i\,..\,i+k]$, where $j = A[i]$? We first establish equivalence of this equality with equality of proper fragments of $\mathit{Compress}(A')$. Note, that $A'[\ell] = A'[\ell']$ does not imply the equality

of two corresponding fragments of $Compress(A')$, as they may refer to other values of A'. Still, such references can be only $\log^2 n$ elements backwards. This observation is formalised as:

Lemma 6. *Let $j = A[i]$. Then*

$$A'[j\,..\,j+k] = A'[i\,..\,i+k]$$

if and only if

$$Compress(A')[Start[j+\log^2 n]\,..\,Start[j+k+1]-1] = \tag{4}$$
$$Compress(A')[Start[i+\log^2 n]\,..\,Start[i+k+1]-1]$$
$$\text{and } A'[j\,..\,j+\log^2 n] = A'[i\,..\,i+\log^2 n] \tag{5}$$

We call the checks of the form (4) the *compressed consistency checks*, checks of the form (5) the *short consistency checks* and the *near short consistency checks* when moreover $|i-j| < \log^2 n$.

The compressed consistency checks can be answered in amortised constant time using LCA query [3] on the suffix tree built for $Compress(A')$. What is left is to show how to perform short consistency checks in amortised constant time as well.

Performing near short consistency checks. To do near short consistency checks efficiently, we split A' into blocks of $\log^2 n$ consecutive letters: $A' = B_1 B_2 \ldots B_k$, see Fig 5. Then we build suffix trees for each pair of consecutive blocks, i.e., $B_1B_2, B_2B_3, \ldots, B_{k-1}B_k$. Each block contains at most $\log^2 n$ values smaller than $\log^2 n$, and at most $48 \log n$ larger values, by Lemma 5, so all the suffix trees can be built in linear time by Lemma 4. For each tree we also build a data structure supporting constant-time LCA queries [3]. Then, any near short consistency check reduces to an LCA query in one of these suffix trees.

Performing general short consistency checks. General short consistency checks are answered by near short consistency checks and naive letter-to-letter comparisons. To obtain linear total time, the results of previous short consistency checks are reused as follows. We store the value j_{best} for which the length of the common prefix of $A'[j\,..\,j+\log^2 n]$ and $A'[i\,..\,i+\log^2 n]$ is relatively long as well as the length $L \le \log^2 n$ itself. When another short consistency checkis done for j such that $|j - j_{best}| \le \log^2 n$, we first compute the common prefix

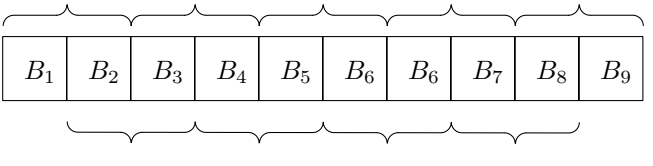


Fig. 5. Scheme of ranges for suffix trees

of $A'[j\,..\,j+\log^2 n]$ and $A'[j_{best}\,..\,j_{best}+\log^2 n]$ and compare it with L: if it is smaller, then clearly the common prefix of $A'[j\,..\,j+\log^2 n]$ and $A'[i\,..\,i+\log^2 n]$ is smaller than L; if it equals L, then we naively check if $A'[j+L+k] = A'[i+L+k]$ for consecutive k. If $|j - j_{best}| > \log^2 n$, j becomes j_{best}, and we find the longest common prefix naively. Amortised analysis yields the following.

Lemma 7. *Answering all consistency checks can be done in $\mathcal{O}(n)$ time.*

Running time. VALIDATE-π' runs in $\mathcal{O}(n)$ time: construction of the suffix trees and doing consistency checks, as well as doing pin checks all take $\mathcal{O}(n)$ time.

Remarks. While VALIDATE-π produces the word w over the minimum alphabet such that $\pi_w = A$ on-line, this is not the case with VALIDATE-π'. At each time-step VALIDATE-π' can output a word over minimum alphabet such that $\pi'_w = A'$, but the letters assigned to positions on the last slope may yet change as further entries of A' are read.

Since VALIDATE-π' keeps the function $\pi[1\,..\,n+1]$ after reading $A'[1\,..\,n]$, virtually no changes are required to adapt it to g validation, where $g(i) = \pi'[i-1]+1$ is the function considered by Duval et al. [8], because $A'[1\,..\,n-1]$ can be obtained from $g[1\,..\,n]$. Running VALIDATE-π' on such A' gives $A[1\,..\,n]$ that is consistent with $A'[1\,..\,n-1]$ and $g[1\,..\,n]$. Similar proof shows that $A[1\,..\,n]$ and $g[1\,..\,n]$ require the same minimum size of the alphabet.

5 Open Problems

Two interesting questions remain: is it possible to remove the suffix trees and LCA queries from our algorithm without hindering its time complexity? We believe that deeper combinatorial insight might result in a positive answer.

References

1. Breslauer, D., Colussi, L., Toniolo, L.: On the comparison complexity of the string prefix-matching problem. J. Algorithms 29(1), 18–67 (1998)
2. Clément, J., Crochemore, M., Rindone, G.: Reverse engineering prefix tables. In: Proceedings of 26th STACS, pp. 289–300 (2009)
3. Cole, R., Hariharan, R.: Dynamic lca queries on trees. In: Proceedings of SODA '99, Philadelphia, PA, USA, pp. 235–244. Society for Industrial and Applied Mathematics (1999)
4. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press, Cambridge (2007)
5. Crochemore, M., Rytter, W.: Jewels of Stringology. World Scientific Publishing Company, Singapore (2002)
6. Dietzfelbinger, M., Karlin, A.R., Mehlhorn, K., auf der Heide, F.M., Rohnert, H., Tarjan, R.E.: Dynamic perfect hashing: Upper and lower bounds. SIAM J. Comput. 23(4), 738–761 (1994)
7. Duval, J.-P., Lecroq, T., Lefebvre, A.: Border array on bounded alphabet. Journal of Automata, Languages and Combinatorics 10(1), 51–60 (2005)

8. Duval, J.-P., Lecroq, T., Lefebvre, A.: Efficient validation and construction of Knuth-Morris-Pratt arrays. In: Conference in honor of Donald E. Knuth (2007)
9. Duval, J.-P., Lecroq, T., Lefebvre, A.: Efficient validation and construction of border arrays and validation of string matching automata. ITA 43(2), 281–297 (2009)
10. Farach, M.: Optimal suffix tree construction with large alphabets. In: Proceedings of FOCS '97, Washington, DC, USA, pp. 137–143. IEEE Computer Society, Los Alamitos (1997)
11. Franěk, F., Gao, S., Lu, W., Ryan, P.J., Smyth, W.F., Sun, Y., Yang, L.: Verifying a border array in linear time. J. Comb. Math. Comb. Comput. 42, 223–236 (2002)
12. Fredman, M.L., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comput. Syst. Sci. 48(3), 533–551 (1994)
13. Hancart, C.: On Simon's string searching algorithm. Inf. Process. Lett. 47(2), 95–99 (1993)
14. I, T., Inenaga, S., Bannai, H., Takeda, M.: Counting parameterized border arrays for a binary alphabet. In: Proc. of the 3rd LATA, pp. 422–433 (2009)
15. Karp, R.M., Miller, R.E., Rosenberg, A.L.: Rapid identification of repeated patterns in strings, trees and arrays. In: STOC '72: Proceedings of the fourth annual ACM symposium on Theory of computing, pp. 125–136. ACM, New York (1972)
16. Knuth, D.E., Morris Jr., J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. 6(2), 323–350 (1977)
17. McCreight, E.M.: A space-economical suffix tree construction algorithm. J. ACM 23(2), 262–272 (1976)
18. Moore, D., Smyth, W.F., Miller, D.: Counting distinct strings. Algorithmica 23(1), 1–13 (1999)
19. Morris Jr., J.H., Pratt, V.R.: A linear pattern-matching algorithm. Technical Report 40, University of California, Berkeley (1970)
20. Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms 51(2), 122–144 (2004)
21. Simon, I.: String matching algorithms and automata. In: Karhumäki, J., Rozenberg, G., Maurer, H.A. (eds.) Results and Trends in Theoretical Computer Science. LNCS, vol. 812, pp. 386–395. Springer, Heidelberg (1994)
22. Ukkonen, E.: On-line construction of suffix trees. Algorithmica 14(3), 249–260 (1995)

Identical Relations in Symmetric Groups and Separating Words with Reversible Automata

R.A. Gimadeev[1] and M.N. Vyalyi[2]

[1] Moscow Institute of Physics and Technology
renat.ariacas@gmail.com
[2] Dorodnitsyn Computing Center of RAS
vyalyi@gmail.com

Abstract. Separating words with automata is a longstanding open problem in combinatorics on words. In this paper we present a related algebraic problem. What is the minimal length of a nontrivial identical relation in the symmetric group S_n?

Our main contribution is an upper bound $2^{O(\sqrt{n}\log n)}$ on the length of the shortest nontrivial identical relation in S_n. We also give lower bounds for words of a special types. These bounds can be applied to the problem of separating words by reversible automata. In this way we obtain an another proof of the Robson's square root bound.

Keywords: automaton, identity, symmetric group.

The problem of separating words with automata is to determine the size of the smallest deterministic automaton distinguishing two words. The best known upper bound is $O(\ell^{2/5}\log^{3/5}\ell)$ states, where ℓ is the length of the words. It was obtained by J.M. Robson [7] in 1989. The logarithmic bound $\Omega(\log\ell)$ states is the best known lower bound for this problem [2]. Later Robson found reversible automata with $O(\sqrt{\ell})$ states separating the words of length ℓ [8]. This bound is the best known for reversible automata.

Better results were achieved in separating words with context-free grammars. In [2] the upper bound $O(\log\ell)$ and the lower bound $\Omega(\log\ell/\log\log\ell)$ were proved.

There are similar problems concerning the reconstruction of words from fragments [4]. A fragment is a subsequence of a word. The main problem in reconstruction from fragments is to determine the minimal fragment length such that any pair of words can be distinguished by multisets of fragments of this length. In this problem the bounds also differ drastically. The best lower bound is logarithmic (for the best factor see [9]) and the best upper bound is square root (obtained by I. Krasikov, Y. Roditty in 1997 [3], for the best factor see [5]).

In this paper we present a related algebraic problem: to determine the minimal length of a nontrivial identical relation in the symmetric group S_n.

Let's recall basic definitions from combinatorial group theory (see [6]).

Let $X_s = \{x_0,\ldots,x_{s-1}\}$ and $X_s^{\pm1} = X_s \cup \{x_0^{-1},\ldots,x_{s-1}^{-1}\}$. We consider words over the alphabet $X_s^{\pm1}$ as multiplicative expressions in the variables x_i. By definition a word w is *an identical relation* in a group G iff it takes the identity

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 144–155, 2010.

value under all possible assignments of variables x_i to elements of the group G. For example, a word $x_0x_0^{-1}$ is an identical relation in any group.

A reduced word do not contain subwords in the form $x_kx_k^{-1}$ or $x_k^{-1}x_k$. *A nontrivial identical relation* is a nonempty reduced word which is an identical relation. Let $L_s(n)$ be the length of the shortest nontrivial identical relation in s variables in the symmetric group S_n. The identity problem is to determine the value of the $L_s(n)$. To the best of our knowledge the problem was not discussed earlier.

The identity problem has an easy solution for a 1-letter alphabet (see Section 1). For $s > 1$ there is an exponential gap between known upper and lower bounds of the $L_s(n)$.

In this paper we prove the upper bound $L_s(n) = 2^{O(\sqrt{n}\log n)}$ for each $s > 1$ (see Theorem 1 in Section 2). Up to the date we are unable to prove any nonlinear lower bound of the $L_s(n)$ for $s > 1$.

We also consider a restriction of the identity problem to special classes of words. In this setting it is helpful to introduce *the permutation complexity* $\nu(w)$ of a word $w \in (X_s^{\pm 1})^*$ as the minimal n such that w is not an identical relation in the group S_n. Note that an upper bound on the minimum of $\nu(w)$ over all words of length at most ℓ over the alphabet $X_s^{\pm 1}$ implies a lower bound of the $L_s(n)$ and vice versa.

For words of a special type the permutation complexity is related to separating with reversible automata. Namely, if a word has the form $w = uv^{-1}$, $u, v \in X_s^*$, then $\nu(w)$ is the smallest size of a reversible automaton separating words u and v. So, for the permutation complexity of these words the square root upper bound holds due to Robson's bound for separating words with reversible automata.

In this paper we generalize the above example. The crucial property for establishing sublinear upper bounds on permutation complexity is expressed in terms of a walk on the integer lattice $\mathbb{Z}^s$ induced by the word w. We call this property *edge unbalancing*. For $\mathbb{Z}^2$-edge unbalanced words we prove the upper bound $\nu(w) = O(\ell^{2/3}\log \ell)$, where ℓ is the length of w (see Theorem 2 in Section 3). In the case of words in the form $w = uv^{-1}$, $u, v \in X_s^*$, it can be improved to Robson's bound of $O(\sqrt{\ell})$ (Section 4).

1 Identical Relations in Groups: Definitions and Simple Facts

Let $w = w_\ell w_{\ell-1} \dots w_1$ be a word over the alphabet $X_s^{\pm 1}$. For any group G and an assignment of variables $x_k := g_k$, where g_k – elements of G, the evaluation of the word w is the product

$$\mathrm{Ev}(w; g_0, \dots, g_{s-1}) = \tilde{g}_\ell \tilde{g}_{\ell-1} \dots \tilde{g}_1, \qquad \tilde{g}_i = \begin{cases} g_k, & \text{if } w_i = x_k; \\ g_k^{-1}, & \text{if } w_i = x_k^{-1}. \end{cases}$$

Definition 1. A word w is *an identical relation* in a group G if

$$\mathrm{Ev}(w; g_0, \dots, g_{s-1}) = e \text{ for all } g_0, \dots, g_{s-1} \in G,$$

where e is the identity element of the group G.

Words $x_0x_0^{-1}$ and $x_0^{-1}x_0$ are identical relations in any group. So, insertion or deletion of these words do not change the evaluation function. Recall that the reduced form of a word is obtained by removing all subwords of the form $x_ix_i^{-1}$ and $x_i^{-1}x_i$ as long as possible (for details see books in combinatorial group theory, say, [6]).

It was mentioned above that *a nontrivial identical relation* is a nonempty reduced word which is an identical relation.

Definition 2. *The permutation complexity* $\nu(w)$ of the word w is the minimal n such that w is not an identical relation in the symmetric group S_n. (For trivial identical relations $\nu = +\infty$.)

The length of the shortest nontrivial identical relation $L_s(n)$ is

$$L_s(n) = \min(|w| : w \in (X_s^{\pm 1})^*,\ n < \nu(w) < +\infty).$$

Recall that commutator $[x, y]$ of group elements x and y equals $xyx^{-1}y^{-1}$.

Example 1. The permutation complexity of the word

$$w = [[x_0, x_1], [x_2, x_3]] = x_0x_1x_0^{-1}x_1^{-1}x_2x_3x_2^{-1}x_3^{-1}x_1x_0x_1^{-1}x_0^{-1}x_3x_2x_3^{-1}x_2^{-1}$$

equals 4. Indeed, the word w is an identical relation in any dihedral group (an easy exercise) and the group S_3 is isomorphic to the dihedral group D_3. In the group S_4 the word w takes the value $(14)(23)$ under the assignment $x_0 := (12)$, $x_1 := (23)$, $x_2 := (13)$, $x_3 := (34)$.

It is easy to determine the value of $L_1(n)$.

Lemma 1. *The least common multiple of orders of elements in the symmetric group* S_n *is* $\operatorname{lcm}(1, 2, \ldots, n)$. *For each integer* $k < \operatorname{lcm}(1, 2, \ldots, n)$ *there is a permutation* $\pi \in S_n$ *such that* $\operatorname{ord}\pi \nmid k$.

Proof. The first claim follows from the cyclic decomposition of a permutation. It is clear that the order of a permutation with the cycle lengths $n_1, \ldots n_t$ is $\operatorname{lcm}(n_1, \ldots, n_t)$.

If $k < \operatorname{lcm}(1, 2, \ldots, n)$ then $j \nmid k$ for some $1 \leq j \leq n$. So, a cycle of length j satisfies the second claim. □

The exact answer for $L_1(n)$ follows immediately from the previous lemma.

Lemma 2. $L_1(n) = \operatorname{lcm}(1, 2, \ldots, n) = e^{n+o(n)}$.

The asymptotic bound in Lemma 2 is due to the well-known asymptotic bound on the Chebyshev function $\theta(x)$. We cite it below in Section 2.

For all $s \geq 2$ all functions $L_s(n)$ are asymptotically equal up to a constant factor. The exact claim is stated in the following lemma.

Lemma 3. *For any* $s \geq 2$ *we have* $L_s(n) = \Theta(L_2(n))$ *for* $n \to \infty$.

Proof. By definition $L_s(n) \le L_k(n)$ if $s > k$. Indeed, a k-letter identical relation is an s-letter identical relation. So, $L_s(n) = O(L_2(n))$.

Let w be a nonempty reduced identical relation over the alphabet $X_s^{\pm 1}$. To build an identical relation over the alphabet $X_2^{\pm 1}$ we substitute each letter x_i by an appropriate word u_i in X_2. To guarantee that the reduced form of the resulted word is not empty we choose u_i such that

$$u_i = b_i^{-1} x_1 b_i, \quad b_i = \prod_{\alpha=0}^{\ell} x_{i_\alpha},$$

where $i_0 i_1 \cdots i_\ell$ is binary representation of i of length $\ell = \lceil \log_2 s \rceil + 1$ padded by zeroes to the left. The words b_i are pairwise distinct. So the reduced form of the substituted word is nonempty and its length is $O(|w| \log s)$. Consequently, $L_s(n) = \Omega(L_2(n))$. □

Due to Lemma 3 we can now focus on the case of 2-letter alphabet X_2.

2 The Upper Bound for the Identity Problem

Let's start from a simple example.

Example 2. $L_2(5) < L_1(5) = \mathrm{lcm}(1, 2, 3, 4, 5) = 60$. Indeed, take a word

$$w = x_0^5 x_1 x_0^{12} x_1^{-1} x_0^{-5} x_1 x_0^{-12} x_1^{-1} = [x_0^5, x_1 x_0^{12} x_1^{-1}]$$

of length 38.

If the order d of the permutation $x_0 \in S_5$ is not 5 then d is a divisor of 12 because all cycles in the cycle decomposition of x_0 are shorter than 5. So, either x_0^5 or x_0^{12} is the identity in the S_5. In both cases w takes the identity value.

Taking commutators will be the main tool in construction below.

Let $H_0, H_1, \ldots, H_{N-1}$ be integers and $N = 2^h$ is a power of 2. Note that vertices of the complete binary tree T_h of the height h have a natural labeling by binary words of length $\le h$. Namely, the root of the tree is labeled by the empty word λ and descendants of the vertex labeled by a word a are labeled by words $a0$ and $a1$. We assign a word w_a over the alphabet $X_2^{\pm 1}$ to a vertex of T_h labeled by a word a using the following inductive rule:

- If the length of a is h then $w_a = [x_0^{H_{\overline{a}_2}}, x_1]$, where $\overline{a}_2$ is the integer with binary representation a;
- If the length of a is less than h then $w_a = [w_{a0}, w_{a1}]$.

We will refer to the words w_λ assigned to the root of T_h as *the tree commutators* based on the set $\{H_0, \ldots, H_{N-1}\}$.

Below we show that a tree commutator for appropriately chosen integers H_i is a nontrivial identical relation of sufficiently short length.

Example 3. Let $H_0 = 15$, $H_1 = 12$, $H_2 = 8$, $H_3 = 7$. Then

$$w_\lambda = [[[x_0^{15}, x_1], [x_0^{12}, x_1]], [[x_0^8, x_1], [x_0^7, x_1]]]$$

$$w_0 = [[x_0^{15}, x_1], [x_0^{12}, x_1]] \qquad w_1 = [[x_0^8, x_1], [x_0^7, x_1]]$$

$$w_{00} = [x_0^{15}, x_1] \qquad w_{01} = [x_0^{12}, x_1] \qquad w_{10} = [x_0^8, x_1] \qquad w_{11} = [x_0^7, x_1]$$

Any order of the permutation in the group S_8 is a divisor of some integer H_i. So, w_λ is an identical relation in the S_8. The length of the reduced form of w_λ equals

$$240 < \operatorname{lcm}(1, 2, 3, 4, 5, 6, 7, 8) = 840.$$

In general case we have the following bound on the length of the reduced form of w_λ.

Lemma 4. *If integers H_i are pairwise distinct and $\max_i H_i \le H$ then for the reduced form w' of the tree commutator w_λ we have $0 < |w'| \le (2H + 2)4^h = (2H + 2)N^2$.*

Proof. The upper bound is proved by the induction on h. For $h = 0$ the statement holds: $|[x_0^{H_0}, x_1]| \le (2H + 2)$.

If $|w_0|$ and $|w_1|$ do not exceed $(2H + 2)4^{h-1}$ then $|w_\lambda| = 2|w_0| + 2|w_1| \le (2H + 2)4^h$.

Note that

$$[[x_0^{H_1}, x_1], [x_0^{H_2}, x_1]] = x_0^{H_1} x_1 x_0^{-H_1} x_1^{-1} x_0^{H_2} x_1 x_0^{-H_2} x_1^{-1} x_1 x_0^{H_1} x_1^{-1} x_0^{-H_1} x_1 x_0^{H_2} x_1^{-1} x_0^{-H_2}$$

is reducible. For $H_1 \ne H_2$ its reduced form starts with $x_0^{H_1} x_1$ and ends with $x_1^{-1} x_0^{-H_2}$.

By downward induction we prove the following claim: for any a of length $< h$ the reduced form of the word w_a starts with $x_0^{H_1} x_1$ and ends with $x_1^{-1} x_0^{-H_2}$ and binary representations of H_1, H_2 starts with a.

The basis $|a| = h - 1$ has been proven above. The inductive step follows from the computation

$$w_a = [w_{a0}, w_{a1}] =$$
$$x_0^{H_1} x_1 \dots x_1^{-1} x_0^{-H_2} x_0^{H_3} x_1 \dots x_1^{-1} x_0^{-H_4} x_0^{H_2} x_1 \dots x_1^{-1} x_0^{-H_1} x_0^{H_4} x_1 \dots x_1^{-1} x_0^{-H_3} =$$
$$x_0^{H_1} x_1 \dots x_1^{-1} x_0^{H_3 - H_2} x_1 \dots x_1^{-1} x_0^{H_2 - H_4} x_1 \dots x_1^{-1} x_0^{H_4 - H_1} x_1 \dots x_1^{-1} x_0^{-H_3}$$
$$= x_0^{H_1} x_1 \dots x_1^{-1} x_0^{-H_3}$$

Here $H_3 - H_2 \ne 0$, $H_2 - H_4 \ne 0$, $H_4 - H_1 \ne 0$ by the induction hypothesis (binary representations of their indices differ so the integers are distinct). □

Definition 3. A set of integers $\mathcal{H} = \{H_0, H_1, \dots, H_{N-1}\}$ is called *n-hitting* if for each set $n_1, n_2, \dots, n_t$ of integers such that $\sum_{i=1}^{t} n_i \le n$ there is a common multiple of n_i in the set $\mathcal{H}$.

Example 3 gives an illustration of this definition. The integers $15, 12, 7, 8$ form a 8-hitting set: 12 is a common multiple of 3, 2, 2 and so on.

Lemma 5. *If a set $\mathcal{H}$ is an n-hitting set then the tree commutator based on the set $\mathcal{H}$ is an identical relation in the group S_n.*

Proof. If a word w_a takes the identity value then for all prefixes b of a, the word w_b takes also the identity value. Let $n_1, n_2, \ldots, n_t$ be the lengths of cycles in the cycle decomposition of the permutation $x \in S_n$. Then some integer H_i is a multiple of the order of the x by definition of a hitting set. This means that some leaf of the tree takes the identity value under the assignment $x_0 := x$. Thus the root of the tree takes the identity value for all assignments of the variable x_0. □

Now we construct an n-hitting set $\mathcal{H}_n$ for arbitrary n. In the construction we use the notation $m = \lceil \sqrt{n} \rceil$ and

$$K_n = \prod_{i=1}^{\lfloor \log_2 n \rfloor} \prod_{\substack{p^i \le n, p \le m \\ p \text{ is a prime}}} p.$$

(In what follows we assume that the variable p (possibly indexed) takes prime values only.)

Note that the integer K_n is the factor of the prime factorization of the $\mathrm{lcm}(1, 2, \ldots, n)$ that includes all prime powers greater than 1.

We need a standard number theoretical asymptotic bound.

Theorem (See [1]). $\theta(x) = \sum_{p \le x} \log p = x + o(x)$.

This gives a bound on K_n for sufficiently large n:

$$\log K_n = \sum_{i=1}^{\lfloor \log_2 n \rfloor} \sum_{p^i \le n, p \le m} \log p = \sum_{p \le m} \log p + \sum_{i=2}^{\lfloor \log_2 n \rfloor} \sum_{p^i \le n} \log p =$$

$$\sum_{p \le m} \log p + \sum_{i=2}^{\lfloor \log_2 n \rfloor} \sum_{p \le n^{1/i}} \log p =$$

$$m + n^{1/2} + o(n^{1/2}) + n^{1/3} + o(n^{1/3}) + \cdots + n^{1/\lfloor \log_2 n \rfloor} + o(n^{1/\lfloor \log_2 n \rfloor})$$

$$\le m + n^{1/2} + o(n^{1/2}) \le 3m$$

The set $\mathcal{H}_n$ consists of all products of m different integers in the form $K_n p$, where $m < p \le n$. So elements of the $\mathcal{H}_n$ are $K_n \cdot p_{l_1} \cdot p_{l_2} \cdot \ldots \cdot p_{l_m}$, where $m < p_\alpha \le n$ and $p_\alpha \ne p_\beta$ for $\alpha \ne \beta$.

By construction the cardinality of the set $\mathcal{H}_n$ is less than

$$\binom{n}{m} < n^m = e^{m \ln n}$$

(there are less than n primes in the range $[m+1, n]$). The maximum over the set $\mathcal{H}_n$ is at most $n^m \cdot K_n \le e^{m \ln n} e^{3m} = e^{m \ln n + 3m}$.

Lemma 6. *The set $\mathcal{H}_n$ is an n-hitting set.*

Proof. Let $n_1, \ldots, n_t$ be a set of integers such that $\sum_{i=1}^{t} n_i \le n$. Consider the prime factorization of $A = \operatorname{lcm}(n_1, \ldots, n_t)$:

$$A = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_j^{\alpha_j} p_{j+1} \ldots p_r.$$

We assume that $p_1 < p_2 < \cdots < p_r$ and $p_j \le m < p_{j+1}$. Let's divide A into two factors:

$$A_1 = \prod_{i=1}^{j} p_i^{\alpha_i}, \quad A_2 = \prod_{i=j+1}^{r} p_i.$$

Note that $A_1 \mid K_n$ because $A \mid \operatorname{lcm}(1, 2, \ldots, n)$ and the K_n includes all prime powers of the $\operatorname{lcm}(1, 2, \ldots, n)$ greater than 1. On the other hand, $A_2 = p_{j+1} \ldots p_r$ is a product of distinct primes and $r - j \le m$ (there are no more than $\sqrt{n}$ integers n_i such that $n_i \ge \sqrt{n}$). Thus A is a divisor of some integer from the set $\mathcal{H}_n$. □

Now we get the announced upper bound on $L_2(n)$.

Theorem 1. *$L_2(n) \le e^{4\sqrt{n}\ln n}$ for sufficiently large n.*

Proof. To apply the tree commutator construction described above we extend the set $\mathcal{H}_n$ by suitable number of distinct integers to ensure that the cardinality N of the extended set is a power of 2 and the reduced form of the tree commutator is not empty. This extension increase the cardinality by factor at most 2 as well as the maximal integer in the set.

Lemma 5 and 6 say that the tree commutator based on the extended set is an identity relation in the group S_n.

To bound the length of the tree commutator we use Lemma 4. For sufficiently large n the cardinality of the extended set is at most $2e^{m \ln n}$ and the maximal integer in it is at most $2e^{m \ln n + 3m}$. Thus the length of the reduced form of the tree commutator is at most $(2 \cdot e^{m \ln n + 3m} + 2) \cdot (2 \cdot e^{m \ln n})^2 \le e^{4\sqrt{n} \ln n}$. □

3 Lower Bounds for the Permutation Complexity of Unbalanced Words

Definition 4. A word $w \in (X_s^{\pm 1})^*$ is called *balanced* if $|w|_{x_i} - |w|_{x_i^{-1}} = 0$ for all i.

Lemma 7. *Let w be an unbalanced word. Then $\nu(w) = O(\log \ell)$, $|w| = \ell$.*

Proof. Suppose that $|w|_{x_0} - |w|_{x_0^{-1}} \ne 0$. Assign the identity value to all variables except x_0. Then by Lemma 2

$$\nu(w) = O(\log k),$$

where k is the number of the letters x_0, x_0^{-1} in the word w. □

Now we introduce a stronger notion of *edge balancing*. It is based on a correspondence between words and walks on digraphs of special kind. These digraphs include transition graphs for reversible automata and Cayley graphs for finitely generated groups.

Definition 5. *An s-regular digraph* is a digraph $\Gamma = (V, E)$ equipped by an edge coloring $c\colon E \to [1, \dots, s]$ in s colors such that for each vertex $v \in V$ and each color i there is the exactly one edge of i-th color going out the vertex v and there is the exactly one edge of i-th color going to the vertex v.

(We assume that loops and parallel edges are permitted for digraphs.)

Note that if a vertex set V is finite then an s-regular digraph is a transition graph for a deterministic reversible automaton.

Cayley graphs for finitely generated groups are also s-regular: a vertex set in this case is a group and colors are generators used in Cayley graph construction.

A word $w \in X_s^{\pm 1}$ induces the unique walk

$$\tau_\Gamma(w; v_0) = v_0 e_1 v_1 e_2 \dots e_\ell v_\ell$$

on an s-regular edge colored digraph Γ with a given base point v_0 by the following rule: if $w_i = x_{c_i}$ then the edge e_i has a color c_i and goes from v_{i-1} to v_i (a *forward pass* through the edge), if $w_i = x_{c_i}^{-1}$ then the edge e_i has a color c_i and goes from v_i to v_{i-1} (a *reverse pass*).

The multiplicity of the edge e in the walk $\tau_\Gamma(w; v_0)$ is the difference between the number of forward passes and the number of reverse passes through the edge.

Definition 6. A word w is (Γ, v_0)-*edge balanced* if the multiplicity of each edge $e \in E(\Gamma)$ in the walk $\tau_\Gamma(w; v_0)$ is zero. Otherwise, the word is (Γ, v_0)-*edge unbalanced*.

A word w is n-*universally edge balanced* if it is (Γ, v_0)-edge balanced for all digraphs Γ on $\le n$ vertices and for all $v_0 \in V(\Gamma)$.

For a finite s-regular digraph on n vertices edges of i-th color form a graph of the permutation g_i on n vertices. Note that the mapping that sends a vertex v_0 to the last vertex v_ℓ of the walk $\tau_\Gamma(w; v_0)$ is the value of the reversed word w^R under the assignment $x_i := g_i$.

It is easy to see that n-universally edge balanced words are identical relations in the group S_n. Indeed, an edge balanced walk is cyclic: it starts and finishes at the same vertex. On the other hand, a construction of the n-universally edge balanced word from an identical relation w in S_n is also straightforward: just take the word

$$b = [w^{-1}, x_0][w, x_0] = w^{-1} x_0 w x_0^{-1} w x_0 w^{-1} x_0^{-1}.$$

It is an n-universally edge balanced because for endpoints u, v of any 0-colored edge (u, v) the cyclic walks $\tau_\Gamma(w; u)$ and $\tau_\Gamma(w; v)$ are passed exactly once in forward and reverse directions during the walk $\tau_\Gamma(b; u)$ and the edge (u, v) is passed exactly twice in forward and reverse directions. So, there are n-universally edge balanced words of length $\le 4L_s(n) + 4$.

In the opposite direction a bound is established by the following lemma.

Lemma 8. *There is a constant C such that for any Γ-edge unbalanced word w we have $\nu(w) \le |V(\Gamma)|(C \log |w| + 3)$.*

Proof. Denote by r the non-zero multiplicity of the edge $e \in E(\Gamma)$. Obviously, $|r| \le |w|$. Choose the least prime p that does not divide r. Note that $p \le C \log |r| + 3$ for some constant C.

To construct for the word w a non-identity assignment of the permutations of the degree $p|V(\Gamma)|$ we use the following idea: count the multiplicity of the edge e modulo p while walking along the digraph Γ.

More formally, let's define a permutation π_i on the set $V(\Gamma) \times Z_p$ (Z_p is a cyclic group of the order p) by the rules: if

$$\pi_i(v, j) = (v', j')$$

then an i-colored edge e' starts at the v and ends at the v' and

$$j' = \begin{cases} j, & \text{if } e' \ne e; \\ j + 1 \pmod p, & \text{if } e' = e. \end{cases}$$

We claim that the evaluation of the word $w = w_\ell \cdots w_2 w_1$ at the assignment $x_i := \pi_i$ is not the identity. More exactly,

$$\mathrm{Ev}(w; \pi_1, \ldots, \pi_s)(v_0, 0) = (\tilde{v}, a),$$

where $a \ne 0$. Indeed, while applying the evaluations of the suffixes of the w under the assignment $x_i := \pi_i$ to $(v_0, 0)$ the first components run the vertices along the walk $\tau_\Gamma(w; v_0)$. At the same time a forward pass through the edge e adds $+1$ modulo p in the second component and a reverse pass adds -1. So, the final value of the second component equals $r \bmod p \ne 0$. □

For an infinite graph Lemma 8 gives no interesting bound. Meanwhile, any reduced nonempty word is edge unbalanced for an infinite graph. Note that an s-regular infinite tree is the Cayley graph of the free group of the rank s. So, the multiplicities of the edges for the walk induced by a reduced word are ± 1.

An another example of the infinite graph is the integer lattice $\mathbb{Z}^s$ — the Cayley graph of the free Abelian group of the rank s. Let's take the case $s = 2$.

Vertices of the $\mathbb{Z}^2$ are pairs of integers (i, j). Edges of the form $((i, j), (i+1, j))$ are colored in the color 0 and edges of the form $((i, j), (i, j + 1))$ are colored in the color 1.

Theorem 2. *$\nu(w) = O(\ell^{2/3} \log \ell)$ for a $\mathbb{Z}^2$-edge unbalanced word w of length ℓ.*

Theorem 2 follows from Lemma 8. To apply it we show that for some digraph Γ on $O(\ell^{2/3})$ vertices the word w is edge unbalanced.

We will choose a graph Γ as the Cayley graph of the cyclic group Z_q of the order q.

W.l.o.g. we assume that a $\mathbb{Z}^2$-unbalanced edge has a color 0 and the word w is balanced. We form a polynomial in two variables

$$f_w(x, y) = \sum_{j,k} \delta_0(j, k) x^j y^k,$$

where $\delta_0(j,k)$ is the multiplicity of the edge $((j,k),(j+1,k))$. Actually, f_w is a Laurent polynomial because the walk can go through vertices with negative coordinates. Nevertheless, one can exclude negative exponents in f_w by changing the base point.

For a pair g_0, g_1 generating Z_q the digraph Γ_{q,g_0,g_1} is the Cayley graph of Z_q w.r.t. to the generating set $\{g_0, g_1\}$.

Suppose that the word w is Γ_{q,g_0,g_1}-edge balanced for all $q \le m$, g_0, g_1.

Edge balancing for 0-colored edges in the Γ_{q,g_0,g_1} is expressed by equations in the coefficients of the polynomial f_w:

$$\sum_{g_0 j + g_1 k \equiv c \pmod q} \delta_0(j,k) = 0, \tag{1}$$

where c ranges from 0 to $q-1$.

Now we check that for any primitive root of unity ω_q of the degree q and for any integers a, b equations (1) for all pairs g_0, g_1 imply $f_w(\omega^a, \omega^b) = 0$. Indeed, compute $f_w(\omega^a, \omega^b)$:

$$f(\omega^a, \omega^b) = \sum \delta_0(j,k)\omega^{aj}\omega^{bk} = \sum_{c=0}^{q-1} \omega^c \sum_{aj+bk=c \bmod q} \delta_0(j,k) = 0.$$

To complete the proof of the Theorem 2 we need a pair a, b of integers such that $f_w(t^a, t^b)$ is not zero. Let a, b be such a pair that a linear form $ax_0 + bx_1$ takes the maximal value N on the Newton polytope of the f_w at the unique point. Then the Nth coefficient of the $f_w(t^a, t^b)$ is not zero. So, $f_w(t^a, t^b)$ satisfies the conditions of the following lemma.

Lemma 9. *Let $\Bbbk$ be an algebraically closed field and $g(t) \in \Bbbk[t]$ be a nonzero polynomial. If $g(t)$ vanishes at all roots of unity of the degree $\le m$ then $\deg g = \Omega(m^2)$.*

Proof. Polynomials vanishing at primitive roots of unity of the degree q form an ideal I. The ideal I is generated by the cyclotomic polynomial $\Phi_q(t)$. The degree of the $\Phi_q(t)$ is $\varphi(n)$, where $\varphi(n)$ is the Euler function.

Cyclotomic polynomials are relatively prime so the polynomial $g(t)$ should be a multiple of the product of $\Phi_q(t)$, $q \le m$. Thus

$$\deg g \ge \sum_{i=1}^{m} \deg \Phi_i(m) = \sum_{i=1}^{m} \varphi(i) \ge m^2/6.$$

(The last inequality can be found in [10].) □

Now we are able to prove the bound on the permutation complexity $\nu(w)$. It is clear that $\deg f_w \le \ell$. Below we denote the degree of f_w by d. To bound values of a, b note that each edge of the Newton polytope defined by an equation $a'x_0 + b'x_1 = c'$ with relatively prime a', b' contributes b' in the degree of the

polynomial f_w. All relatively prime pairs with elements $a < b \leq B$ make a contribution

$$\sum_{b=1}^{B} b\varphi(b) = \Omega(B^3).$$

So, there exists a pair a, b such that the maximal value of its elements is $O(d^{1/3})$. Thus we have

$$\ell^{4/3} \geq \ell d^{1/3} \geq d \max(a, b) \geq \deg f_w(t^a, t^b) = \Omega(m^2). \tag{2}$$

The last inequality is due to Lemma 9.

Thus for some $q = O(\ell^{2/3})$, g_0, g_1 the word w is Γ_{q,g_0,g_1}-edge unbalanced. By Lemma 8 we get $\nu(w) = O(\ell^{2/3} \log \ell)$.

The proof of the Theorem 2 is completed.

4 An Application to Separating Words with Reversible Automata

Contrary to the case of the infinite tree there are reduced words that are $\mathbb{Z}^2$-edge balanced. For any pair of balanced words u, v the commutator $[u, v]$ is $\mathbb{Z}^2$-edge balanced.

But $\mathbb{Z}^2$-edge unbalanced words cover the important case of words in the form uv^{-1}, where $u, v \in X_s^*$. The maximum of the permutation complexity for these words of length 2ℓ equals the smallest size of reversible automata separating words of length ℓ.

For the words in the form uv^{-1}, where $u, v \in X_s^*$, the bound of the Theorem 2 can be improved to Robson's square root bound. In this section we describe this improvement.

Now we prove that a word $w = uv^{-1}$, $u \neq v \in X_2^*$, is $\mathbb{Z}^2$-edge unbalanced.

To indicate an unbalanced edge let's take the last position in which the words u, v differ. W.l.o.g. we assume that $u = u_0 x_1 w_0$ and $v = v_0 x_0 w_0$. Let $\#_0(u_0)$ be the number of x_0 in the word u_0 and $\#_1(u_0)$ be the number of x_1. Then 0-colored edge starting at the point $(\#_0(u_0), \#_1(u_0))$ has multiplicity -1 in the $\mathbb{Z}^2$-walk induced by the uv^{-1}. The form $x_0 + x_1$ takes the maximal value on the Newton polytope of the polynomial f_w at the same point and it is the unique point of maximum. So, $f_w(t, t) \neq 0$.

Thus we can omit the factor $d^{1/3}$ in the bound (2) and get

$$m = O(\ell^{1/2}).$$

Direct use of the Lemma 8 gives $\nu(w) = O(\ell^{1/2} \log \ell)$. But note that $f_w \neq 0$ in a field of characteristic 2. So the arguments in the proof of Theorem 2 are valid in characteristic 2. This means that we can set $p = 2$ in the proof of the Lemma 8 and cancel the logarithmic factor. Finally, we get Robson's bound $\nu(w) = O(\ell^{1/2})$.

Acknowledgments

Authors are grateful to A. Kanel-Belov, V. Leontiev, A. Romashhenko and S. Tarasov for stimulating discussions and help with references. Also we are thankful to unknown referees for their detailed reviews which help to improve the text.

The work of the first author was supported by the RFBR grant 08–01–00414 and the work of the second author also was supported by RFBR grant 09–01–00709 and the grant NS5294.2008.1.

References

1. Bach, E., Shallit, J.: Algorithmic number theory. In: Efficient algorithms, vol. 1. MIT Press, Cambridge (1996)
2. Currie, J., Petersen, H., Robson, J.M., Shallit, J.: Separating words with small grammars. J. Automata, Languages, and Combinatoric 4, 101–110 (1999)
3. Krasikov, I., Roditty, Y.: On a reconstruction problem for sequences. J. Comb. Theory, Ser. A. 77(2), 344–348 (1997)
4. Leontév, V.K., Smetanin Yu, G.: Problems of information on the set of words. Journal of Mathematical Sciences 108(1), 49–70 (2002)
5. Leontév, V.K., Smetanin Yu, G.: Recognition Model with Representation of Information in the Form of Long Sequences. Pattern Recognition and Image Analysis: Advances in Mathematical Theory and Applications 12(3), 250–263 (2002)
6. Lyndon, R.C., Shupp, P.E.: Combinatorial group theory. Springer, Heidelberg (2001)
7. Robson, J.M.: Separating strings with small automata. Information Processing Letters 30, 209–214 (1989)
8. Robson, J.M.: Separating words with machines and groups. Informatique théorique et Applications 30, 81–86 (1996)
9. Smetanin Yu, G.: Refined logarithmic bound for the length of fragments ensuring the unambiguous reconstruction of unknown words. Pattern Recognition and Image Analysis: Advances in Mathematical Theory and Applications 11(1), 100–102 (2001)
10. Vinogradov, I.M.: Elements of Number Theory. Dover Publications, Mineola (2003)

Time Optimal d-List Colouring of a Graph*

Nick Gravin

St.Petersburg Department of Steklov Mathematical Institute RAS.
Nanyang Technological University of Singapore
ngravin@gmail.com

Abstract. We present the first linear time algorithm for d-list colouring of a graph—i.e. a proper colouring of each vertex v by colours coming from lists $\mathcal{L}(v)$ of sizes at least $deg(v)$. Previously, procedures with such complexity were only known for Δ-list colouring, where for each vertex v one has $|\mathcal{L}(v)| \geq \Delta$, the maximum of the vertex degrees. An implementation of the procedure is available.

1 Introduction

Graph colouring is probably the most popular subject in graph theory. Although even the 3-colourability of a graph is known to be an NP-complete problem, there is a number of positive results dealing with the case where the number of colours is close to Δ — the maximal degree of a graph. The most famous among them is the Brooks's theorem [5] asserting that a connected graph that is neither a clique nor an odd cycle, can be coloured with Δ colours. However, the question becomes much harder even for $\Delta-1$ colours. Borodin and Kostochka conjectured in [4] that for $\Delta \geq 9$ the graphs with no clique of size Δ are $(\Delta-1)$-colourable. The conjecture remains open; for $\Delta \geq 10^{14}$ it was affirmed by Reed in [15] by means of the probabilistic method.

Another extension of the Brooks's theorem in the direction of list-colourings was made independently by Vizing [19] and Erdős et al. [7]. Recently this line of research became even more popular than simple colourings. For example it is studied in depth in [9,18,13,11] and many other sources, e.g. as recently as [10]. In 1994 at a graph theory conference held at Oberwolfach Thomassen pointed out that one can also prove a choosability version of Gallai's result [8], which characterized the subgraphs of k-colour-critical graphs induced by the set of all vertices of degree $k-1$. Surprisingly, as Kostochka in [12] has mentioned, this theorem can be easily derived from the result of Borodin [2,3] and Erdős et al. [7]. Specifically, Erdős has characterised all d-choosable graphs as follows. The non-d-choosable graphs are exactly the graphs, whose blocks are cliques or odd cycles. So if a graph possesses a block that is neither a clique nor an odd cycle, it can be coloured for every d-list-assignment. This characterisation does not treat the case of colouring of a non-d-choosable graph for a given d-list-assignment.

* Partially supported by RFBR grant 09-01-12137-ofi_m, the president of Russia grant "Leading Scientific Schools" NSh-4392.2008.1.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 156–168, 2010.

The latter was addressed by Borodin in little-known [2], covering the case when we are given a non-d-choosable graph and a d-list-assignment.

The present paper focused on a linear-time algorithm for the d-list colouring. We also present a proof of the Borodin's result as a corollary of our algorithmic result.

Algorithmic aspects. Although the general list-colouring problem is a hard problem even to approximate, there is a linear time algorithm by Skulrattanakulchai [16] for any Δ-list-assignment, which improves the Lovász's algorithm [14] of such complexity for Δ-colouring. The algorithm of Skulrattanakulchai emerged from a new proof of a theorem of Erdős, Rubin and Taylor [7]. Unlike the latter, we rely upon a simple idea of recursive deletion of vertices presented in the proof by Dirac and Lovász [1] of Brooks Theorem [5]. It allows us to present the first linear time algorithm for d-list colouring problem. Since d-list colouring obeys a simple rule of reduction, our procedure can be easily applied to the problem of partial d-list colouring completion and thus to the completion of a partial Δ-list colouring. So our work generalises, in a number of directions, the result of [16] where the first time optimal algorithm for Δ-list colouring is presented. We describe, as a part of the main algorithm, a linear-time procedure for Δ-colouring adapted for usage in our main algorithm, and which also can be applied to general Δ-colouring problems. It also yields an algorithmic proof for the Brooks's Theorem. Our procedure appears to be quite practical.

An implementation of the entire algorithm together with the procedure can be found in [20]. Finally, we remark that our work can be considered as an algorithmic, and rather nontrivial, counterpart to Borodin's result [2].

2 Preliminaries

For a graph G, a *proper colouring* or just a *colouring* of G is a map $c : V(G) \to \mathbb{N}$ such that $c(v) \neq c(u)$ whenever $uv \in E(G)$. It is called a k-colouring if at most k colours were used. It is natural to consider a more restricted setting, where the allowed colours are prescribed for each vertex.

Formally, a *list-assignment* is a function $\mathcal{L}$ assigning to every vertex $v \in V(G)$ a set $\mathcal{L}(v)$ of natural numbers, which are called *admissible colours* for v. An $\mathcal{L}$-*colouring* of G is a proper colouring c of G which assigns to any vertex v a colour $c(v) \in \mathcal{L}(v)$. If $|\mathcal{L}(v)| \geq k$ for every $v \in V(G)$, then $\mathcal{L}$ is a k-*list-assignment*. If $|\mathcal{L}(v)| \geq deg(v)$ for every $v \in V(G)$, then $\mathcal{L}$ is a d-*list-assignment*. A *partial* $\mathcal{L}$-*colouring* of G of a vertex subset S is an $\mathcal{L}$-colouring of the induced subgraph $G(S)$. An $\mathcal{L}$-colouring $\mathcal{C}$ of G is a *precolouring extension* of a partial colouring $\mathcal{C}'$ on the set S if $\mathcal{C}(v) = \mathcal{C}'(v)$ for all $v \in S$.

Thus, we are given a graph and a list-assignment function $\mathcal{L}$, and the task is either to find a proper $\mathcal{L}$-colouring, or to establish its nonexistence. In a precolouring extension of $\mathcal{L}$-colouring problem we are given, in addition to G and $\mathcal{L}$, a precoloured set $S \subset V$ with a proper colouring on $G(S)$, and the task is to extend this $\mathcal{L}$-colouring to the all vertices of G or to find that it is impossible.

Graph is called d-*choosable* if for any d-list assignment function $\mathcal{L}$ the $\mathcal{L}$-*colouring Problem* has a solution.

2.1 Terminology

For a graph $G = (V, E)$ let $E = E(G)$ denote the set of edges of G and $V = V(G)$ denote the set of vertices of G. For a subset S of vertices, $G(S)$ denotes the induced subgraph of G on S. In this paper we consider only finite simple graphs, i.e. those without loops and multi-edges. We denote by $deg_G(u)$, or $deg(u)$, the degree of u in G, i.e. the number of edges incident with u. Let us from the very beginning assume that G is connected, since the colouring problem can be solved separately for each component of G.

A *cut* vertex is a vertex after deletion of which the graph splits into more than one connected component. A two-connected graph is a graph which remains connected after deletion of any vertex (here we assume that an empty graph is connected). A *block* of G is a maximal by inclusion two-connected induced subgraph of G. Let us recall some facts about blocks and cut vertices, which one can find in [1] or in [6]:

- Two blocks can have at most one common vertex and if they have one then it is a cut vertex of G.
- Any edge of G belongs to exactly one block.
- A bipartite graph with the first part being the set of blocks of G and the second one being the set of cut vertices of G such that every cut vertex is adjacent to the blocks which contain it, is a tree.

We call such a tree of blocks and cut vertices a *block tree*.

Suppose we have a spanning tree T of G with a root r, and an edge orientation towards the root. Such a tree is called *normal* if there is an oriented path in T joining any two $u, v \in V$ such that $(u, v) \in E$. In [6] it is proved that every connected graph contains a normal tree. One can easily check that the depth-first search spanning tree is a normal tree. Using this fact one can easily see that the root of normal spanning tree of two-connected graph is a leaf in this tree, since otherwise by deleting the root we will get more than one connected component.

2.2 Main Results

Before discussing the algorithm, let us describe some natural requirements on the input data. In the present paper we consider the most concise graph representation, i.e. the vertices of the graph are represented by integers from 1 to $N = |V|$, and $E(G)$ is given as a set of pairs.

Definition 1. *Let the maximum value of the colour for $\mathcal{L}$ be at most $c|E(G)|$, for a constant c. Then we say that $\mathcal{L}$ satisfies the colour density condition.*

One needs the latter condition to efficiently handle colours. It is not restrictive, since it allows one to describe all possible essentially different situations for $\mathcal{L}$ up to renumbering of the colours. The same condition on the maximal colour value and numeration of vertices was imposed in [16].

Theorem 1. *There is a linear time algorithm which, for a given graph G and d-list-assignment function $\mathcal{L}$ satisfying the colour density condition, either finds an $\mathcal{L}$-colouring of G, or reports its nonexistence.*

Our algorithm is effectively based on the following theorem characterising all the list assignments and graphs which have an admissible assignment.

Theorem 2. *Let $G = (V, E)$ be a graph, with the set of blocks $\mathcal{B}$, and $\mathcal{L}$ a d-list-assignment for G. Then G has no $\mathcal{L}$-list colouring if and only if:*

1. *Every $B \in \mathcal{B}$ is either a clique or an odd cycle.*
2. *There exists a map $\mathcal{S}$ which assigns to each pair (v, B), where $v \in B \in \mathcal{B}$, a set of colours $\mathcal{S}(v, B)$ satisfying the following conditions:*
   (a) $\bigcup_{B:\ v \in B \in \mathcal{B}} \mathcal{S}(v, B) = \mathcal{L}(v)$
   (b) $\mathcal{S}(v, B) \bigcap \mathcal{S}(v, B') = \emptyset$ *for any* $B \neq B' \in \mathcal{B}$ *with* $v \in B \cap B'$.
   (c) $|\mathcal{S}(v, B)| = deg_{G(B)}(v)$ *for any* $v \in B$
   (d) $\mathcal{S}(v, B) = \mathcal{S}(u, B)$ *for any* $u, v \in B$

Remark 1. Conditions 2a and 2c of the theorem imply condition 2b; the condition 1 is just the Erdős' characterisation of the d-choosable graphs. Note that the appearance of cliques and odd cycles is not a coincidence, due to Brooks Theorem [5], that says that a connected graph H that is neither a clique nor an odd cycle satisfies $\chi(H) \leq \Delta(H)$.

Remark 2. This theorem is a reformulation of the Borodin's result [2]. One more variant of the Borodin's result one can find in [12].

In other words, the theorem states that non-colourable instances of d-list colouring problem can be described as follows. In the block tree of G we can split every list of a cut vertex v into t disjoint parts and leave only one such part for a block containing v (each part should corresponds to exactly one block) in such a way, that now it is impossible to find a proper colouring of any block using only the lists of admissible colours of its vertices (for each cut vertex in the block we have to use the part of the list we have left for these block and vertex). (See Fig. 1)

Keeping in mind this characterisation while reading the algorithm description may make the latter more transparent. The algorithm for Theorem 1 is described in Section 3.

3 d-List Colouring Problem

INPUT: A graph $G = (V, E)$ and list-assignment $\mathcal{L}$ with $\mathcal{L}(v) \geq deg_G(v)$ for every vertex $v \in V$.

TASK: Find a proper $\mathcal{L}$-colouring of G or deduce that there is none.

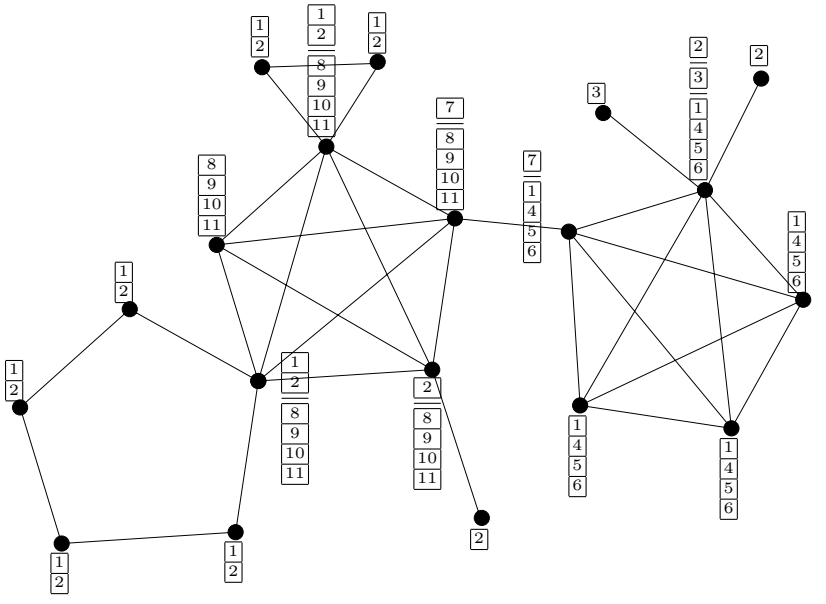


Fig. 1. A graph and $\mathcal{L}$-assignment that does not lead to a $\mathcal{L}$-colouring. The colour list of each cut vertex is split into a number of lists according to the statement of Theorem 2.

To simplify the presentation, let G be a connected graph. However, almost the same algorithm works for not necessarily connected graphs.

Our algorithm, essentially, exploits a simple observation, that if G has a vertex v with $|\mathcal{L}(v)| > deg_G(v)$, then we can easily colour the whole graph. In the algorithm we use some generalisation of this idea: we are trying to find a vertex u and its admissible colour, such that, given u coloured with that colour the remaining graph $G - u$ will be connected and posses a vertex v with $|\mathcal{L}(v)| > deg_{G-u}(v)$.

The working scheme is as follows. At the beginning, we find a block tree T in G. At the next step we take a block B corresponding to the leaf in T. Using the tool mentioned above we either colour the whole graph, or find a colouring of B, except the corresponding cut vertex v_0 (here B should be an odd cycle or a clique). In the last case we remove $B \setminus \{v_0\}$ from G, as well as all colours of v_0's neighbours from $\mathcal{L}(v_0)$. We repeat this step until we either colour the graph, or discover a vertex with an empty list of colours.

3.1 Algorithm

Let us give a few definitions before we start describing an algorithm.

Definition 2. *We call a block a leaf block if it contains at most one cut vertex, i.e. it corresponds to a leaf vertex in the block tree $\mathcal{T}$.*

Definition 3. *We call an ordered pair of two adjacent vertices* (u, v) *distinguished by the colour* i *if* $i \in \mathcal{L}(u) \setminus \mathcal{L}(v)$.

In the main algorithm we will use the following auxiliary procedures:

1. **Recursive deletion of vertices.** It finds a desired d-colouring of a connected graph G, if G has a vertex v with $|\mathcal{L}(v)| > deg_G(v)$.
2. **Search for blocks of** G**.** It finds the representation of each block by the set of its vertices.
3. **Search for a colour distinguishing the pair** (u, v)**.** It requires us to introduce one boolean array $\mathcal{C}$ of the size $c|E(G)|$, which we reuse by every application of the procedure.
4. **Δ-colouring of a connected graph.** For a connected graph H that is neither a clique nor an odd cycle, the procedure finds a proper Δ-colouring.

Now we are ready to describe the algorithm.

- **STEP 1.** Suppose there exists $v \in V$ with $|\mathcal{L}(v)| > deg_G(v)$. The **recursive deletion of vertices** will give us a colouring we seek and we terminate the algorithm.
- **STEP 2. A)** Find all blocks of G and the corresponding block tree $\mathcal{T}$.
  1. Find the blocks of G using **search for blocks of** G.
  2. For each vertex v of G find a list of the blocks containing v.
  3. Construct a bipartite graph H_0 with one part consisting of all blocks and another one consisting of all cut vertices and with $E(H_0) = \{(v, B) \mid v \in B\}$.
  4. Construct a block tree $\mathcal{T}$ as the *breadth-first search tree* of H_0, i.e. find the corresponding order $\mathcal{B}$ on the blocks and cut vertices.
- **B)** Take the last block B in the order $\mathcal{B}$ and its unique cut vertex v_0. In the next steps we are trying to reduce G by cutting off $B \setminus \{v_0\}$, or find a desired colouring.
- **STEP 3.** Try to find in B a pair of distinguished by a colour vertices (u, v), such that u is not a cut vertex, i.e. $u \neq v_0$.
  1. If there is such a pair in B, say (u, v) distinguished by the colour c, then we colour u with c and delete it from G. In the remaining graph, $|\mathcal{L}(v)| > deg(v)$. Also the graph $G' = G \setminus \{u\}$ is connected. We colour G' by the **recursive deletion of vertices** and terminate.
  2. Otherwise we proceed to the next step.
- **STEP 4.** Check whether the block B is a clique or an odd cycle. Can be done by pre-computing vertex degrees of G.
  1. If B is a clique or an odd cycle, we reduce G by cutting off $B \setminus \{v_0\}$. Since in STEP 3 v_0 and a vertex v adjacent to v_0 are not distinguished by a colour, we have $\mathcal{L}(v_0) \supseteq \mathcal{L}(v)$. We reduce $\mathcal{L}(v_0)$ by $\mathcal{L}(v)$. Then we return to the STEP 2 B) with next after B in $\mathcal{B}$ block. Here we use a property of order $\mathcal{B}$ to reduce efficiently $\mathcal{L}(v_0)$.
  2. Other possibilities for B. In this case we necessarily get a desired colouring and terminate.

- Colour $G(V \setminus V(B))$ by applying the **recursive deletion of vertices** in $G(V \setminus B)$.
- Delete from $\mathcal{L}(v_0)$ all the colours which are present among neighbours of v_0.
- Check if v_0 has the same as the other vertices in B set of admissible colours.
- If v_0 does not, then for the remaining not coloured vertices a pair (v_0, u), where u is any vertex adjacent to v_0 in B, is distinguished by a colour. We complete the colouring of the whole graph as in STEP 3.1 and terminate.
- Otherwise we apply **Δ-colouring** procedure to B.

3.2 Implementation of Procedures

Here we give a description of our auxiliary procedures in details together with their complexity analysis.

The recursive deletion of vertices. Suppose there exists $v \in V$ with $|\mathcal{L}(v)| > deg_G(v)$. Put v in a stack $\mathcal{O}$ and delete it from the graph.

Then the remaining graph again should have a vertex with the same property, since G is a connected graph and $|\mathcal{L}(u)| \geq deg(u)$ for any $u \in V$. We repeat the previous action for this vertex, i.e. put it in $\mathcal{O}$ and delete it from G, and keep doing such deletion until there are no vertices left in G.

Then we take the first vertex from the $\mathcal{O}$ and colour it with any admissible colour, then take the second one and again colour it with any admissible colour which does not occur among its already coloured neighbours and so on (we succeed to colour every such vertex because we have appropriately chosen vertices in the $\mathcal{O}$). Eventually we colour all the vertices of G.

Remark 3. This procedure also works well for graphs which have more than one connected component if in each such component there is a vertex w with $|\mathcal{L}(w)| > deg(w)$.

Proposition 1. *The recursive deletion of vertices in G and subsequent colouring, works in linear in the size of the graph time.*

Proof. We can compute the degrees of all vertices of G in linear time. Indeed, after putting any vertex in $\mathcal{O}$, any of its neighbours can be put in $\mathcal{O}$ immediately after it. So we can make a depth-first search starting with a vertex, which has degree less than its list size, and put in $\mathcal{O}$ corresponding vertex at each step of that search. All of this works in linear time.

To colour all the vertices in $\mathcal{O}$ in linear time it is sufficient to colour each vertex $u \in O$ in $O(|\mathcal{L}(u)|)$ time.

As we have the bound on the maximal colour value we can create in linear time, during the initialisation of the algorithm, an array $\mathcal{C}_0$ of boolean values of size equal to maximum value among all possible colours and with initial False values.

At the colouring step of the procedure for a vertex u we match in $\mathcal{C}_0$ with True the colours of all already coloured neighbours of u and then search in $\mathcal{L}(u)$ for a colour c with False value of $\mathcal{C}_0[c]$. When we have found c, we reset back to False, by one pass along the neighbours of u, all the values of $\mathcal{C}_0$. Thus we return $\mathcal{C}_0$ to the initial state and we can use it again for other vertices. Then we colour u with c. This concludes the proof, since we use only $O(|\mathcal{L}(u)|)$ time to colour u; also in total we use only $c|E(G)|$ of space, since we create the array only once.

Search for blocks of G. The algorithm for finding all blocks in linear time can be found, e.g. in [17].

Search for a colour distinguishing the pair (u, v). As we have the bound on the maximal colour value we can in linear in $|\mathcal{L}|$ time create, during the initialisation of the algorithm, an array $\mathcal{C}$ of boolean values of size equal to the maximum value among all possible colours and with initial False values. Then we can mark with True value all the colours which are in $\mathcal{L}(v)$ and then verify whether there is a colour c in $\mathcal{L}(u)$ for which $\mathcal{C}[c]$ is False, and finally reset all the True values of $\mathcal{C}$ back to False. Thus we either find a distinguishing colour or check that there are no such a colour, in linear in $|\mathcal{L}(u)| + |\mathcal{L}(v)|$ time.

Δ-colouring of a connected graph. We use a linear time algorithm for Δ-colouring by Lovász [14]. But there is another, easier, way to deal with the Δ-colouring and we will describe it in Section 4, which also provides a new constructive proof of the Brooks Theorem.

3.3 Proof of Correctness of the Algorithm

STEP 1. We compute all vertex degrees in G and check for each vertex whether $|\mathcal{L}(v)| > deg_G(v)$.

STEP 2.

A) Find all blocks of G and the corresponding block tree $\mathcal{T}$. We consider the representation of each block by the set of its vertices; also we construct for each vertex a list of all blocks it contains. To recall the definition of block tree, one can think of a bipartite graph H with one part consisting of all blocks and another consisting of all cut vertices (that is vertices which belong to at least two blocks) and with $E(H) = \{(B, v) \mid v \in B\}$; by construction H is a tree and is called the block tree. For complexity reasons we need to construct $\mathcal{T}$ as the **breadth-first search tree**, i.e. we require that $\mathcal{T}$ has the specific order on blocks and cut vertices.

Remark 4. We have to do the **search for blocks** as the unsolvable instances of the problem are described in terms of certain conditions on blocks [7].

For tree representation it is useful to view an orientation towards the root. So we add an artificial vertex to G and join it to a vertex of G. Thus we add an artificial block to $\mathcal{T}$. Let us take this artificial block B_0 to be a root of $\mathcal{T}$. Let us pick a block B_0 and consider the order $\mathcal{B}$ in which blocks appear in a breadth-first search on $\mathcal{T}$ with the root in B_0.

B) Let B be the last block of $\mathcal{T}$ in $\mathcal{B}$.

Note that B is a leaf block. Let us denote by v_0 the cut vertex of B, i.e. the unique cut vertex in $\mathcal{T}$ joined with B. Since B is the last in $\mathcal{B}$, and $\mathcal{B}$ is the order of a breadth-first search, all the other neighbours of v_0 that appear after v_0 in $\mathcal{B}$, are leaf blocks.

STEP 3.

Case A. Suppose we have found a pair of vertices (u, v) distinguished by the colour c, such that u is not a cut vertex in B.

Then we colour u with c and delete it from G. In the remaining graph, $|\mathcal{L}(v)| > deg(v)$. Also the graph $G' = G \setminus \{u\}$ is connected. So we can colour G' by the recursive deletion of vertices.

Case B. If there are no such pairs then for all non-cut vertices $w, v \in B$ we have $\mathcal{L}(v) = \mathcal{L}(w)$, whereas for the cut vertex v_0 one has $\mathcal{L}(v_0) \supseteq \mathcal{L}(v)$. Consequently, all the vertices of $B \setminus \{v_0\}$ have the same degree. Then we come to the STEP 4.

At the next step of the algorithm we either reduce G by cutting off $B \setminus \{v_0\}$ or find a colouring of G by recursive deletion of vertices.

STEP 4. All lists for non cut vertices of B are the same and that one is contained in the list of cut vertex of B.

According to the Brooks Theorem [5] (cf. Remark 1) there are two cases. (That can be distinguished by pre-computing vertex degrees)

Case I. (B is either a clique or an odd cycle)

We need to make the reduction of $\mathcal{L}(v_0)$ very carefully, since $|\mathcal{L}(v_0)|$ can be much bigger than the size of B. We describe it in details in Section 3.4.

Case II. (remaining possibilities for B) In this case due to classification by Erdős [7] we know that G should have a proper colouring for any list-assignment function.

3.4 Complexity Analysis of the Algorithm

Clearly each of the Steps 1 and 2 takes only $O(|G| + |\mathcal{L}|)$ time.

STEP 3. Part 1. Search of a distinguished by a colour pair of vertices in $B \setminus \{v_0\}$.

Proposition 2. *Search for a distinguished by a colour pair of vertices (u, v), such that $u, v \neq v_0$, can be done in linear in the block's size time, where by the size of a block we mean the number of edges plus number of vertices in this block.*

Proof. We can consider a spanning depth-first search tree of B with the root in v_0 and check only pairs of admissible colour sets for adjacent vertices in this tree. Since B is a block, v_0 has unique neighbour in the tree. So we do double check for two adjacent vertices u and v twice, at first for the pair (u, v) then for (v, u).

Now let us show that the total time of all checks is linear in $\sum_{v\in B\setminus\{v_0\}} |\mathcal{L}(v)|$. By every check of u and v which are not cut vertices we either get that $\mathcal{L}(u) = \mathcal{L}(v)$, or find a pair of distinguished by a colour vertices together with such colour. In the latter case we terminate the search of the pair. In the former case we see that the check works in linear in $min(|\mathcal{L}(u)|, |\mathcal{L}(v)|)$ time. So the total time will be linear in $\sum_{v\in B\setminus\{v_0\}} |\mathcal{L}(v)|$.

Part 2. Check whether a pair (u_0, v_0) is distinguished by a colour in linear in $\mathcal{L}(u_0)$ time, where u_0 is adjacent to v_0 in B.

Remark 5. Since B is a block and we consider the depth-first search spanning tree with the root in v_0, we need only to make one check for v_0 and its unique neighbour u_0 in the tree. But we cannot allow ourselves doing this check even in linear in $|\mathcal{L}(v_0)| + |\mathcal{L}(u_0)|$ time in straightforward manner for all blocks pending at v_0. Since $|\mathcal{L}(v_0)|$ may be much bigger than the size of B and furthermore v_0 may be the cut vertex for many small blocks like B, the total time can be quadratic in input size, e.g.for $G = K_{n-1,1}$. For the same reasons we are not allowed to make a straightforward reduction of $\mathcal{L}(v_0)$ in Step 4.

To avoid multiple checks described in Remark 5 and reductions of $\mathcal{L}(v_0)$ for large $\mathcal{L}(v_0)$ we can remember the True values of $\mathcal{L}(v_0)$ in $\mathcal{C}$ and use it not for one pending at v_0 block but for all such blocks. So strictly speaking we do the following:

- Introduce at the beginning of the algorithm a new array $\mathcal{C}'$ with $|\mathcal{C}'| = |\mathcal{C}|$ of boolean, initially marked with False values.
- If the cut vertex of a leaf block B differs from the previous one, say v_0', then we do the following.
  1. Revert $\mathcal{C}'$ to initial state of only False values in $|\mathcal{L}(v_0')|$ time, just going once along $\mathcal{L}(v_0')$.
  2. Mark in $\mathcal{C}'$ by True values the colours of $\mathcal{L}(v_0)$.
- For a leaf block B with the cut vertex v_0 check the pair (u_0, v_0) in linear in $|\mathcal{L}(u_0)|$ time. One can do it by one passing along $\mathcal{L}(u_0)$ and checking if $\mathcal{C}'[c]$ is True for $c \in \mathcal{L}(u_0)$.

For the search of a distinguishing colour for the pair (u_0, v_0) it suffice to use $\mathcal{C}'$ instead of $\mathcal{L}(v_0)$, since in the search we only check whether a colour $i \in \mathcal{L}(u_0)$ belongs to $\mathcal{L}(v_0)$.

STEP 4.

Effective reduction of $\mathcal{L}(v_0)$. Since we do not need the whole list $\mathcal{L}(v_0)$, while v_0 is a cut vertex, at any reduction we can efficiently maintain only $\mathcal{C}'$ instead of $\mathcal{L}(v_0)$. Thus at every reduction of $\mathcal{L}(v_0)$ we just mark back with False all the colours of $\mathcal{L}(u_0)$ in $\mathcal{C}'$ and do not change $\mathcal{L}(v_0)$ yet. We will really change $\mathcal{L}(v_0)$ only when we cut off all the leaf blocks pending at v_0 and when v_0 becomes a

non-cut vertex. By definition of $\mathcal{B}$ we will do such change only once for v_0, so the total time of reducing lists for all cut vertices will take a linear time. After such "fake" reduction of $\mathcal{L}(v_0)$ we colour all vertices in $B \setminus \{v_0\}$ and delete them from G. Then we go back to Step 2 B) and find other leaf block in the reduced graph.

Thus Step 3 and Step 4 work in $O(|B|)$ time for each particular block B and to maintain $\mathcal{C}'$ for each cut vertex in total takes $O(|\mathcal{L}|)$ time. Adding up all above together, we see that the algorithm works in $O(|G| + |\mathcal{L}|)$ time.

4 Constructive Proof of the Brooks Theorem

Δ-Colouring of a two-connected graph which is neither a clique nor an odd cycle. We describe an algorithm different from Step 4 III. Here we may suppose that all vertices in $B \setminus \{v_0\}$ (v_0 is the unique cut vertex in B) have the same list of admissible colours. Hence all the vertices in $B \setminus v_0$ have the same degree.

To this moment we know that B is a block, i.e. two-connected graph. Moreover we have found a depth-first search spanning tree for B (it is just a part of depth-first search tree for G). And also we have described a procedure of *recursive deletion of vertices*. So using all of this we can now describe a quite simple algorithm for colouring of G. The only difficult place in this algorithm is the number of combinatorial cases we are considering. So the description of the algorithm will consist of working out these cases, as follows.

- Suppose we have found two non-adjacent vertices, say u and v, in $B \setminus \{v_0\}$ such that if we do the following:
  - Colour u and v with the same colour.
  - Delete this colour from $\mathcal{L}(w)$ for each neighbour w of v or u.
  - Delete both u and v from G,

  then the remaining graph have a *recursive deletion of vertices*. We call such vertices u and v a *good pair*. Thus if we have found a good pair then we can easily colour G in linear time.
- If $deg_G(u) = 2$ for an $u \in B \setminus \{v_0\}$, then B is a cycle and we know that B is not an odd cycle. So it is even. Let us colour $B \setminus \{v_0\}$ then delete from $\mathcal{L}(v_0)$ the colour of its two neighbours in B and then colour the remaining graph, since it has *recursive deletion of vertices*.
- Two descendant neighbours of a vertex u in the depth-first search spanning tree, i.e. normal tree, with v_0 as a root, are a *good pair*. Indeed, one can easily derive that these two descendants are not adjacent from the fact that a tree with root v_0 is normal. Also one can see that if we delete from B these two descendants, then in the remaining graph there always will be a path from u to any other vertex. These two things show the goodness of the pair.
- If our depth-first search spanning tree is not a path, then we can efficiently, i.e. in linear in block's size time, find such a pair. In this case we are done.

– So our tree is just a path $w_0, w_1, w_2, \ldots, w_{k-1}, v_0$, and, as above, v_0 is the unique cut vertex of B for G. For convenience let $v_0 = w_k$. Consider the closest to w_0 vertex, say w_t, on the path, which is not adjacent to w_0, and so $deg_G(w_0) \geq t - 1$.

1. The pair (w_0, w_t) is *good* when either there is a vertex among $w_1, w_2, \ldots, w_{t-1}$ which is adjacent to something else than $w_0, w_1, \ldots, w_t$, or $t+1 \leq k$ and w_{t+1} is adjacent to w_0.
   This is true, as w_0 and w_t are not adjacent and there cannot be more than two connected components in the graph $B \setminus \{w_0, w_t\}$, since w_0 is a leaf in the normal tree (the vertices $w_1, w_2, \ldots, w_{t-1}$ will be connected in $B \setminus \{w_0, w_t\}$ and so will be v_0 with remaining vertices).
2. Suppose $deg_G(w_0) \geq t + 1$, then w_1 is adjacent to a w_l with $l \geq t + 1$, since $deg_G(w_1) = deg_G(w_0) \geq t + 1$. By case 1 above, we have found a good pair.
3. If $t-1 = deg_G(w_0)$, then w_{t-1} is not adjacent to some w_r with $r < t-1$, as $deg_G(w_0) = deg_G(w_{t-1})$ and w_{t-1} is adjacent to w_t. Hence w_r and w_{t-1} are a *good pair*, since w_r and w_{t-1} are both adjacent to w_0 and w_t.
4. Suppose $t = deg_G(w_0)$ and that we are not in case 1. Hence $w_0, w_1, \ldots, w_{t-1}$ and $w_1, \ldots, w_t$ form cliques, as $t = deg_G(w_0) = \cdots = deg_G(w_t)$ and no w_i with $1 \leq i \leq t-1$ is adjacent to w_j with $j \geq t+1$. Also $k \geq t+2$, as w_0 is not adjacent to w_t and w_{t+1}, but there is a vertex w_i adjacent to w_0 with $i > t$. We can assume $i > t+1$, since the possibility, when $i = t+1$, we have considered in the case 1. Thus, as $deg_G(w_t) = deg_G(w_0) > 2$, w_{t-1} and w_{t+1} are a *good pair*.

Remark 6. The same algorithm works for the Δ-colouring of a 2-connected graph H. Indeed, we can take any vertex of H as a cut vertex v_0 and we can easily colour H in linear in the size of H time if there is a vertex u with $deg_H(u) < \Delta(H)$.

Remark 7. This algorithm also provides a proof of Brooks Theorem [5] for two-connected graphs, since if the graph is not a clique or an odd cycle we can apply the algorithm and get the proper colouring.

Acknowledgement. We thank Oleg Borodin and Alexander Kostochka for rather helpful pointers to the literature.

References

1. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. Elsevier, New York (1976)
2. Borodin, O.V.: A chromatic criteria for degree assignment. In: IV USSR conference on the theoretical cybernetics problems, in Novosibirsk, proceeding reports, pp. 127–128 (1977) (in Russian)
3. Borodin, O.V.: Problems of colouring and of covering the vertex set of a graph by induced subgraphs, Ph.D. Thesis, Novosibirsk State University, Novosibirsk (1979) (in Russian)

4. Borodin, O.V., Kostochka, A.V.: On an upper bound on a graph's chromatic number, depending on the graphs's degree and density. J. Comb. Th. (B) 23, 247–250 (1977)
5. Brooks, R.L.: On colouring the nodes of network. Proc. Cambridge Phil. Soc. 37, 194–197 (1941)
6. Diestel, R.: Graph Theory. Springer, Heidelberg (2000)
7. Erdős, P., Rubin, A.L., Taylor, H.: Choosability in graphs. In: Proc. West-Coast Conf. on Combinatorics, Graph Theory and Computing, Arcata, California, Congr. Numer., vol. XXVI, pp. 125–157 (1979)
8. Gallai, T.: Kritische Graphen I. Publ. Math. Inst. Hung. Acad. Sci. 8, 373–395 (1963)
9. Jensen, T.R., Toft, B.: Graph Colouring Problems. John Wiley & Sons, New York (1995)
10. Kawarabayashi, Ken-ichi, Mohar, B.: List-color-critical graphs on a fixed surface. In: SODA '09: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms, New York, pp. 1156–1165 (2009)
11. Kostochka, A., Stiebitz, M.: A list version of Dirac's Theorem. J. Graph Th. 39 (2002)
12. Kostochka, A., Stiebitz, M., Wirth, B.: The colour theorems of Brooks and Gallai extended. Discrete Math 162, 299–303 (1996)
13. Kratochvíl, J., Tuza, Z., Voigt, M.: New trends in the theory of graph colorings: choosability and list coloring. DIMACS Ser. Discrete Math. Theoret. Comput. Sci, vol. 49, pp. 183–197. Amer. Math. Soc., Providence (1999)
14. Lovász, L.: Three short proofs in graph theory. J. Comb. Th. (B) 19, 269–271 (1975)
15. Reed, B.: A strengthening of Brooks' theorem. J. Comb. Th. (B) 76, 136–149 (1999)
16. Skulrattanakulchai, S.: Δ-List Vertex Coloring in Linear Time and Space. Information Processing Letters 98(3), 101–106 (2006)
17. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM J. Comput. 1(2), 146–160 (1972)
18. Tuza, Z.: Graph colorings with local constraints—a survey. Discussiones Math. Graph Th. 17(2), 161–228 (1997)
19. Vizing, V.: Colouring the vertices of a graph in prescribed colours. Metody Diskret. Anal. v Teorii Kodov i Schem 29, 3–10 (1976) (in Russian)
20. http://www1.spms.ntu.edu.sg/~ngravin/code/Dcols.tgz

The Cantor Space as a Generic Model of Topologically Presented Knowledge

Bernhard Heinemann

Fakultät für Mathematik und Informatik,
FernUniversität in Hagen,
58084 Hagen, Germany
bernhard.heinemann@fernuni-hagen.de

Abstract. We prove a new completeness theorem for topologic, a particular system for reasoning about knowledge and topology. In fact, we show that topologic is complete with respect to the Cantor space, i.e., the set of all infinite 0-1-sequences endowed with the initial segment topology. To this end, we make use of the connection between the semantics of topologic and McKinsey and Tarski's topological interpretation of the modal box operator, as well as of Georgatos' Normal Form Lemma.

Keywords: reasoning about knowledge, topological reasoning, the Cantor space, modal logic, subset spaces, bisimulations.

1 Introduction

It is known since McKinsey and Tarski, [12], that not only the modal system S4 is sound and complete for topological spaces in general, but also for the real line $\mathbb{R}$ endowed with the standard topology. (In fact, S4 is sound and complete with respect to every dense-in-itself metric separable space.) Thus, a certain *spatial aspect* appeared very early in the history of modal logic, even before the nowadays ubiquitous Kripke semantics was established. Due to the need for a rigorous treatment of space in computer science and, in particular, artificial intelligence, that spatial dimension inherent to modal logic was recently rediscovered and more thoroughly investigated. The reader is referred to the handbook [2] for an overview of the modern developments in this area.

A new proof of McKinsey and Tarski's theorem is given in [1]. Quasi as a starter, the authors of that paper show that S4 is sound and complete with respect to the Cantor space; see also [13]. We take up again this data structure here and reinforce the just quoted completeness theorem by substituting S4 with topologic.

The system topologic is based on the *modal logic for subset spaces,* which was introduced in the paper [14]; more background information and the corresponding technical details were presented later, in [7]. Here we only recall the underlying bi-modal language, $\mathcal{L}$. Basically, $\mathcal{L}$ comprises a modality K describing the *knowledge* of an agent under discussion, and a second one, $\Box$, measuring qualitatively the *effort to acquire knowledge*. The relevant semantic domains,

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 169–180, 2010.

called *subset spaces,* consist of a non-empty set X of *states (of the world),* a set $\mathcal{O}$ of subsets of X representing the *epistemic states* (or *knowledge states*) of the agent, and a valuation V determining the states where the atomic propositions are true. $\mathcal{L}$-formulas are interpreted in subset spaces with respect to *neighborhood situations* x, U, for which $x \in U \in \mathcal{O}$ is valid by definition. The operator K quantifies over all states of some epistemic state $U \in \mathcal{O}$ then, whereas $\Box$ quantifies 'downward' over all $U' \in \mathcal{O}$ that are contained in U, since shrinking and gaining knowledge correspond to each other. Now, the topological aspect appears since a certain approximation procedure in some space of sets is modelled with that. Going the other way round, i.e., interpreting open sets as knowledge states of an agent, yields an epistemic interpretation of topological spaces. Thus, reasoning about knowledge and topological reasoning may be regarded as two sides of the same coin. This is substantiated by the fact that topological spaces can completely be axiomatized through $\mathcal{L}$-formulas, resulting in the system topologic; see Section 2.

We now argue that the Cantor space, $\mathcal{C}$, can serve as a model of knowledge of *binary streams*. Such data processing may be represented by computable functions $g : \mathbb{N} \to \{0, 1\}$, i.e., by certain infinite 0-1-sequences, in an abstract way. $\mathcal{C}$ consists of the set $\mathcal{B}$ of *all* such sequences together with the initial-segment topology $\mathcal{T}$. Thus, it is allowed to call $\mathcal{C}$ a *generic* model for that kind of knowledge. We want to put a finer point on that now, in particular, we want to say to what extent this knowledge is *topologically presented*. Suppose that $g : \mathbb{N} \to \{0, 1\}$ is a given stream, and there is an observer monitoring an appropriate device. The set $\mathcal{B}$ can be illustrated by the full infinite binary tree. Moreover, every node of this tree can be annotated with the basic open neighborhood $U \in \mathcal{T}$ containing all possible prolongations of the initial segment leading to this node. In this way, a *tree of subsets of* $\mathcal{B}$ results. Now, computing the function g yields, step by step, a *bisection* of the actual open set, constituting a new epistemic state of the observer at each time. Thus the knowledge of the observer is reflected in a certain subset of the set of opens of a particular topological space.

At this point, it is natural to ask whether the prominent part the Cantor space plays for the context just described finds expression in the logic as well. We give an affirmative answer in the following. To be precise, we show that the system topologic is sound and complete for $\mathcal{C} = (\mathcal{B}, \mathcal{T})$. This result should underpin the notion of being 'generic' attributed to the Cantor space above.

The technical part of the paper is organized as follows. In the next section, we recall the basics of topologic. Afterwards, the peculiarities of this system required in the course of this paper are arranged in a way that proves to be useful for us later on. This is done in Section 3 and Section 4. In Section 5, we consider the notion of bisimulation for both topological spaces and subset spaces in general. This facilitates the proof of our main result, which is stated in Section 6. The paper is concluded with a summing up and some additional remarks, in particular, on the completeness of topologic with respect to $\mathbb{R}$.

Finally, two recent generalizations of McKinsey and Tarski's theorem that are different from ours should be quoted here; see [3] and [10].

2 Prerequisites

In this section, we recapitulate the language underlying topologic as well as some of the basic properties of this system. Doing so, we use [7] as a reference without mentioning this each time.

We first define the syntax of the language. Let $\mathsf{Prop} = \{p, q, \ldots\}$ be a denumerably infinite set of symbols called *proposition variables* (which are to represent the basic facts about the states of the world). Then, the set Form of *formulas* over Prop is defined by the rule $\alpha ::= p \mid \neg\alpha \mid \alpha \wedge \alpha \mid \mathsf{K}\alpha \mid \Box\alpha$. Later on, the boolean connectives that are missing here are treated as abbreviations, as needed. And the duals of the modal operators K and $\Box$ are denoted by L and $\Diamond$, respectively. In accordance with the above, K is called the *knowledge operator* and $\Box$ the *effort operator*. Note that the set of all formulas of usual modal logic, denoted by MF, is obviously a subset of Form. (MF is obtained by omitting the last but one clause of the above generation rule.)

Second, we turn to the semantics. For a start, we define the relevant domains and comment on this definition afterwards. We let $\mathcal{P}(X)$ designate the powerset of a given set X.

Definition 1 (Subset Frames and Subset Spaces)

1. *Let X be a non-empty set (of* states*) and $\mathcal{O} \subseteq \mathcal{P}(X)$ a set of subsets of X such that $\{\emptyset, X\} \subseteq \mathcal{O}$. Then the pair $\mathcal{S} := (X, \mathcal{O})$ is called a* subset frame*.*
2. *Let $\mathcal{S} = (X, \mathcal{O})$ be a subset frame. The set $\mathcal{N}_\mathcal{S} := \{(x, U) \mid x \in U$ and $U \in \mathcal{O}\}$ is called the* set of neighborhood situations *of $\mathcal{S}$.*
3. *Let $\mathcal{S} = (X, \mathcal{O})$ be a subset frame. An* $\mathcal{S}$-valuation *is a mapping $V : \mathsf{Prop} \to \mathcal{P}(X)$.*
4. *Let $\mathcal{S} = (X, \mathcal{O})$ be a subset frame and V an $\mathcal{S}$-valuation. Then, $\mathcal{M} := (X, \mathcal{O}, V)$ is called a* subset space (based on $\mathcal{S}$)*.*

A couple of things should be noted here. First, it will become clear soon that the requirement '$\{\emptyset, X\} \subseteq \mathcal{O}$' from item 1 above is quite natural; see the next definition where the mode of operation of the modalities is fixed. Second, the term 'neighborhood situation' from item 2 is introduced just to denominate the semantic atoms of our language. Note that the first component of such a situation x, U indicates the actual state of the world while the second displays the uncertainty of the agent in question about it. And third, note that $\mathcal{S}$-valuations only depend on states (and are independent of sets thus). This is in accordance with the common modelling practice, eg, in contexts where temporal logics are used.

Now, let a subset space $\mathcal{M}$ be given. We define the relation of satisfaction, $\models_\mathcal{M}$, between neighborhood situations of the underlying frame and formulas from Form. In the following, neighborhood situations are often written without parentheses.

Definition 2 (Satisfaction and Validity). *Let $\mathcal{M} = (X, \mathcal{O}, V)$ be a subset space based on $\mathcal{S} = (X, \mathcal{O})$, and let $x, U \in \mathcal{N}_\mathcal{S}$ be a neighborhood situation of $\mathcal{S}$.*

Then

$$\begin{aligned}
x, U \models_{\mathcal{M}} p &\iff x \in V(p) \\
x, U \models_{\mathcal{M}} \neg\alpha &\iff x, U \not\models_{\mathcal{M}} \alpha \\
x, U \models_{\mathcal{M}} \alpha \wedge \beta &\iff x, U \models_{\mathcal{M}} \alpha \text{ and } x, U \models_{\mathcal{M}} \beta \\
x, U \models_{\mathcal{M}} \mathsf{K}\alpha &\iff \forall\, y \in U : y, U \models_{\mathcal{M}} \alpha \\
x, U \models_{\mathcal{M}} \Box\alpha &\iff \forall\, U' \in \mathcal{O} : (x \in U' \subseteq U \Rightarrow x, U' \models_{\mathcal{M}} \alpha)\,,
\end{aligned}$$

where $p \in \mathsf{Prop}$ *and* $\alpha, \beta \in \mathsf{Form}$. *In case* $x, U \models_{\mathcal{M}} \alpha$ *is true we say that* α holds in $\mathcal{M}$ at *the neighborhood situation* x, U. *Furthermore, a formula* α *is called* valid in $\mathcal{M}$ *iff it holds in* $\mathcal{M}$ *at every neighborhood situation of* $\mathcal{S}$.

The intuitive notion of knowledge and effort described in the introduction is obviously captured by this definition. Note that any iterated application of K's and $\Box$'s always has an 'only downward' effect in subset spaces.

The following list provides a sound and complete axiomatization of the logic of subset spaces:

1. All instances of propositional tautologies
2. $\mathsf{K}(\alpha \to \beta) \to (\mathsf{K}\alpha \to \mathsf{K}\beta)$
3. $\mathsf{K}\alpha \to (\alpha \wedge \mathsf{K}\mathsf{K}\alpha)$
4. $\mathsf{L}\alpha \to \mathsf{K}\mathsf{L}\alpha$
5. $\Box\,(\alpha \to \beta) \to (\Box\alpha \to \Box\beta)$
6. $(p \to \Box p) \wedge (\Diamond p \to p)$
7. $\Box\alpha \to (\alpha \wedge \Box\Box\alpha)$
8. $\mathsf{K}\Box\alpha \to \Box\mathsf{K}\alpha$,

where $p \in \mathsf{Prop}$ and $\alpha, \beta \in \mathsf{Form}$. In this way, it is expressed that, for every Kripke model M validating these axioms,

- The accessibility relation $\xrightarrow{\mathsf{K}}$ of M belonging to the knowledge operator is an equivalence (i.e., K is an $\mathsf{S5}$-modality),
- The accessibility relation $\xrightarrow{\Box}$ of M belonging to the effort operator is reflexive and transitive (i.e., $\Box$ is $\mathsf{S4}$-like),
- The composite relation $\xrightarrow{\Box} \circ \xrightarrow{\mathsf{K}}$ is contained in $\xrightarrow{\mathsf{K}} \circ \xrightarrow{\Box}$ (this is usually called the *cross property*), and
- The valuation of M is constant along every $\xrightarrow{\Box}$-path, for all proposition variables.

The most important fact is the cross property here, formalizing the interplay of knowledge and effort. The 'stability property' of the proposition variables corresponds to their semantics and is caused by Axiom 6. – Note that the logic of subset spaces is decidable although it does not satisfy the finite model property.

For the rest of this paper, we almost exclusively deal with the smaller class of all topological spaces, i.e., subset spaces where $\mathcal{O}$ is closed under arbitrary unions and finite intersections. These two properties are forced by Axiom 9 and Axiom 10 following right now.

9. $\Diamond\alpha \wedge \mathsf{L}\Diamond\beta \to \Diamond\,(\Diamond\alpha \wedge \mathsf{L}\Diamond\beta \wedge \mathsf{K}\Diamond\mathsf{L}(\alpha \vee \beta))$
10. $\Diamond\Box\alpha \to \Box\Diamond\alpha$,

where $\alpha, \beta \in \mathsf{Form}$. – It is not hard to see that Axiom 9 is valid in spaces closed under arbitrary unions and Axiom 10 in spaces closed under arbitrary intersections.[1] However, the proof of the next theorem is highly non-trivial.

Theorem 1 (Georgatos, [8]). *The logic determined by the just given list of axioms and the common modal proof rules, is sound and complete with respect to the class of all topological spaces.*

Thus, it is justified to call this logic topologic. – In the following, we have to take a closer look at some of the properties of topologic.

3 Topological Modal Logic versus topologic

First in this section, we recall a few basic facts from topological modal logic needed later on. (The reader is referred to the paper [1] for a more detailed exposition.) Afterwards, we study a particular embedding of the set of all formulas of usual modal logic, MF, into the set Form, by which means the topological semantics is related to the semantics of topologic.

Let $T = (X, \tau)$ be a topological space and $V : \mathsf{Prop} \to \mathcal{P}(X)$ a T-valuation. Then, V is extended to all (mono-)modal formulas as follows:

$$\begin{aligned}
\tilde{V}(p) &:= V(p)\\
\tilde{V}(\neg\alpha) &:= X \setminus \tilde{V}(\alpha)\\
\tilde{V}(\alpha \wedge \beta) &:= \tilde{V}(\alpha) \cap \tilde{V}(\beta)\\
\tilde{V}(\Box\alpha) &:= \mathsf{Int}\big(\tilde{V}(\alpha)\big),
\end{aligned}$$

where $p \in \mathsf{Prop}$, $\alpha, \beta \in \mathsf{MF}$, and $\mathsf{Int}(A)$ denotes the *interior* of the set A. This is (essentially) the McKinsey-Tarski interpretation of the $\Box$-operator. The *topological semantics* of a modal formula α *in the model* $M = \big(X, \tau, \tilde{V}\big)$ *at a point* $x \in X$ is now given by $\big(M, x \models \alpha :\Longleftrightarrow\ x \in \tilde{V}(\alpha)\big)$.

It is easy to see that $M, x \models \Box\alpha$ iff $\exists\, U \in \tau : (x \in U$ and $\forall y \in U : M, y \models \alpha)$ is valid. The latter formulation indicates how the embedding announced above should operate on $\Box$-formulas: the modal quantification provided by $\Box$ should be replaced by an existential quantification over open sets followed by a universal quantification over points.

We touched on the following theorem of McKinsey already at the beginning of this paper. It concerns, of course, the topological semantics of modal formulas; cf [11].

Theorem 2 (McKinsey). *The system* S4 *is sound and complete with respect to the class of all topological spaces. Moreover, finite topological spaces even suffice for that.*

[1] Note that the latter schema characterizes the property of *weak directedness* of the accessibility relation in basic modal logic; cf, eg, [9], § 1.

Theorem 2 can actually be strengthened. As is known, a topological space (X, τ) is called *connected* iff its point set X cannot be written as a union of two open proper subsets which are disjoint. In the paper [1], the authors call a topological space *well-connected* iff this non-partitioning property is true even if the two open subsets are not necessarily disjoint. Showing that well-connectedness corresponds to Kripke-structures for S4 which are *generated* (see [9], § 1), they prove the subsequent stronger completeness result.

Theorem 3 ([1], Theorem 3.10). *The system* S4 *is sound and complete with respect to the class of all finite well-connected topological spaces.*

This theorem is used for the proof of Lemma 6 below. – The notion of *bi-persistence of formulas* introduced in the next definition plays a crucial part for topologic, in particular.

Definition 3 (Bi-Persistence). *Let derivability in* topologic *be denoted by* $\vdash$*, and let* $\alpha \in$ Form *be a formula. Then,* α *is called* bi-persistent (for topological spaces) *iff* $\vdash \Diamond\alpha \to \Box\alpha$.

Thus, satisfaction of a bi-persistent formula at a neighborhood situation x, U depends only on the point x.

Let $\Pi \subseteq$ Form be the closure of Prop under negations, conjunctions, and the operators $\Diamond$K and $\Box$L. It is a fact that all elements of Π are bi-persistent for topological spaces. For that, Axiom 10 from Section 2 is responsible, among other things; see [7], Prop. 3.1.

We now define a mapping $\varphi : \mathsf{MF} \to \mathsf{Form}$ in the following way.

$$\begin{aligned} \varphi(p) &= p \\ \varphi(\neg\alpha) &= \neg\varphi(\alpha) \\ \varphi(\alpha \wedge \beta) &= \varphi(\alpha) \wedge \varphi(\beta) \\ \varphi(\Box\alpha) &= \Diamond\mathsf{K}\varphi(\alpha), \end{aligned}$$

for all $p \in$ Prop and $\alpha, \beta \in$ MF.

A bijection from MF onto Π is obviously induced by restricting the mapping φ in the range (thus this bijection is denoted by φ, too). Hence the inverse mapping φ^{-1} from Π to MF exists. φ^{-1} is utilized in the proofs of the results of Section 5.

We close this section by stating the above indicated connection between the McKinsey-Tarski semantics of modal logic and the semantics of topologic introduced in Definition 2.

Proposition 1 ([7], Proposition 3.5). *Let* $T = (X, \tau)$ *be a topological space and* $V :$ Prop $\to \mathcal{P}(X)$ *a* T*-valuation. Let* $\mathcal{M} = (X, \tau, V)$ *and* $M = \left(X, \tau, \tilde{V}\right)$ *be the induced subset space and topological model, respectively. Then, for every* $\alpha \in$ ML *and* $x \in X$*, we have that*

$$M, x \models \alpha \iff x, X \models_{\mathcal{M}} \varphi(\alpha).$$

4 The Normal Form Lemma

This short section is devoted to the most important property of topologic, viz the decomposition of formulas from Form into a 'horizontal' component regarding knowledge and a 'vertical' one regarding effort. The precise formulation of this is a certain *Normal Form Lemma,* which is due to Georgatos [8].[2] In order to be able to state this result, we must first fix some notation.

The set Π introduced right after Definition 3 plays an important part in the following. But we need an additional set of formulas.

Definition 4. *Let a set Σ of formulas from* Form *be defined as the closure of the set $\{\mathsf{K}\pi \mid \pi \in \Pi\} \cup \{\mathsf{L}\pi \mid \pi \in \Pi\}$ under conjunctions.*

Since K is a normal modality, it distributes over conjunctions; see, eg, [5], Example 1.40. This implies that any $\sigma \in \Sigma$ is logically equivalent to a formula of the form $\mathsf{K}\pi \wedge \bigwedge_i \mathsf{L}\pi_i$, where i ranges from 1 to some natural number $j \in \mathbb{N}$. Thus, we may assume without loss of generality that σ in fact has this form.

Definition 5 (Normal Forms). *Let $\alpha \in$* Form *be a formula. Then α is said to be* in prime normal form *iff there are $\pi \in \Pi$ and $\sigma \in \Sigma$ such that $\alpha = \pi \wedge \sigma$. And α is said to be* in normal form *iff α is a disjunction of formulas in prime normal form.*

Let the formula $\alpha \in$ Form be in prime normal form, $\alpha = \pi \wedge \sigma$ (where $\pi \in \Pi$ and $\sigma \in \Sigma$). Then π is the 'vertical component' and σ is the 'horizontal component' of α. For suppose that α holds at any neighborhood situation x, U. Then, due to bi-persistence, π holds at all situations x, V where V, containing x, runs 'up' and 'down' the underlying space. On the other hand, σ moves satisfaction of certain formulas from Π horizontally, i.e., to other points inside U. This 'orientation' of formulas is particularly reasonable in case the open sets of the space one has in mind have a kind of tree structure, like the Cantor space.

We call two formulas $\alpha, \beta \in$ Form *topologically equivalent* iff for all subset spaces $\mathcal{M}$ based on a topological space T and all neighborhood situations x, U of T the assertion $(x, U \models_{\mathcal{M}} \alpha \iff x, U \models_{\mathcal{M}} \beta)$ is valid. Using this way of speaking, we are now in a position to quote Georgatos' Normal Form Lemma.

Lemma 1 (Normal Form Lemma, [8]). *Let $\alpha \in$* Form *be any formula. Then α is topologically equivalent to some formula $\beta \in$* Form *in normal form.*

The proof of Lemma 1 uses, among other things, Axiom 9 from Section 2 in a tricky way.

5 Bisimulations

It is well-known that the right concept for invariance of modal formulas with respect to their usual (i.e., Kripke) models is that of *bisimulation;* cf [5], Ch. 2.2.

[2] Section 3.1 of the paper [7] contains a proof of the Normal Form Lemma as well.

A corresponding notion for topological spaces is given in [1], Definition 2.1. For convenience of the reader, we include this definition here. We also list the key result of Section 4.1 of the paper [1]. Afterwards, we introduce bisimulations also for subset spaces and prove a couple of lemmata connected to this subject. Our final concern is the *sum* of topological spaces. We mention an important special case at the end of this section.

Definition 6 (Topo-Bisimulation). *Let* $M_1 = \left(X_1, \tau_1, \tilde{V}_1\right)$ *as well as* $M_2 = \left(X_2, \tau_2, \tilde{V}_2\right)$ *be topological models, and let* $R \subseteq X_1 \times X_2$ *be a non-empty binary relation. Then* R *is called a* topo-bisimulation *iff the following three conditions are satisfied whenever* $x_1 R x_2$ *is valid for any* $x_1 \in X_1$ *and* $x_2 \in X_2$ *:*

1. *For all* $p \in \mathsf{Prop}$, $\left(x_1 \in \tilde{V}_1(p) \iff x_2 \in \tilde{V}_2(p)\right)$.
2. *If* $x_1 \in U_1$ *for any* $U_1 \in \tau_1$*, then there exists some* $U_2 \in \tau_2$ *such that*
   (a) $x_2 \in U_2$ *and*
   (b) *for all* $y_2 \in U_2$ *there is some* $x_1' \in U_1$ *satisfying* $x_1' R y_2$.
3. *If* $x_2 \in U_2$ *for any* $U_2 \in \tau_1$*, then there exists some* $U_1 \in \tau_1$ *such that*
   (a) $x_1 \in U_1$ *and*
   (b) *for all* $y_1 \in U_1$ *there is some* $x_2' \in U_2$ *satisfying* $y_1 R x_2$.

As usual, items 1, 2 and 3 of Definition 6 (in this order), are called the *base, forth* and *back condition,* respectively.

The following lemma coincides with 'FACT 2.3' from [1], Section 2.3, in terms of content.

Lemma 2 (Topological Invariance of Formulas). *Let* $M_1 = \left(X_1, \tau_1, \tilde{V}_1\right)$ *and* $M_2 = \left(X_2, \tau_2, \tilde{V}_2\right)$ *be as above and* $R \subseteq X_1 \times X_2$ *a topo-bisimulation. Furthermore, assume that* $x_1 \in X_1$ *and* $x_2 \in X_2$ *satisfy* $x_1 R x_2$*. Then, for all formulas* $\alpha \in \mathsf{MF}$*, we have that* $M_1, x_1 \models \alpha$ *iff* $M_2, x_2 \models \alpha$*.*

We want to remind the reader of the notion of morphism of topological spaces before we formulate the key result from [1] we need below. Let $T_1 = (X_1, \tau_1)$ and $T_2 = (X_2, \tau_2)$ be topological spaces and $f : X_1 \to X_2$ a mapping. Then f is called *continuous* iff $f^{-1}[U_2] \in \tau_1$ for every $U_2 \in \tau_2$. Moreover, f is called *open* iff $f[U_1] \in \tau_2$ for every $U_1 \in \tau_1$.

Lemma 4.5 and Corollary 4.6 from [1] are subsumed in the subsequent proposition. We choose a slightly different formulation which is most qualified for our purposes. Note that $\mathcal{C} = (\mathcal{B}, \mathcal{T})$ designates the Cantor space.

Proposition 2. *Let* $M = \left(X, \tau, \tilde{V}\right)$ *be a finite topological model such that the underlying topological space is well-connected. Then there exist a* $\mathcal{C}$*-valuation* V' *and a surjective continuous and open mapping* χ *from* $\mathcal{B}$ *onto* X *which induces a topo-bisimulation of the models* $\left(\mathcal{B}, \mathcal{T}, \tilde{V}'\right)$ *and* M*.*

The proof of Proposition 2 proceeds by suitably unravelling X into $\mathcal{B}$. We omit further details here.

The question whether there is also an idea of bisimulation for *subset spaces* is quite natural. Clearly, this must be a relation between *neighborhood situations*

since these are the basic semantic entities of topologic. And we must respect both modalities, K and $\Box$, in addition. We now present an appropriate definition, comment on it and prove two lemmata, of which one concerns a particular example we get back to later.[3]

Definition 7 (Subset Space Bisimulation). *For $i = 1, 2$, let $\mathcal{S}_i = (X_i, \mathcal{O}_i)$ be subset frames and $\mathcal{M}_i = (X_i, \mathcal{O}_i, V_i)$ subset spaces based on $\mathcal{S}_i$. Moreover, let $R \subseteq \mathcal{N}_{\mathcal{S}_1} \times \mathcal{N}_{\mathcal{S}_2}$ be a non-empty binary relation. Then R is called a* subset space bisimulation *iff the following five conditions are satisfied whenever $(x_1, U_1)\, R\, (x_2, U_2)$ is valid for any $(x_1, U_1) \in \mathcal{N}_{\mathcal{S}_1}$ and $(x_2, U_2) \in \mathcal{N}_{\mathcal{S}_2}$:*

1. *For all $p \in \mathsf{Prop}$, $\big(x_1 \in V_1(p) \iff x_2 \in V_2(p)\big)$.*
2. *For all $x_1' \in U_1$ there exists $x_2' \in U_2$ such that $(x_1', U_1)\, R\, (x_2', U_2)$.*
3. *For all $x_2' \in U_2$ there exists $x_1' \in U_1$ such that $(x_1', U_1)\, R\, (x_2', U_2)$.*
4. *If $x_1 \in U_1'$ for any $U_1' \in \mathcal{O}_1$ such that $U_1' \subseteq U_1$, then there exists $U_2' \in \mathcal{O}_2$ satisfying $x_2 \in U_2' \subseteq U_2$ and $(x_1, U_1')\, R\, (x_2, U_2')$.*
5. *If $x_2 \in U_2'$ for any $U_2' \in \mathcal{O}_2$ such that $U_2' \subseteq U_2$, then there exists $U_1' \in \mathcal{O}_1$ satisfying $x_1 \in U_1' \subseteq U_1$ and $(x_1, U_1')\, R\, (x_2, U_2')$.*

We obviously have two forth conditions as well as two back conditions here, given by items 2, 4 and 3, 5 of Definition 7, respectively. Note that subset space bisimulations apply to topological spaces in particular, however, with respect to a different semantics. – The next lemma is an analogue to Lemma 2.

Lemma 3 (Subset Space Invariance of Formulas). *Let $\mathcal{M}_1 = (X_1, \mathcal{O}_1, V_1)$ and $\mathcal{M}_2 = (X_2, \mathcal{O}_2, V_2)$ be subset spaces based on $\mathcal{S}_1 = (X_1, \mathcal{O}_1)$ and $\mathcal{S}_2 = (X_2, \mathcal{O}_2)$, respectively, and let $R \subseteq \mathcal{N}_{\mathcal{S}_1} \times \mathcal{N}_{\mathcal{S}_2}$ be a subset space bisimulation. Furthermore, assume that $(x_1, U_1) \in \mathcal{N}_{\mathcal{S}_1}$ and $(x_2, U_2) \in \mathcal{N}_{\mathcal{S}_2}$ satisfy $(x_1, U_1)\, R\, (x_2, U_2)$. Then, for all formulas $\alpha \in \mathsf{Form}$, we have that $x_1, U_1 \models_{\mathcal{M}_1} \alpha$ iff $x_2, U_2 \models_{\mathcal{M}_2} \alpha$.*

It turns out that every *embedding* of subset frames induces a subset space bisimulation in a straightforward manner. We give the precise definition first and state this fact right after that.

Definition 8 (Embedding). *Let $\mathcal{S}_1 = (X_1, \mathcal{O}_1)$ and $\mathcal{S}_2 = (X_2, \mathcal{O}_2)$ be subset frames. An injective mapping $\eta : X_1 \to X_2$ is called an* embedding (of $\mathcal{S}_1$ into $\mathcal{S}_2$) *iff $\eta[U_1] \in \mathcal{O}_2$ for every $U_1 \in \mathcal{O}_1$ and $\eta^{-1}[U_2] \in \mathcal{O}_1$ for every $U_2 \in \mathcal{O}_2$. In this case we write $\eta : \mathcal{S}_1 \rightarrowtail \mathcal{S}_2$.*

Lemma 4 (Embedding Lemma). *Let $\mathcal{S}_1 = (X_1, \mathcal{O}_1)$ and $\mathcal{S}_2 = (X_2, \mathcal{O}_2)$ be subset frames and $\eta : \mathcal{S}_1 \rightarrowtail \mathcal{S}_2$ an embedding. Furthermore, let $\mathcal{M}_1 = (X_1, \mathcal{O}_1, V_1)$ be based on $\mathcal{S}_1$. Define an $\mathcal{S}_2$-valuation V_2 through $V_2(p) := \eta[V_1(p)]$, for all $p \in \mathsf{Prop}$, and let $\mathcal{M}_2 = (X_2, \mathcal{O}_2, V_2)$. Finally, let $R \subseteq \mathcal{N}_{\mathcal{S}_1} \times \mathcal{N}_{\mathcal{S}_2}$ be defined by*

$$(x_1, U_1)\, R\, (x_2, U_2) : \iff x_2 = \eta(x_1) \text{ and } U_2 = \eta[U_1],$$

[3] One of the anonymous referees pointed to the fact that Definition 7 and Lemma 3 are contained in Section 3.3 of the paper [4] already.

for all $x_1 \in X_1$, $x_2 \in X_2$, $U_1 \in \mathcal{O}_1$ and $U_2 \in \mathcal{O}_2$. Then R is a subset space bisimulation.

Two more examples of subset space bisimulation occur in the proof of Lemma 8 from the next section.

Finally in this section, we turn to a further concept from topology which plays a part in the proof of our main theorem: that of *topological sum;* cf [6], Ch. I, § 2.4. We recall the special case relevant to our purposes here. Let a finite number $T_1 = (X_1, \tau_1), \ldots, T_n = (X_n, \tau_n)$ of topological spaces be given. Assume that their sets of points, $X_1, \ldots, X_n$, are pairwise disjoint.[4] Then, the topological sum $\amalg_{1 \leq i \leq n} T_i$ of the spaces $T_1, \ldots, T_n$ has $X := X_1 \cup \cdots \cup X_n$ as carrier set, and the set of all $U \subseteq X$ satisfying $U \cap X_i \in \tau_i$ for every $i = 1, \ldots, n$ as set of opens.

Concerning topological sums, we remind the reader of the fact that the sum of n copies of the Cantor space can topologically be embedded into the Cantor space itself. This is also true with regard to subset spaces.

Lemma 5. *Let $\amalg_n \mathcal{C}$ be the topological sum of n copies of the Cantor space. Then there exists an embedding $\eta : \amalg_n \mathcal{C} \rightarrowtail \mathcal{C}$.*

6 Main Result

We now return to topologic. First in this section, we state another three auxiliary results, with putting things from previous sections together already. With that, we obtain our main result, viz the completeness of topologic with respect to the Cantor space.

We call a formula $\alpha \in \mathsf{Form}$ *topologically satisfiable* iff there exist a topological space $T = (X, \tau)$, a subset space $\mathcal{M} = (X, \tau, V)$ based on T, and a neighborhood situation x, U of T such that $x, U \models_{\mathcal{M}} \alpha$. Our first lemma says that every topologically satisfiable formula of the form $\pi \wedge \mathsf{K}\pi'$, where π and π' belong to the set Π introduced in Section 3, is even satisfiable in a finite well-connected topological space.

Lemma 6. *Let $\pi, \pi' \in \Pi$, and let $\alpha = \pi \wedge \mathsf{K}\pi'$ be topologically satisfiable. Then there exist a finite well-connected topological space $T = (X, \tau)$ and a subset space $\mathcal{M} = (X, \tau, V)$ based on T such that, for some neighborhood situation x, U of T, the formula α holds in $\mathcal{M}$ at x, U.*

If we drop the requirement on the special form of the formula α in Lemma 6, then topological sums of finite well-connected topological spaces appear as realizing domains. This is the content of the next lemma.

Lemma 7. *Let the formula $\alpha \in \mathsf{Form}$ be topologically satisfiable. Then there exist a natural number $m \in \mathbb{N}$, finite well-connected topological spaces $T_1 =$*

[4] This may be assumed without loss of generality, since one can take a suitable homeomorphic copy of some of the X_i otherwise.

(X_1, τ_1), ..., $T_m = (X_m, \tau_m)$, and a $\amalg_{1 \leq i \leq m} T_i$-valuation V, such that, for some neighborhood situation x, U of $\amalg_{1 \leq i \leq m} T_i$, the formula α holds in the subset space $\mathcal{M} := (\amalg_{1 \leq i \leq m} T_i, V)$ at x, U.

Note that the *finite model property* (see [5], Section 2.3) of topologic follows from Lemma 7 immediately; see [7], Section 3.5.

In the next step we prove that a topologically satisfiable formula is realized in an m-fold topological sum of the Cantor space, for some $m \in \mathbb{N}$.

Lemma 8. *Let $\alpha \in$ Form be a topologically satisfiable formula. Then there exist some $m \in \mathbb{N}$ and a subset space $\hat{\mathcal{M}}$ based on $\amalg_m \mathcal{C}$ in which α holds at some neighborhood situation.*

We are now up to establish the result we were heading for throughout this paper.

Theorem 4 (Completeness for the Cantor Space). *The system* topologic *is sound and complete with respect to $\mathcal{C}$.*

7 Concluding Remarks

Decades ago, McKinsey and Tarski proved that the modal logic S4 is sound and complete with respect to every dense-in-itself metric separable space, in particular, $\mathcal{C}$ and $\mathbb{R}$. We utilized new proofs of this theorem for strengthening it, at least regarding the Cantor space, by replacing S4 with topologic; the latter system was designed by Moss and Parikh for both epistemic and topological reasoning. To be quite unequivocal here, we proved the soundness and completeness of topologic with respect to $\mathcal{C}$. For this purpose, we considered a notion of bisimulation for subset spaces which turned out to support our proof substantially.

Not only our main result is of theoretical interest, but it also applies to the semantics of programming languages to a certain extent. The thing is that topologic can be identified as the subset space logic of *(algebraic) complete partial orderings (cpos)* endowed with the *Alexandroff* (respectively *Scott*) *topology,* in fact, by utilizing the completeness of this system with respect to $\mathcal{C}$. Thus topologic may be viewed as a *programming logic* in the broader sense. This issue will be taken up in greater detail in the full version of this paper.

Finally, what about $\mathbb{R}$? This is certainly the most obvious question remaining. Well, we know that for every finite well-connected topological space (X, τ) there exists a surjective continuous and open mapping from the open interval $(0, 1) \subseteq \mathbb{R}$ onto X; see [1], 'FACT 4.10' and Lemma 4.13. Furthermore, one can easily see that the topological sum of a finite number of copies of $(0, 1)$ can be embedded into $\mathbb{R}$. Noting that the respective results in case of $\mathcal{C}$ (Proposition 2 and Lemma 5 above) are the key facts we have used for completeness, we are, therefore, able to finish the paper with another nice theorem.

Theorem 5 (Completeness for the Reals). *The system* topologic *is sound and complete with respect to $\mathbb{R}$.*

Acknowledgement

I am grateful to the anonymous referees for their comments which helped to improve this paper.

References

1. Aiello, M., van Benthem, J., Bezhanishvili, G.: Reasoning about space: The modal way. Journal of Logic and Computation 13(6), 889–920 (2003)
2. Aiello, M., Pratt-Hartmann, I.E., van Benthem, J.F.A.K.: Handbook of Spatial Logics. Springer, Heidelberg (2007)
3. Artemov, S., Nogina, E.: Topological semantics of justification logic. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) Computer Science – Theory and Applications. LNCS, vol. 5010, pp. 30–39. Springer, Heidelberg (2008)
4. Başkent, C.: Topics in Subset Space Logic. Institute for Logic, Language and Computation, Universiteit van Amsterdam (July 2007)
5. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. In: Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, Cambridge (2001)
6. Bourbaki, N.: General Topology, Part 1. Hermann, Paris (1966)
7. Dabrowski, A., Moss, L.S., Parikh, R.: Topological reasoning and the logic of knowledge. Annals of Pure and Applied Logic 78, 73–110 (1996)
8. Georgatos, K.: Modal Logics of Topological Spaces. Ph.D. thesis, City University of New York (May 1993)
9. Goldblatt, R.: Logics of Time and Computation, 2nd edn. CSLI Lecture Notes, vol. 7. Center for the Study of Language and Information, Stanford (1992)
10. Kudinov, A.: Topological modal logics with difference modality. In: Governatori, G., Hodkinson, I., Venema, Y. (eds.) Advances in Modal Logic, vol. 6, pp. 319–332. College Publications, London (2006)
11. McKinsey, J.C.C.: A solution to the decision problem for the Lewis systems S2 and S4, with an application to topology. Journal of Symbolic Logic 6(3), 117–141 (1941)
12. McKinsey, J.C.C., Tarski, A.: The algebra of topology. Annals of Mathematics 45, 141–191 (1944)
13. Mints, G.: A completeness proof for propositional S4 in Cantor space. In: Orlowska, E. (ed.) Logic at work. Essays dedicated to the memory of Helena Rasiowa, pp. 79–88. Physica-Verlag, Heidelberg (1999)
14. Moss, L.S., Parikh, R.: Topological reasoning and the logic of knowledge. In: Moses, Y. (ed.) Theoretical Aspects of Reasoning about Knowledge (TARK 1992), pp. 95–105. Morgan Kaufmann, Los Altos (1992)

Algorithmics – Is There Hope for a Unified Theory?*
(Invited Talk)

Juraj Hromkovič

Department of Computer Science, ETH Zurich, Switzerland
juraj.hromkovic@inf.ethz.ch

Abstract. Computer science was born with the formal definition of the notion of an algorithm. This definition provides clear limits of automatization, separating problems into algorithmically solvable problems and algorithmically unsolvable ones. The second big bang of computer science was the development of the concept of computational complexity. People recognized that problems that do not admit efficient algorithms are not solvable in practice. The search for a reasonable, clear and robust definition of the class of practically solvable algorithmic tasks started with the notion of the class $\mathcal{P}$ and of $\mathcal{NP}$-completeness. In spite of the fact that this robust concept is still fundamental for judging the hardness of computational problems, a variety of approaches was developed for solving instances of $\mathcal{NP}$-hard problems in many applications. Our 40-years short attempt to fix the fuzzy border between the practically solvable problems and the practically unsolvable ones partially reminds of the never-ending search for the definition of "life" in biology or for the definitions of matter and energy in physics. Can the search for the formal notion of "practical solvability" also become a never-ending story or is there hope for getting a well-accepted, robust definition of it? Hopefully, it is not surprising that we are not able to answer this question in this invited talk. But to deal with this question is of crucial importance, because only due to enormous effort scientists get a better and better feeling of what the fundamental notions of science like life and energy mean. In the flow of numerous technical results, we must not forget the fact that most of the essential revolutionary contributions to science were done by defining new concepts and notions.

1 A Piece of Computer Science History

The history of the mathematical foundations of computer science started in 1936, when Alan Turing [40] introduced his famous Turing machine as a formal definition of the notion "algorithm" (a mathematical method for symbol manipulation). Church [13], Kleene [25], Post [31], and also others introduced different formalisms in order to provide a formal model of this notion. All these distinct

* This work was partially supported by SNF grant 200021-109252/1.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 181–194, 2010.

models are equivalent to each other with respect to the set of algorithmically solvable problems. The consequence of this experience is the Church-Turing thesis which can be viewed as the first axiom of computer science. Due to the exact formal concept of an algorithm, we have a sharp border between algorithmically solvable problems and algorithmically unsolvable ones and can investigate the limits of the automatization.

The second-most fundamental concept of computer science is the concept of computational complexity [39,18]. Due to this concept, developed in the 1960s, we learned that it is not sufficient to design an algorithm for a computing task, because it may be unrealistic to execute the corresponding amount of computer work. Still worse, there are problems that do not admit any efficient algorithm and hence cannot be solved automatically by the means of computers. A new fundamental question came up: "Which computing problems are "easy" (tractable) and can be efficiently solved and which algorithmically solvable problems are "hard" (intractable), i. e., unsolvable in the practice?" Therefore, people wanted to classify problems with respect to the hardness, but to do this one needs two things:

(1) A closer definition of practical solvability.
(2) A method that can be used to prove that a large amount of computer work is necessary to solve a concrete algorithmic problem (i. e., that the problem is not practically solvable).

As the reader may know, we did not succeed to achieve any of these goals. But we learned a lot by trying to develop a methodology for the classification of problems with respect to their computational difficulty. The first attempt resulted in taking $\mathcal{P}$ (the set of decision problems solvable in polynomial-time) as an approximation of the set of "practically" solvable problems. Why did one accept this "approximation" of practical solvability although it is obvious that an algorithm with $O(n^{100})$ time complexity is everything, but not practical, and that an $O(2^{\sqrt{n}/1000})$ exponential algorithm can be extremely useful in practice? The two main reasons (except being not able to do it better) for connecting polynomial-time computations with the intuitive notion of practical solvability are the following:

(i) The first reason is of theoretical nature. Any definition of a fundamental problem class (or of a basic notion) has to be robust in the sense that it is invariant with respect to all reasonable models of computation. Taking $TIME(n^6)$ as the class of tractable decision problems could cause that a decision problem would be tractable for JAVA programs, but not for multi-tape Turing machines. Since all reasonable computing models are polynomial-time reducible each to each other; the class of problems solvable in polynomial time has the required degree of robustness.
(ii) The second reason is based on our experience, and thus related to practice. In almost all cases, once a polynomial-time algorithm has been found for an algorithmic problem that formerly appeared to be hard, some key insight into the problem has been gained. With this new knowledge, faster

polynomial-time algorithms with a lower degree than the first polynomial algorithm were designed.
There are only a few exceptions like primality testing for which the best-known deterministic polynomial-time algorithm is not practical. On the other hand, we do not know natural computing problems of interest for which the best-known algorithm has a time complexity like $O(2^{\sqrt{n}/1000})$.

To achieve the second goal (2) seems to be a really hard story. No one is able to prove a nonlinear lower bound on the complexity of a $\mathcal{NP}$-complete problem. This is a huge gap between our goal to prove super-polynomial lower bounds on the time complexity of concrete problems and our incompetence to prove at least an $\Omega(n \log n)$ lower bound. Razborov and Rudich [35] showed the critical dimension of this situation: No known method for proving lower bounds on the complexity can work for this purpose. Unable of providing evidence that a problem is hard, Cook [14] and Karp [24] introduced the concept of $\mathcal{NP}$-completeness for classifying problems into easy ones and hard ones. The great idea behind $\mathcal{NP}$-completeness is that, if one $\mathcal{NP}$-hard problem is in $\mathcal{P}$, then all $\mathcal{NP}$-complete problems are efficiently solvable. This is the way to bring the experience about a huge unsuccessful effort of designing efficient algorithms for several thousand $\mathcal{NP}$-complete problems into the theory. Since almost nobody believes that all these many $\mathcal{NP}$-complete problems are easy, the concept of "if one is easy then all are easy" became accepted as a tool for proving the hardness of problems. Moreover, there are also other reasons to believe in the computational hardness of $\mathcal{NP}$-complete problems. If $\mathcal{P} = \mathcal{NP}$ holds, then the verification of proofs of problems in $\mathcal{NP}$ would be as hard as the generation of (as searching for) the proofs and this as well is something we do not want to believe in.

2 Searching for the Definition of the Class of "Practically" Solvable Problems

The end of the story presented above looks like a happy end with some acceptable compromises. But it is not. Probably, it is the beginning of a potentially never-ending story. Why? Right after introducing the concept of $\mathcal{NP}$-hardness, people found ways to solve instances of $\mathcal{NP}$-hard problems in practice. The fact that a problem is $\mathcal{NP}$-hard was not considered as an impossibility of solving it efficiently. The $\mathcal{NP}$-hardness of a problem only became a reason to attack this problem using different tools than those for classical problems from $\mathcal{P}$.

For non-specialists, these statements may be really confusing. How can one solve $\mathcal{NP}$-hard problems without finding polynomial algorithms for them and thus without proving $\mathcal{P} = \mathcal{NP}$?

The true art of designing algorithms for hard problems consists of saving a huge amount of computer work by making as few changes as possible in the specification of the problem or in the requirements of the problem solutions. If it works, one can provide solutions to instances of some hard problems that

are highly appreciated in the corresponding applications. We learned that many $\mathcal{NP}$-hard problems are very sensitive. A negligible change of requirements can help to save an unbelievable amount of work. Even jumps from an intractable amount of computer work to a few seconds on a common computer are possible. In what follows, we divide the basic algorithmic approaches of attacking hard problems into three categories:

1 Weakening of requirements. One can deviate from the requirements to guarantee the computation of the correct result in different ways.

1.1 *Approximation algorithms.* Instead of requiring an optimal solution of an $\mathcal{NP}$-hard optimization problem, one accepts solutions whose quality (cost) does not essentially differ from the cost of the optional solutions. The gain of this approach can lead to an exponential decrease of computational complexity. Note that, in the case of FPTAS (fully polynomial-time approximation scheme), we are even allowed to choose the approximation ration ε and the time complexity is polynomial in the input size n as well as in $1/\varepsilon$. Note that approximation algorithms [17] were already proposed before the concept of $\mathcal{NP}$-completeness was introduced [14] in 1971 and immediately after that, several efficient approximation algorithms for $\mathcal{NP}$-hard optimization problems were designed (see, for instance, [23,36,12,22,27]).

1.2 *Randomization.* The idea of randomized algorithms [33,34,38,1] is to move from the hypothetical absolute guarantee of computing the correct result to computing correct results with high probability. Nobody has doubts that bounded-error randomized algorithms are very practical and usually we consider a problem to be practically solvable if we can solve it by an efficient (polynomial-time) randomized algorithm with bounded-error probability. In spite of the fact that complexity theory provides a good reason for conjecturing $\mathcal{P} = \mathcal{BPP}$, we remain very interested in the design of randomized algorithms, because also a speed-up by a multiplicative factor of n^3 is great. On the other hand, we still know problems, such as the equivalence testing for polynomials or searching for a quadratic non-residue, that can be solved efficiently by randomized algorithms, but where the best-known deterministic algorithms solving these problems have an exponential complexity.

2 Useful Exponential Algorithms. Algorithms with time complexity like 2^n or $n!$ are applicable for very small problem instances only. But algorithms with the time complexity c^n for $c < 2$ could be of practical interest. The classical result of Monien and Speckenmeyer [28,29] provides such an algorithm for kSAT.

Example 1. The starting idea (not providing the result from [28]) for 3SAT is the following one. Instead of looking at each of the 2^n possible assignments, one quickly excludes most of them as candidates that do not need to be checked. If a clause $x_1 \vee x_2 \vee x_3$ has to be satisfied, one needs only to consider the possibilities as depicted in Fig. 1.

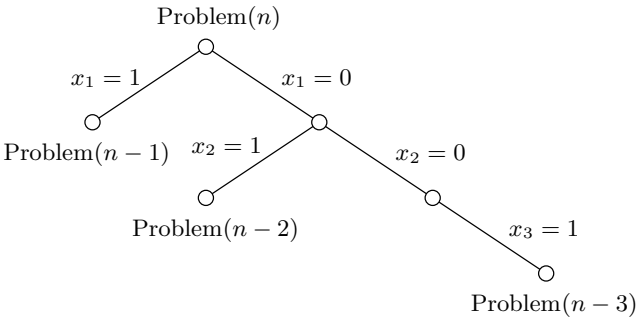


Fig. 1.

If n is the number of variables, this leads to a divide-and-conquer strategy with complexity $T(n)$, given by the recursive equation

$$T(n) = T(n-1) + T(n-2) + T(n-3),$$

which is definitely better than

$$T(n) = 2 \cdot T(n-1)$$

given by the complete search. □

For satisfiability problems, several useful randomized exponential algorithms were developed.

3 Classification of Problem Instances. This is probably the most promising approach to attack hard problems in practice. In fact, one tries to give a deeper analysis of the hard problem considered by trying to figure out the essence of the hardness of the problem. For sure, we cannot speak about the algorithmic hardness of a particular problem instance or a finite set of instances. But one can fix infinite sets of problem instances of a hard problem that can be efficiently solvable. On the other hand, one can prove that some infinite subsets of problem instances contain the instances making the corresponding problem hard. Following these strategies, we know several distinct successful approaches.

3.1 *Pseudopolynomial Algorithms.* For most problems, the input instances can be viewed as sequences of integers. For sure, an integer is exponential in the size of its representation and can thus be also exponential in the size n of the problem instance. Pseudopolynomial algorithms run in time polynomial in n and in the value of the largest integer of the given input instance. If a hard problem admits an efficient pseudopolynomial algorithm, then the subproblem with integers polynomial in the size of their input instances is easy. A consequence is that, for input values restricted to a fixed interval of possible integers, the problem can be efficiently solved.

3.2 *Parameterized Complexity.* The concept of parameterized complexity was introduced by Downey and Fellows [15] and can be viewed as a generalization of the concept of pseudopolynomial algorithms. The complexity function is considered as a function of two variables. One variable is, as usual, the size of the problem instance, and the second argument is the so-called parameter. A parameter has to be an important number characterizing the instances of the given problem that somehow "measures" the hardness of the problem instances. For an input size n and a parameter size k, one considers algorithms with time complexity in $O(f(k) \cdot p(n))$, for a polynomial p and an arbitrary function f, to be useful. In that way, one can fix classes of "easy" problem instances as the classes of problem instances with a small value of k.

3.3 *Stability of Approximation.* This approach is similar to the parameterized complexity in the sense that we again search for a characteristic of the instances of a given hard problem, that captures its hardness. The difference to parameterized complexity is that the parameter does not influence the time complexity, but the quality of the computed solution. One considers this approach for optimization problems that do not admit any polynomial-time algorithm with a constant approximation ratio. The idea is to partition the set of all instances of such an optimization problem into infinitely many classes of infinite sizes. This partition of problem instances has to be given by a parameter k. An approximation algorithm is called stable with respect to a considered parameter k, if its approximation ratio grows with k, but not with n. In this way, the parameter k measures the hardness of the polynomial-time approximability of the given optimization problem.

Example 2. TSP is known to be a hard optimization problem with respect to approximability. One is not even able to guarantee a polynomial approximation ratio within polynomial-time. The proof of this fact is straightforward. One reduces the Hamiltonian Cycle Problem to TSP with edge costs 1 or 2^n where n is the number of vertices. The cost 1 is assigned to all edges of the graph of the instance of the Hamiltonian Cycle Problem, and cost 2^n is assigned to all remaining edges of the complete graph of the constructed instance of TSP.

What does this mean? For sure, these are hard instances of TSP. But are there many hard instances or only those that allow huge differences in the costs of particular edges? We know that for the geometric TSP one has a PTAS and that for the metric TSP one has approximation algorithms with constant approximation ratios. These facts may propose that the metric property is essential for measuring the hardness of the instances of TSP. This hypothesis can be really proved using the concept of stability of approximation. Consider the general β-triangle inequality

$$cost(\{u, v\}) < \beta(cost(\{u, w\}) + cost(\{w, v\})),$$

for arbitrary three vertices u, v, and w of a graph. Obviously, for $\beta = 1/2$, all edges must have the same cost and so such instances of TSP are

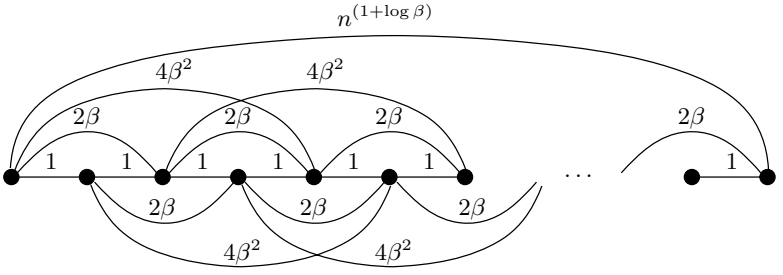


Fig. 2.

trivial. For $\beta = 1$ we get the classical Euclidian metrics. Böckenhauer and Seibert [10] proved that TSP with β-triangle inequality is APX-hard already for any $\beta > 1/2$ and their concrete lower bound on the approximability grows with an increasing β. This is an interesting starting point. In several papers [2,3,5,6,4,20], the authors designed stable efficient algorithms for TSP with growing approximation ratios with growing β. This proves our hypothesis that extremely hard instances of TSP are only few and not natural. The hard instances must break the metric property in such a way that β even becomes dependent on the size of the instance (the number of vertices).

Note that the well-known 2-approximation algorithm for the metric TSP is not stable with respect to the relaxed triangle inequality. In this algorithm, one starts with a minimum spanning tree, and then takes the order of the vertices of a depth-first search as the resulting Hamiltonian cycle. Following this procedure, each edge of the tree is used or "shortened" exactly twice. If the β-triangle inequality for $\beta > 1$ holds, then shortening paths of length m can cause a cost increase that is not only a function of β, but also a function of m (see Fig. 2). Since m can be related to the size of the graph, this algorithm is not stable. Christofides algorithm [12] also is not stable since it shortens paths even twice (once by shorthening paths of an optimal Hamiltonian cycle and later by shorthening the paths of a minimum spanning tree).

A simple idea for designing a stable approximation algorithm for TSP is based on the classical theorem of Sekanina [37] which claims that, for any tree T, the graph T^3 must contain a Hamiltonian cycle. This means that, taking a minimum spanning tree of a complete weighted graph, one can build a Hamiltonian cycle such that

(i) it traverses or shortens each edge of the tree exactly twice and
(ii) the edges of the resulting Hamiltonian cycle are shortcuts of at most three edges of the spanning tree.

In this way, the designed algorithm runs in linear time and guarantees the worst-case approximation ratio $2 \cdot \beta^2$. For more details, we refer to [21], page 302. □

3.4 *Heuristics.* If we speak about heuristics, we mean algorithms whose behavior we are unable to analyze. This means that we are not able to prove that they work efficiently or that they provide reasonable results. In spite of missing a guarantee of getting correct results in time, heuristics, such as simulated annealing or genetic algorithms, may behave very well on "typical" problem instances in some applications. Also, algorithm design methods such as local search, primal-dual method, branch and bound, or greedy may lead to successful heuristics. It is more a rule than an exception that in operations research the most successful implemented algorithms are heuristics and their success is not mathematically proven, but measured in an empirical way. We know examples of $\mathcal{NP}$-hard optimization problems, for which the costs of the solutions provided by heuristics differ by 1.5% in average from the costs of the optimal solutions.

3.5 *Hybrid Algorithms.* Let us consider inputs that can be viewed as problem instances of different problems. A good example is simply a weighted complete graph. A hybrid algorithm for two different problems of that kind efficiently provides a correct solution to one of the two problems. The only trouble is that we are not allowed to force for which one. The algorithm simply tries to solve both problems and then "recognizing" which one is easier for the particular instance, the algorithm finishes its work by computing a solution for the easier task [8,16].

What can one learn from this? If one has an efficient hybrid algorithm for two hard problems, then one discovered that the sets of hard instances of these two problems are disjoint.

3 Discussion about Upper Bounds

In Section 2, we presented some fundamental approaches for attacking hard problems and making some of them "practically solvable" or "practically solvable to some extent". In this way, we may view these approaches as tools for proving upper bounds on the computational difficulty of algorithmic problems. These approaches document very well how hard it is to provide a robust definition of "practical solvability". From the approaches considered in Section 2, only randomization provides a closer picture. Without any doubt, efficient bounded-error randomized algorithms are practical. This is the reason why we correctly consider the class of problems solvable by polynomial-time bounded-error randomized algorithms as a robust class of tractable problems.

But what is the influence of other approaches? Consider approximability. For sure, optimization problems having FPTAS can be viewed to be tractable. Are we allowed to say this also for problems admitting PTAS? Are also efficient approximation algorithms with a small constant approximation ratio as 1.01 a proof of the tractability of the corresponding problem? If yes, which concrete approximation ratio has to provide the exact limit of tractability? Can the stability of approximation have any influence on the notion of tractability? Are all

fixed-parameter-tractable problems really tractable? For instance, what about $(2^{2^h} \cdot n^2)$-parameterized algorithms? Can successful heuristics have any influence on a reasonable definition of a robust class of tractable problems? And on top of it all, one can combine all the approaches listed above. Is there at all a chance to find a clear and well-accepted border between tractable problems and the really "practically unsolvable" problems?

It looks like a visit by a medician. For each problem one tries to apply different approaches or their combination in order to solve it at least to some achievable extent. This corresponds to the current situation. We have a huge amount of results about attacking several thousand problems and these results very rarely help us to fix the practical solvability of the corresponding problems or to compare the computational hardness of two $\mathcal{NP}$-hard problems.

Is there a way out? Are we forced to sample a huge amount of often incomparable algorithms for different problems, and so to build a continuously larger and larger zoological garden forever? Are we forced to build highly specialized "doctors" for small specific classes of computing problems who do not need to take care of the general context of algorithmics, because a unified theory providing a useful insight is missing?

To find a way out, we have to try to fix the reason why everything becomes so fuzzy if one wants to classify algorithmic problems with respect to the tractability. I believe that at least one of the reasons (if not the only one) of our troubles lies in our definition of the complexity in the worst case. We needed this definition in order to be able to analyze the complexity of algorithms within a reasonable effort and in order to compare the complexity of different algorithms. Taking Time(n) as the maximum of the complexity of all computations on inputs of size n instead of considering the average complexity with respect to some inputs probability distribution is not wrong. In algorithms as a theory, we are always forced to provide guarantees of computing correct results in a reasonable time limit. I do not like the idea of giving up this effort. Proving guarantees in the above mentioned sense is our way to a deeper understanding of algorithm design and computational complexity, and so it is an essential contribution to the computer science fundamentals. On the other hand, if proportionally few problem instances are allowed to make a problem hard and these instances are even somehow exotic, then one is allowed to prefer to view this problem (if not as easy) as a not very hard one. The way out is really not taking a weighted average, because this is not robust. Different probability distributions of inputs may lead to different results and different applications may force to consider different probability distributions. Therefore, taking the weighted average, one can risk to lose robustness.

One way out we propose is related to the concepts of parameterized complexity and stability of approximation. One has to try to understand and express the hardness of the problem by partitioning the set of all instances of a considered problem into infinitely many classes of infinite cardinality. In each of these infinitely many classes, one has to use the worst-case measurement for the computational complexity or the solution quality. If this effort is successful, then one

splits all instances of the problem into infinitely many classes of instances with growing hardness from class to class. It seems that this is a reasonable way of classifying the hardness of input instances. The fact that an instance is in a class provides us an upper bound on the hardness of this instance. If one succeeds to embed her or his intuition into such a classification, then one does a big step in understanding the computational hardness of the problem.

This can be even helpful for the classification of the problems with respect to their hardness. I do not see how to use this concept in order to get a clear border between tractable problems and intractable ones. But being able to parameterize algorithmic problems in the way described above can lead to a reasonable measurement of the problem hardness. Even if it results in some hierarchy of several degrees of hardness, it can be helpful for judging the tractability of the problems in concrete applications and instrumental in searching for a way to master them.

Note that this approach does not suggest to follow the most typical approach of parameterized complexity, where a part of the input is considered as a parameter (for instance, k is a parameter of the input instance (G, k) of the vertex cover problem asking for the existence of a vertex cover of size k in G). One is forced to discover the proper reasons of the hardness of the problem investigated by searching for some crucial property of the problem instances that takes an essential influence on the problem hardness. I personally believe that this kind of research is very promising.

4 Proving Lower Bounds on Complexity

Proving lower bounds on the computational complexity of problems seems to be the hardest core of current research in computer science fundamentals. As already mentioned above, mathematics looks to be underdeveloped for this purpose. Proving the non-existence of an oject is usually a harder task than to prove its existence, especially if one can do it in a constructive way. But proving the non-existence of efficient algorithms for specific algorithmic problems is a real challenge and we do not have any promising idea how to develop instruments which are able to successfully attack this problem. Let us shortly summarize our effort to get at least some progress in developing mathematical tools for this purpose and/or to provide a useful methodology to algorithm designers.

1 **Proving lower bounds on the complexity of concrete problems.** Since we are not able to prove super-linear lower bounds on the time complexity of concrete problems in $\mathcal{NP}$ (or super-logarithmic lower bounds on the space complexity of problems in $\mathcal{P} \supseteq \mathcal{NLOG}$) one tries at least to prove lower bounds on the complexity measures of restricted models of computation. On one side, this contributes to the development of a mathematical machinery for proving lower bounds (see, for instance, the concept of communication complexity introduced by Yao [41,19,26]). A byproduct of this effort is a lot of new results about fundamental computing models like automata

or different kinds of circuits. On the other hand, this effort is helpful to separate determinism, nondeterminism, and different modes of randomization and so it essentially contributed to our understanding of computation and its fundamental phenomena. We do not try to present any survey on proving lower bounds on the complexity of concrete problems in restricted scenarios, because one would need a book series for this purpose.

2 Proving that some design technique does not work. Another way to restrict the class of all algorithms is to consider algorithm design techniques such as greedy, local search, dynamic programming, etc. The idea is to prove at least that some approaches cannot help to efficiently solve the considered problems. Here, we can distinguish two different research streams. One is based on absolute results and another one is based on assumptions such as $\mathcal{NP} \neq \mathcal{P}$ or similar ones.

2.1 *Absolute results.* The results that some design techniques cannot be successful in efficiently solving concrete problems are of methodological importance. In this way, we can learn more about the reasons of the hardness of a problem, as well as about the advantages and drawbacks of specific methods for algorithm design. For instance, a nice classical result of Papadimitriou and Steiglitz shows that local search is not suitable for solving TSP [32]. Even neighbourhoods of exponential size do not suffice to reach a polynomial approximation ratio with reasonable probability.

Another approach to understand greedy algorithms and their generalizations in combinatorial optimization was recently proposed by Borodin (see [11] for a survey). In spite of the fact that this kind of effort does not solve our problem with general lower bounds, to work on corresponding goals is very promising for our understanding of algorithms and their limits. Contributions showing boundaries for some design techniques may be instrumental for algorithms design.

2.2 *Assuming* $\mathcal{NP} \neq \mathcal{P}$. There are many lower bounds based on the well accepted assumption that $\mathcal{NP} \neq \mathcal{P}$ or on similar assumptions. For sure, we understand that $\mathcal{NP}$-hardness of a problem is a confirmation of the fact that the problem is computationally hard. But our goal is to approach the limits of practical solvability, and we gave arguments for considering several $\mathcal{NP}$-hard problems to be tractable. Therefore, we aim to speak about higher degrees of hardness than $\mathcal{NP}$-hardness in this section.

Let us list some of them. The strong $\mathcal{NP}$-hardness of some problems shows that there are no pseudopolynomial algorithms for them. This concept is interesting for us, because strong $\mathcal{NP}$-hardness is based on the fact that already "easy" instance classes with small integer values are $\mathcal{NP}$-hard. From the viewpoint of parameterization, this proves that using the maximum integer value as a parameter is not suitable for classifying the hardness of instances of these problems. In this sense, one can propose other versions of "strong" $\mathcal{NP}$-hardness to show that other parameterizations do not work as well.

In the area of approximability, we have several advanced techniques and concepts like $\mathcal{APX}$-hardness for proving that some discrete optimization problems do not admit efficient algorithms with a "good" approximation ratio (for a survey, see, for instance, [21]).

Another way to show a higher degree of hardness for concrete problems is the concept of reoptimization (see [7] for a survey and [9] for some specific lower bounds). The idea is to provide an optimal solution to a "similar" instance for free, in addition to the input of an optimization problem. If this does not make the problem easier (for example, if the problem remains $\mathcal{APX}$-hard), one may consider the corresponding problem to be hard. Since the reoptimization versions of some optimization problems are easier than the original problem and some others are not, we get another new view (criterion) of problem hardness.

For the parameterized complexity [15, 30], we also have a special notion of hardness that is stronger than $\mathcal{NP}$-hardness. Due to it, one can prove that there is no parameterized algorithm for some considered problems.

5 What to Do with Lower Bounds on Computational Hardness?

I think we are asked to continue in all directions mentioned above, because all are promising in the sense that we learn a lot during our effort to understand why some algorithmic approaches do not work. Additionally, allow me to encourage people to think about lower bounds with respect to the classification of problem instances. In contrast to our ability to prove upper bounds on the hardness of particular problem instances, one cannot prove a lower bound on a particular problem instance, because something like that does not exist. But a good example about what is achievable is given by Böckenhauer and Seibert [10]. They proved explicit lower bounds on the approximability of TSP with β-triangle inequality, and these lower bounds increase with growing β.

Concluding the discussion, I would like to express my belief that, in order to make essential progress in algorithmics, one has to move from measuring the problem hardness in a worst-case manner to classifying the hardness of the instances of the investigated algorithmic problems.

References

1. Adleman, L.M., Manders, K.L., Miller, G.L.: On taking roots in finite fields. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977), pp. 175–178 (1977)
2. Andreae, T.: On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality. Networks 38(2), 59–67 (2001)
3. Andreae, T., Bandelt, H.-J.: Performance guarantees for approximation algorithms depending on parameterized triangle inequalities. SIAM Journal on Discrete Mathematics 8, 1–16 (1995)

4. Bender, M., Chekuri, C.: Performance guarantees for TSP with a parametrized triangle inequality. Information Processing Letters 73, 17–21 (2000)
5. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem. In: Bongiovanni, G., Petreschi, R., Gambosi, G. (eds.) CIAC 2000. LNCS, vol. 1767, pp. 72–86. Springer, Heidelberg (2000)
6. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem. Theoretical Computer Science 285(1), 3–24 (2002)
7. Böckenhauer, H.-J., Hromkovič, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 50–65. Springer, Heidelberg (2008)
8. Böckenhauer, H.-J., Klasing, R., Mömke, T., Steinová, M.: Improved approximations for TSP with simple precedence constraints. In: Proceedings of the 7th International Conference on Algorithms and Complexity, CIAC 2010 (to appear, 2010)
9. Böckenhauer, H.-J., Komm, D.: Reoptimization of the metric deadline TSP. Journal of Discrete Algorithms 8, 87–100 (2010)
10. Böckenhauer, H.-J., Seibert, S.: Improved lower bounds on the approximability of the traveling salesman problem. RAIRO Theoretical Informatics and Applications 34, 213–255 (2000)
11. Borodin, A.: The Power and Limitations of Simple Algorithms: A Partial Case Study of Greedy Mechanism Design for Combinatorial Auctions. In: Dolev, S. (ed.) ALGOSENSORS 2009. LNCS, vol. 5804, p. 2. Springer, Heidelberg (2009)
12. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University (1976)
13. Church, A.: An undecidable problem in elementary number theory. American Journal of Mathematics 58, 345–363 (1936)
14. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971), pp. 151–158. ACM, New York (1971)
15. Downey, R.G., Fellows, M.R.: Parameterized Complexity. In: Monographs in Computer Science. Springer, New York (1999)
16. Eppstein, D.: Paired approximation problems and incompatible inapproximabilities. In: Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 1076–1086 (2010)
17. Graham, R.: Bounds for certain multiprocessor anomalies. Bell Systems Technical Journal 45, 1563–1581 (1966)
18. Hartmanis, J., Stearns, R.: On the computational complexity of algorithms. Transactions of the American Mathematical Society 177, 285–306 (1965)
19. Hromkovič, J.: Communication Complexity and Parallel Computing. Springer, Heidelberg (1997)
20. Hromkovič, J.: Stability of approximation algorithms for hard optimization problems. In: Bartosek, M., Tel, G., Pavelka, J. (eds.) SOFSEM 1999. LNCS, vol. 1725, pp. 29–47. Springer, Heidelberg (1999)
21. Hromkovič, J.: Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. In: Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin (2003)
22. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM 22(4), 463–468 (1975)

23. Johnson, D.S.: Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences 9, 256–278 (1974)
24. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Plenum, New York (1972)
25. Kleene, S.: General recursive functions of natural numbers. Mathematische Annalen 112, 727–742 (1936)
26. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1997)
27. Lovász, L.: On the ratio of the optimal integral and fractional covers. Discrete Mathematics 13, 383–390 (1975)
28. Monien, B., Speckenmeyer, E.: 3-satisfiability is testable in $O(1.62^r)$ steps. Bericht Nr. 3/1979, Reihe Theoretische Informatik, Universität-Gesamthochschule Paderborn (1979)
29. Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2^n. Discrete Applied Mathematics 10(3), 287–295 (1985)
30. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
31. Post, E.: Finite combinatory process–formulation. Journal of Symbolic Logic 1, 103–105 (1936)
32. Papadimitriou, C.H., Steiglitz, K.: On the complexity of local search for the traveling salesman problem. SIAM Journal of Computing 6(1), 76–83 (1977)
33. Rabin, M.O.: Probabilistic algorithms. In: Traub, J.F. (ed.) Algorithms and Complexity: Recent Results and New Directions, pp. 21–39. Academic Press, London (1976)
34. Rabin, M.O.: Probabilistic algorithms for primality testing. Journal of Number Theory 12, 128–138 (1980)
35. Razborov, A., Rudich, S.: Natural proofs. Journal of Computers and System Sciences 55(1), 24–35 (1997)
36. Sahni, S., Gonzalez, T.F.: $\mathcal{P}$-complete approximation problems. Journal of the ACM 23(3), 555–565 (1976)
37. Sekanina, M.: On an ordering of the vertices of a graph. Publications of the Faculty of Sciences University of Brno 412, 137–142 (1960)
38. Solovay, R., Strassen, V.: A fast monte-carlo test for primality. SIAM Journal of Computing 6(1), 84–85 (1977)
39. Stearns, R.E., Hartmanis, J., Lewis, P.M.: Hierarchies of memory limited computations. In: Proceedings of IEEE Sixth Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965), pp. 179–190. IEEE, Los Alamitos (1965)
40. Turing, A.: On computable numbers with an application to the Entscheidungsproblem. In: Proceedings of the London Mathematical Society, vol. 42, pp. 230–265 (1936)
41. Yao, A.C.: The entropic limitations on VLSI computations (extended abstract). In: Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC 1981), pp. 308–311 (1981)

Classifying Rankwidth k-DH-Graphs

Ling-Ju Hung and Ton Kloks*

Department of Computer Science and Information Engineering
National Chung Cheng University, Chia-Yi 621, Taiwan
hunglc@cs.ccu.edu.tw

Abstract. Let k be a natural number. A graph is k-distance hereditary if it has a tree-decomposition such that every cutmatrix has a block structure that is some submatrix of $\left(\begin{smallmatrix} I_k & 0 \\ 0 & 0 \end{smallmatrix}\right)$, where I_k is the $k \times k$ identity matrix. We characterize k-distance hereditary graphs and we show that for fixed k there exists an $O(n^3)$ time algorithm that recognizes the graphs in this class.

1 Introduction

Recent results on tree-decompositions of graphs, such as rank– and cliquewidth-decompositions, make it a point of interest to investigate graphs that can be decomposed such that every cutmatrix has a certain shape. To explain our focus we need a few definitions.

A graph G is a pair $G = (V, E)$ where V is a finite set, of which the elements are called the vertices of G, and where E is a set of two-element subsets of V, of which the elements are called the edges of G.

Definition 1. *A* tree-decomposition *of a graph* G *is a pair* (T, f) *where* T *is a ternary tree and where* f *is a bijection from the leaves of* T *to the vertices of* G.

Definition 2. *Let* (T, f) *be a tree-decomposition of a graph* $G = (V, E)$. *Let* e *be a line in* T *and consider the two sets* A *and* B *of leaves of the two subtrees of* $T - e$. *The* cutmatrix M_e *is the submatrix of the adjacency matrix of* G *with rows indexed by the vertices of* A *and columns indexed by the vertices of* B.

Oum studies rank-decompositions in [12]. A graph has rankwidth k if it has a tree-decomposition such that every cutmatrix has binary rank at most k. The interest in graphs with small rankwidth stems from the fact that problems that can be formulated in monadic second-order logic, can be solved in $O(n^3)$ time on graphs with rankwidth at most k. For each k there exists an $O(n^3)$ algorithm which produces a tree-decomposition of rankwidth at most k for an input graph G with n vertices, if that exists. This is proven in two ways. Firstly, it can be seen that the class of graphs with rankwidth at most k is closed under taking induced subgraphs and under complementing subgraphs that are induced by neighborhoods of vertices. This last operation is usually referred to as a local complementation. The graphs that are obtained from a graph G

* This author is supported by the National Science Council of Taiwan, under grants NSC 98–2218–E–194–004 and NSC 98–2811–E–194–006.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 195–203, 2010.

by a series of these operations are called the vertex-minors of G. It is shown that graphs with bounded rankwidth can be characterized by a finite collection of forbidden vertex-minors. A test for a vertex-minor can be formulated in monadic second-order logic, and this proves the claim. Secondly, also an explicit recognition algorithm is reported.

In order to classify graphs with rankwidth k we study graphs that have a tree-decomposition such that every cutmatrix has a certain shape.

Definition 3. *Consider a* $0,1$*-matrix* M. *Let* M' *be the maximal submatrix of* M *with no two rows equal and no two columns equal. The* shape of M *is the class of matrices equivalent to* M' *under permuting rows, permuting columns, and taking the transpose.*

Let's look at a few classic examples of graph classes that are characterized by shapes of cutmatrices. A graph is a cograph if it has no induced P_4, that is, a path with four vertices. We can characterize cographs with the aid of 'twins.' A *module* is a set M of vertices such that

$$x, y \in M \quad \Rightarrow \quad N(x) - M = N(y) - M.$$

A *twin* is a module with two vertices. The twin is *false* if the vertices are not adjacent and it is *true* if the vertices are adjacent. Cographs are the graphs for which every nontrivial induced subgraph has a twin. Corneil *et al.* show that a graph is a cograph if and only if it has a tree-decomposition such that every cutmatrix has a shape which is equivalent to some submatrix of $\begin{pmatrix} 1 & 0 \end{pmatrix}$.

Howorka defines distance-hereditary graphs as follows:

Definition 4. *A graph* G *is* distance hereditary *if for all nonadjacent vertices* x *and* y *in some component of* G*, all induced* x, y*-paths have the same length.*

Chang *et al.* give the following characterization.

Theorem 1. *A graph* G *is distance hereditary if and only if every induced subgraph has a twin, or a pendant vertex, or an isolated vertex.*

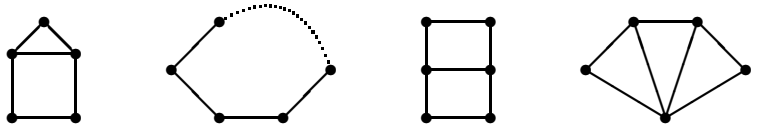


Fig. 1. A graph is distance hereditary if it has no induced house, hole, domino or gem

Distance-hereditary graphs are the graphs with rankwidth one. That is, they are those graphs that have a tree-decomposition (T, f) such that every cutmatrix has a shape equivalent to some submatrix of $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$, where the 1 stands for an all-ones block and the zeros stand for all-zero blocks. In our terminology, distance-hereditary graphs are 1-distance hereditary. Alternatively, this class is characterized as the class of graphs that do not have C_5 as a vertex-minor. Note that the characterization by vertex-minors is finite, whereas the characterization by forbidden induced subgraphs is infinite. However, note also that for $k > 1$, k-distance hereditary graphs are *not* closed under local

complementations. For example, the 5-cycle is 2-distance hereditary and it is locally equivalent to the gem, but the gem is not 2-distance hereditary. At the moment we are not aware of a finite characterization of k-DH-graphs in terms of 'forbidden structures.'

As a third example we mention graphs that have a tree-decomposition such that every cutmatrix is shape-equivalent to some submatrix of $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ [2,8]. They are the graphs without C_5, bull, gem, or co-gem; *i.e.*, the Seidel-switching class has no C_5. Alternatively, they can be characterized as those graphs for which every nontrivial, induced subgraph has a twin or an antitwin. Also, if G_x is the graph in the switching class of G for which the vertex x is isolated, then G is in the class if and only if every, or also if some, G_x is a cograph.

In this paper we investigate k-distance-hereditary graphs for fixed, natural numbers k.

Definition 5. *Let* k *be a natural number. A graph* G *is* k*-distance hereditary if it has a tree-decomposition in which every cutmatrix is shape-equivalent to some submatrix of*

$$\begin{pmatrix} I_k & 0 \\ 0 & 0 \end{pmatrix},$$

where I_k *is the* $k \times k$ *identity matrix and where the zeros stand for all-zero blocks.*

If a graph G has only one vertex then a tree-decomposition for G has no cutmatrix. As a rule, we say that a 1-vertex graph is also k-distance hereditary. We prove that there is a characterization for k-DH-graphs which can be formulated in monadic second-order logic. Since k-DH-graphs have rankwidth at most k, this proves that the graphs can be recognized in $O(n^3)$ time. We end this section with some of our notational customs [7] and with one elementary observation.

For two sets A and B we write $A + B$ and $A - B$ instead of $A \cup B$ and $A \setminus B$. We write $A \subseteq B$ if A is a subset of B with possible equality and we write $A \subset B$ if A is a subset of B and $A \neq B$. For a set A and an element x we write $A + x$ instead of $A + \{x\}$ and we write $A - x$ instead of $A - \{x\}$. It will be clear from the context when x is an element instead of a set.

A graph G is a pair $G = (V, E)$ where V is a *finite*, nonempty set, of which the elements are called the vertices of G, and where E is a set of two-element subsets of V, of which the elements are called the edges of G. If the graph is a tree, we call the vertices points, and the edges lines. A graph consisting of a single vertex is called *trivial*. We denote edges of a graph as (x, y) and we call x and y the endvertices of the edge. For a vertex x we write $N(x)$ for its set of neighbors and we write $N[x] = N(x) + x$ for the closed neighborhood of x. For a subset $W \subseteq V$ we write $N(W) = \bigcup_{x \in W} N(x) - W$ for its neighborhood and we write $N[W] = N(W) + W$ for its closed neighborhood. Usually we use $n = |V|$ to denote the number of vertices of G and we use $m = |E|$ to denote the number of edges of G.

For a graph $G = (V, E)$ and a subset $S \subseteq V$ of vertices we write $G[S]$ for the subgraph *induced* by S, that is, the graph with S as its set of vertices and with those edges of E that have both endvertices in S. For a subset $W \subseteq V$ we write $G - W$ for the graph $G[V - W]$. For a vertex x we write $G - x$ rather than $G - \{x\}$.

Lemma 1. *'Creating a twin' of a vertex* x *in a graph* G *is the operation of adding a new vertex* x' *and adding edges incident with* x' *such that* x' *and* x *become twins. Assume that* G *is* k*-distance hereditary for some* $k \geqslant 1$ *and let* G' *be obtained from* G *by creating a twin. Then* G' *is also* k*-distance hereditary.*

Proof. Let (T, f) be a tree-decomposition for G. Construct a tree-decomposition (T', f') for G' by subdividing the line in T that has the image of x as an endpoint. Create a new leaf for x' and make this adjacent to the subdivision point.

First consider the line in T' incident with x'. If x' is isolated, then the cutmatrix is the all-zero matrix. Thus the shape is indeed a submatrix of $\left(\begin{smallmatrix} I_k & 0 \\ 0 & 0 \end{smallmatrix}\right)$. If x' is not isolated, then the shape is equivalent to $\begin{pmatrix} 1 & 0 \end{pmatrix}$ or to $\begin{pmatrix} 1 \end{pmatrix}$, which is also OK because $k \geqslant 1$. Obviously, the same holds true for the cutmatrix of the line incident with x in T'. Finally, for any other line in T', the row (or column) of the cutmatrix that corresponds with x' is a copy of the row (or column) that corresponds with x. Thus the shape is the same as that of the corresponding original line in T. □

In the same manner it follows without much difficulty that the class of k-DH-graphs is hereditary, and that it is also closed under creating pendant vertices and under creating isolated vertices.

2 k-Cographs

The class of k-cographs is defined as follows.

Definition 6. *Let* k *be a natural number. A graph* $G = (V, E)$ *is a* k-cograph *if there exists a coloring of the vertices with colors from* $\{1, \ldots, k\}$ *such that for every subset of vertices* $W \subseteq V$ *with* $|W| \geqslant 2$ *there exist a partition* $\{W_1, W_2\}$ *and a permutation* $\sigma \in \mathrm{Sym}(k)$*, such that for each* $i \in \{1, \ldots, k\}$*, either*

i. *vertices of* W_1 *of color* i *have no neighbor in* W_2*, or*
ii. *vertices of* W_1 *of color* i *are adjacent exactly to vertices of* W_2 *of color* $\sigma(i)$.

We say that the graph is partitioned if it is equipped with a k-coloring. We call a partition as stipulated in this definition, a k-modular partition.

In a recent paper we define a slightly different notion of k-cographs [9]. The following proof, which shows that the class of k-cographs is characterized by a finite set of forbidden induced subgraphs, is essentially the same as the proof for the related class of graphs in that paper. We include it for completeness sake.

Theorem 2. *Let* k *be a natural number. Partitioned* k*-probe cographs are well-quasi-ordered by the induced subgraph relation.*

Proof. A cotree is a binary tree with a bijection from the leaves to the vertices of the graph and internal nodes labeled as join- or union-operators [4]. Two vertices are adjacent in the graph if and only if their lowest common ancestor is a join-node. Kruskal's theorem [10] states that trees, with points labeled by a well-quasi-ordering, are well-quasi-ordered with respect to their lowest common ancestor embedding. Pouzet observed that this implies that cographs are well-quasi-ordered by the induced subgraph relation [13]. For partitioned k-cographs we equip each leaf with a label that is a color from $\{1, \ldots, k\}$. Each internal node receives a label which consists of

a. a permutation $\sigma \in \mathrm{Sym}(k)$, and
b. for each $i = 1, \ldots, k$ an indicator of a union– or a join-operator.

Two vertices are adjacent if their lowest common ancestor has a label that makes them adjacent by a join-operator. Kruskal's theorem implies the claim. □

Remark 1. Note that each color class of a partitioned k-cograph induces a cograph.

Corollary 1 ([9]). *Let* k *be a natural number. The class of partitioned* k*-cographs is characterized by a finite set of forbidden induced colored graphs. Also, the class of* k*-cographs is characterized by a finite set of forbidden induced subgraphs.*

Theorem 3. *For every natural number* k *there exists an* $O(n^3)$ *algorithm that recognizes* k*-cographs.*

Proof. This can be seen in two ways. First notice that k-cographs have a tree-decomposition such that every cutmatrix is shape-equivalent to a submatrix of $\left(\begin{smallmatrix} I_k & 0 \end{smallmatrix}\right)$, where I_k is the $k \times k$ identity matrix [9]. *A fortiori*, k-cographs have bounded rankwidth. Since the definition of k-cographs can be formulated in monadic second-order logic, this proves the claim. Alternatively one could verify that the graph has none of the forbidden induced subgraphs. Note however that this last proof is nonconstructive; Kruskal's theorem does not provide the forbidden induced subgraphs. □

3 A Tree-Structure

In this section we describe the structure of k-distance-hereditary graphs as a tree-structure of k-cographs. Chang *et al.* describe connected distance-hereditary graphs as binary tree-decompositions where each branch is equipped with a 'twinset.' These twinsets are cographs. A twinset of a branch is obtained from the twinsets of the two children by either a union – or a join operation, or by a 'pendant operation.' This last operation copies exactly one of the two twinsets. For k-DH-graphs, to create a k-twinset of a branch, we have these three choices for each of the k σ-paired color classes of the two k-twinsets of the children, where σ is some permutation. We prove that k-twinsets are k-cographs.

Let $G = (V, E)$ be k-distance hereditary and let (T, f) be a tree-decomposition for G such that every cutmatrix is shape-equivalent to some submatrix of $\left(\begin{smallmatrix} I_k & 0 \\ 0 & 0 \end{smallmatrix}\right)$. Let c be an internal point of T and let P, Q, and R be the partition of V induced by the leaves of the three branches at c. Let $P' = N(Q + R)$, and define Q', and R' likewise.

Lemma 2. *There exists a coloring of each of* P'*,* Q'*, and* R' *with at most* k *colors, such that for each pair of* P'*,* Q'*, and* R'*, say* P' *and* Q'*, there is a permutation of the colors* $\sigma \in \mathrm{Sym}(k)$ *such that vertices of color* i *in* P' *either have no neighbors in* Q'*, or else they are adjacent exactly to the vertices with color* $\sigma(i)$ *of* Q'*.*

Proof. Consider the branch at c that carries P and let e be the line of that branch incident with c. By assumption, the cutmatrix of e is shape-equivalent with some submatrix $\left(\begin{smallmatrix} I_k & 0 \\ 0 & 0 \end{smallmatrix}\right)$. Consider the cutmatrix with rows indexed by the vertices of P and columns by

the vertices of $Q + R$. The nonzero rows of this submatrix correspond with the vertices of P'. Thus there exists a coloring of the vertices of P' with at most k colors such that vertices with the same color have exactly the same neighbors in $Q + R$ and vertices with different colors have no common neighbor in $Q + R$. Of course, similar colorings exist for the vertices of Q' and for the vertices of R'. This proves the claim. □

Consider an arbitrary line α of T and root the tree at α. We obtain a binary tree-decomposition by distinguishing left – and right subtrees. Consider a branch, rooted at some line $e = (c, c')$ such that c' is the child of c. Let C be the set of vertices that are mapped to leaves of the subtree.

Definition 7. *The* twinset at e *is the set of vertices of* C *that have neighbors in* $V - C$. *In other words; the twinset at* e *is* $N(V - C)$.

Lemma 3. *The twinset of* e *induces a* k*-cograph.*

Proof. Consider the cutmatrix at e. Let $R = V - C$ and let $R' = N(C)$. Then there exists a partition of R' into at most k color classes, such that vertices with the same color have the same neighbors in C and such that vertices with different colors have no common neighbor in C. If C consists of a single vertex, there is nothing to prove. Otherwise, the tree decomposes C into two sets A and B. Let A' and B' be the vertices of A and B that have neighbors in R. The vertices of A' are colored with at most k colors, such that vertices with the same color have the same neighbors in $V - A$ and such that vertices with different colors have no common neighbor in $V - A$. The same holds for B, and we may permute the colors of B such that vertices of color i in $A' + B'$ have the same neighbors in R. By induction we may assume that the colored graphs A' and B' induce k-cographs. There exists a permutation $\sigma \in \mathrm{Sym}(k)$ that matches vertices of color i in A' with vertices of color $\sigma(i)$ in B' and this proves that $A' + B'$ induces a k-cograph. □

Connecting the twinsets. To complete the description of the k-DH-structure we analyze the connections between color classes of three branches meeting in a point.

Let P, Q, and R be a partition of V induced by the three branches of an internal node c of a tree-decomposition. Let $P' = N(Q + R)$ and define Q' and R' likewise. Consider the graph $\Omega = \Omega(P, Q, R)$ which has as its vertexset $V(\Omega)$ the set of color classes of the twinsets P', Q', and R'. We connect two color classes in different twinsets by an edge in $E(\Omega)$ if their color classes are joined in G. Notice that every vertex of Ω has degree at most two; for example, a vertex that represents a color class of P' has degree two if it is joined to exactly one color class of Q' and it is joined to exactly one color class of R'. Thus Ω is the union of a collection of paths and cycles that pass alternately through P', Q' and R'. Obviously, the total number of vertices in these paths and cycles is at most $3k$.

We end this section with some observations concerning the structure of k-DH-graphs in terms of forbidden induced subgraphs.

The class of 2-DH-graphs contains all cycles. Also the domino and the house are 2-distance hereditary. Forbidden induced subgraphs for 2-DH-graphs include the 3×3-grid and some infinite collection of 'gem-structures.' A gem-structure consists of a path

$[p_1, \ldots, p_t]$ with at least 4 vertices and one additional vertex c. The vertex c is adjacent to p_1, p_2, p_{t-1} and p_t. Furthermore, c is adjacent to an arbitrary subset of vertices of $\{p_3, \ldots, p_{t-2}\}$.

The gem is universally forbidden. We mention this result explicitly since it improves the run-time of the algorithm in practice.

Theorem 4. *k-DH-Graph are gem-free.*

Proof. Let G be k-distance hereditary. Consider a binary tree-decomposition. Assume that there is a gem, and consider a lowest branch rooted at some line e that contains the vertices of the gem in the leaves. The twinset of e is obtained from two twinsets T_1 and T_2 at the children. We may assume that the universal vertex of the gem is contained in some color class C of T_1. By connectedness it follows that all the vertices of the P_4 must be contained in a color class of T_2 that is joined to C. This is not possible, since each color class of a twinset is a cograph. □

We end this section with the key observation that k-cographs have bounded diameter.

Theorem 5. *There exists a constant C_k such that k-cographs have no induced paths of length more than C_k.*

Proof. We prove this by way of contradiction. Assume that there exists a sequence $[P_1, P_2, \ldots]$ of paths of increasing lengths that are all k-cographs. For each i construct the graph P'_i by creating a twin of each of the two endpoints of P_i. The graphs P'_i are k-cographs since this class is closed under creating twins. Since the class of k-cographs is well-quasi-ordered by the induced subgraph relation, there must exist $i < j$ such that P'_i is an induced subgraph of P'_j. This is a contradiction. □

4 A Recognition Algorithm

A k-*DH-tree-decomposition* for a graph G consists of a labeled binary tree T and a bijection f from the leaves of T to the vertices of G. The leaves of T are labeled with a color from $\{1, \ldots, k\}$. To ease the description we equip the lines of T with twinsets. A line that is incident with a leaf has a twinset that consists of the colored vertex that is mapped to the leaf by f.

The internal points of T are labeled with a permutation $\sigma \in \mathrm{Sym}(k)$ that matches color classes of the left twinset with color classes of the right twinset. For each color $i \in \{1, \ldots, k\}$ the point has a label which says whether the vertices of color i in the left twinset are adjacent or not to the vertices of color $\sigma(i)$ in the right twinset. Furthermore, for each color $i \in \{1, \ldots, k\}$ the point has a label which indicates whether the color class of the left – and of the right twinset is copied into the twinset of the line that connects the point to its parent. This defines the twinset of the line incident with the point and its parent. For the root we only need to define the adjacencies between the twinsets of the left – and right subtree.

Note that the twinsets are defined by the bijection f and the labels of the internal nodes; we only introduced these to simplify the description. For a point p we call the

left twinset τ_ℓ and the right twinset τ_r the 'input twinsets' of the point. We call the twinset τ_o of the line that connects the point with its parent, the 'output twinset' of the point.

Theorem 6. *There exists an* $O(n^3)$ *time algorithm which checks if a graph* G *is* k*-distance hereditary.*

Proof. We describe a dynamic programming procedure. Consider a branch with a twinset P' of colored vertices. Since the diameter of k-cographs is bounded, we need to search up to some constant depth for a branch with a twinset Q' such that $P' + Q'$ induces a k-cograph. This proves the theorem. □

5 Some Final Remarks

We study graph classes such as k-DH-graphs in order to gain insight into why, and when some combinatorial problems are solvable for tree-structures such as these.

Consider a graph class $\mathcal{C}$ defined by the property that the graphs allow for a tree-decomposition such that every cutmatrix is shape-equivalent to some element of a finite collection C of matrices. Say the matrices of C have binary rank at most k. It is our aim to characterize the graphs of $\mathcal{C}$ by describing operators that act on the adjacency conditions and that transform k-DH-graphs into graphs of $\mathcal{C}$. Perhaps this leads to an elegant classification of the graphs of rankwidth k.

Notice that the definition of k-cographs can be made suitable by changing the adjacency conditions. Kruskal's theorem guarantees that any of these classes is characterized by a finite number of forbidden induced subgraphs. Furthermore, any of these classes has bounded diameter, which implies the $O(n^3)$ recognition for the tree-version with the adapted version of the twinsets.

References

1. Bouchet, A.: An efficient algorithm to recognize locally equivalent graphs. Combinatorica 11, 315–329 (1991)
2. Cameron, P.J.: Two-graphs and trees. Discrete Mathematics 127, 63–74 (1994)
3. Chang, M.-S., Hsieh, S.-Y., Chen, G.-H.: Dynamic programming on distance-hereditary graphs. In: Leong, H.-V., Jain, S., Imai, H. (eds.) ISAAC 1997. LNCS, vol. 1350, pp. 344–353. Springer, Heidelberg (1997)
4. Corneil, D. G., Lerchs, H., Stewart–Burlingham, L.: Complement reducible graphs. Discrete Applied Mathematics 3, 163–174 (1981)
5. Higman, G.: Ordering by divisibility in abstract algebras. Proceedings of the London Mathematical Society 2, 326–336 (1952)
6. Howorka, E.: A characterization of distance-hereditary graphs. Quarterly Journal of Mathematics 28, 417–420 (1977)
7. Hung, L.-J., Kloks, T.: de Bruijn's combinatorics. Manuscript (2009), http://www.cs.ccu.edu.tw/~hunglc/combinatorics.pdf
8. Hung, L.-J., Kloks, T.: On some simple widths. In: Rahman, M. S. (ed.) WALCOM 2010. LNCS, vol. 5942, pp. 204–215. Springer, Heidelberg (2009)

9. Hung, L.-J., Kloks, T.: Kruskalian graphs; k-cographs. In: Proceedings of CATS 2010. CRPIT, vol. 109, pp. 49–53 (2010)
10. Kruskal, J.: Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. Transactions of the American Mathematical Society 95, 210–225 (1960)
11. Milner, E.C.: Basic WQO– and BQO-theory. In: Rival, I. (ed.) Graphs and Orders, pp. 487–502. D. Reidel Publishing Company, Dordrecht (1985)
12. Oum, S.: Graphs of bounded rank-width, PhD Thesis. Princeton University (2005)
13. Pouzet, M.: Applications of well-quasi-ordering and better-quasi-ordering. In: Rival, I. (ed.) Graphs and Orders, pp. 503–519. D. Reidel Publishing Company, Dordrecht (1985)
14. Seidel, J.J.: Geometry and combinatorics. In: Corneil, D.G., Mathon, R. (eds.) Selected works of J.J. Seidel. Academic Press, London (1991)

Lower Bound on Average-Case Complexity of Inversion of Goldreich's Function by Drunken Backtracking Algorithms*

Dmitry Itsykson

Steklov Institute of Mathematics at St. Petersburg, 27 Fontanka,
St. Petersburg, 191023, Russia
dmitrits@pdmi.ras.ru

Abstract. We prove an exponential lower bound on the average time of inverting Goldreich's function by *drunken* [AHI05] backtracking algorithms; therefore we resolve the open question stated in [CEMT09]. The Goldreich's function [Gol00] has n binary inputs and n binary outputs. Every output depends on d inputs and is computed from them by the fixed predicate of arity d. Our Goldreich's function is based on an expander graph and on the nonliniar predicates of a special type. Drunken algorithm is a backtracking algorithm that somehow chooses a variable for splitting and randomly chooses the value for the variable to be investigated at first. Our proof technique significantly simplifies the one used in [AHI05] and in [CEMT09].

1 Introduction

In 2000 Goldreich introduced the candidate one-way function based on expanders [Gol00]. The function has n binary inputs and n binary outputs. Every output depends on d inputs and is computed from them by the fixed predicate of arity d. Goldreich suggested using expander graphs as graphs of dependency and a random predicate of arity d. There are many similarities between the Goldreich's function and the pseudorandom generator by Nisan and Wigderson [NW94].

One of the approaches for inverting of one-way function is the usage of contemporary SAT solvers [MZ06]. Almost all SAT algorithms are based on backtracking (so called DPLL (by names of authors Davis, Putnam, Logeman, Loveland) algorithms [DP60, DLL62]). Backtracking algorithm is a recursive algorithm. On each recursive call it simplifies an input formula F (without affecting its satisfiability), chooses a variable v and makes two recursive calls on the formulas $F[v := 1]$ and $F[v := 0]$ in some order. It returns the result "Satisfiable" if at least one of recursive calls returns "Satisfiable" (note that it is not necessary to make the second

* Partially supported by RFBR grants 08-01-00640 and 09-01-12137-ofi_m, the Fundamental research program of the Russian Academy of Sciences, the president of Russia grant "Leading Scientific Schools" NSh-4392.2008.1 and by Federal Target Programme "Scientific and scientific-pedagogical personnel of the innovative Russia" 2009-2013.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 204–215, 2010.

call if the first one was successful). Recursion stops if the input formula becomes trivial. That is, the algorithm is only allowed to backtrack when unsatisfiability in the current branch is proved. A backtracking algorithm is defined by simplification rules and two heuristics: the heuristic **A** chooses a variable for splitting and the heuristic **B** chooses a value that will be investigated first.

Lower bounds on the running time of backtracking algorithms on unsatisfiable formulas follow from lower bounds on the size of resolution proofs [Tse68]. Unsatisfiable formulas based on a pseudorandom generator by Nisan and Wigderson are used for proving lower bounds in several propositional proof systems [ABSRW00]. Note that formulas that code the problem of inverting a one-way function are usually satisfiable. If we do not restrict a type of heuristics of backtracking algorithms, then the exponential lower bound on running time of backtracking algorithms implies $\mathbf{P} \neq \mathbf{NP}$ (otherwise heuristic **B** may compute the correct value of the variable in polynomial time).

The first unconditional lower bounds on running time of backtracking algorithms on satisfiable formulas were proved in [AHI05] for *myopic* and *drunken* algorithms. Heuristics in myopic algorithms are restricted in the following way: they are allowed to read only a small fraction of the formula precisely, but they can see the rest of the formula sketchy (for example in [AHI05] they don't see negations but have access to the statistics of number of positive and negative variable occurrences). Exponential lower bound on the running time of myopic algorithms was proved on the family of formulas that actually code the problem of inverting Goldreich's function based on linear predicate. In drunken algorithms heuristic **A** has no restriction (and actually it may be not computable), but heuristic **B** selects the value just at random. For drunken algorithms hard satisfiable formulas are based on any family of hard unsatisfiable formulas.

Goldreich's function based on a linear predicate is not very interesting since it may be inverted by Gaussian elimination. In the paper [CEMT09] the technique from [AHI05] was extended for proving lower bounds for myopic algorithms on nonlinear predicates. In particular it was proved in [CEMT09] that any myopic algorithm has exponential running time in average case when it solves a problem of inverting Goldreich's function based on the predicate $x_1 \oplus x_2 \oplus \cdots \oplus x_{d-2} \oplus x_{d-1}x_d$. The paper [CEMT09] also presents results of experimental analysis of running time of contemporary SAT solvers on the problem of inverting Goldreich's function with the above predicate. Their analysis shows that these formulas are hard for MiniSAT 2.0 [EB05, ES03]. The question of exponential lower bound on inverting Goldreich's function by a drunken algorithm was left open in [CEMT09]. In this paper we give an answer on this question. In particular we prove that the average running time of drunken algorithms on formulas that code the problem of inverting Goldreich's function based on a random graph and the predicate $x_1 \oplus \cdots \oplus x_{d-k} \oplus Q(x_{d-k+1}, \ldots, x_d)$ is exponential with high probability. Here Q is an arbitrary predicate of arity k, $k+1 < \frac{d}{4}$ and d is a constant large enough.

The proof strategy is as follows: at first we prove a lower bound for unsatisfiable formulas using the technique from [BSW01], then we show that it is

possible to introduce some superstructure on drunken algorithms, which does not increase the running time but guarantees that in the first several steps the algorithm does not backtrack. After that we show that with high probability Goldreich's function has a small number of pre-images and with high probability the algorithm with superstructure falls into an unsatisfiable formula and we will apply a lower bound for unsatisfiable formulas. The general plan of our proof is the same as it was in [AHI05] and [CEMT09], but the resulting proof is substantially simpler.

We also show that drunken algorithms are powerful enough: they may solve satisfiable Tseitin formulas in polynomial time while unsatisfiable Tseitin formulas are hard for all backtracking algorithms. Drunken algorithms may also simulate the pure literal simplification rule while myopic algorithms from [CEMT09] are not allowed to use this rule.

2 Preliminaries

Propositional variable is one that has 0/1-value, literal is either a variable or its negation. A clause is a disjunction of literals, a CNF formula is a conjunction of clauses. A k-CNF formula is a CNF formula in which all clauses contain at most k literals. The formula is satisfiable if there exists substitution for its variables such that the value of the formula becomes 1 (we call such substitution a satisfying assignment).

The set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$ we denote as $\mathfrak{F}_n$. For every function $f \in \mathfrak{F}_n$ and every string $b \in \{0,1\}^n$ the equality $f(x) = b$ may be written as a CNF formula with propositional variables $x_1, \ldots, x_n$. We denote such formula $\Phi_{f(x)=b}$.

In this paper $G(V, E)$ is a bipartite graph with multi-edges. The vertices of G are divided into two parts: $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$; the number of vertices in each part is n. Vertices in X are inputs and vertices in Y are outputs. Every vertex in Y has degree d.

Proposition 1 (Chernoff-Hoefding bounds). *Independent and equally distributed random variables $X_1, X_2, \ldots, X_N$, such that for every $1 \le i \le N$ $X_i \in \{0,1\}$ and $E[X_i] = \mu$, satisfy the following inequality:* $\Pr\{|\frac{\sum_{i=1}^{N} X_i}{N} - \mu| \ge \epsilon\} \le 2e^{-2\epsilon^2 N}$.

Goldreich introduced the function $g : \{0,1\}^n \to \{0,1\}^n$ based on a bipartite graph $G(V, E)$ and a predicate $P : \{0,1\}^d \to \{0,1\}$ [Gol00]. Every string from $\{0,1\}^n$ defines a value of inputs $\{x_1, x_2, \ldots, x_n\}$; the value $(g(x))_j$ (j-th symbol of $g(x)$) may be computed as follows: if y_j is adjacent with $x_{j_1}, x_{j_2}, \ldots, x_{j_d}$, then $(g(x))_j = P(x_{j_1}, x_{j_2}, \ldots, x_{j_d})$. We assume that every vertex in Y has some order on the incoming edges. Goldreich suggested using random predicates and expanders.

The problem of inverting of function g on the string b (i.e. equation $g(x) = b$) may be written as a d-CNF formula $\Phi_{g(x)=b}$: every equality $P(x_{j_1}, x_{j_2}, \ldots, x_{j_d}) = b_j$ we write as a d-CNF formula of at most 2^d clauses. For every set of vertices

$A \subseteq Y$ the formula $\Phi^A_{g(x)=b}$ denotes the subformula of $\Phi_{g(x)=b}$ that consists of all clauses corresponding to vertices in A.

Let G be a bipartite graph, its vertices are split into two parts $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$. For $A \subseteq Y$ we denote the set of all vertices in X that are connected with at least one vertex in A as $\Gamma(A)$ (neighbours of A); and denote the set of all vertices in X that are connected with exactly one vertex in A by one edge as $\delta(A)$ (boundary of A).

Definition 1. *The graph G is a (r, d, c)-expander, if 1) the degree of any vertex in Y is equal to d; 2) for any set $A \subseteq Y, |A| \leq r$ we have $\Gamma(A) \geq c|A|$. The graph G is called a (r, d, c)-boundary expander if the second condition is replaced by: 2) for any set $A \subseteq Y, |A| \leq r$ we have $\delta(A) \geq c|A|$.*

Lemma 1 ([AHI05], lemma 1). *Every (r, d, c)-expander is also a $(r, d, 2c-d)$-boundary expander.*

Lemma 2 ([HLW06], lemma 1.9). *For $d \geq 32$, for all big enough n a random bipartite d-regular graph, where parts X and Y contain n vertices is a $(\frac{n}{10d}, d, \frac{5}{8}d)$-expander with probability 0.9 if for every vertex in Y, d edges are chosen independently at random (with repetitions).*

Corollary 1. *In terms of Lemma 2 this graph is a $(\frac{n}{10d}, d, \frac{1}{4}d)$-boundary expander.*

Proof. Follows from Lemma 1.

Let $f \in \mathfrak{F}_n$ be some function. The problem of finding a satisfying assignment of $\Phi_{f(x)=b}$ and the problem of finding an element in $f^{-1}(b)$ are equivalent.

We consider a wide class of SAT algorithms: backtracking algorithms. A backtracking algorithm is defined by two *heuristics* (procedures): 1) Procedure **A** maps a CNF formula to one of its variables. (This is a variable for splitting). 2) Procedure **B** maps CNF formula and its variable to $\{0, 1\}$. (This value will be investigated at first).

An algorithm may also use some syntactic *simplification rules*. Simplification rules may modify the formula without affecting its satisfiability and also make substitutions to its variables if their values can be inferred from a satisfiability of the initial formula.

The backtracking algorithm is a recursive algorithm. Its input is a formula φ and a partial substitution ρ.

Algorithm 1. *Input: formula φ and substitution ρ*

- *Simplify φ by means of simplification rules (assume that simplification rules change φ and ρ; all variables that are substituted by ρ should be deleted from φ).*
- *If current formula is empty (that is, all its clauses are satisfied by ρ), then return ρ. If formula contains an empty clause (unsatisfiable), then return "formula is unsatisfiable".*

- $x_j := \mathbf{A}(\varphi)$; $c := \mathbf{B}(\varphi, x_j)$
- *Make a recursive call with the input* $(\varphi[x_j := c], \rho \cup \{x_j := c\})$*, if the result is "formula is unsatisfiable", then make a recursive call with the input* $(\varphi[x_j := 1 - c], \rho \cup \{x_j := 1 - c\})$ *and return its result, otherwise return the result of the first recursive call.*

Definition 2. *Drunken algorithms [AHI05] are backtracking algorithms, where the heuristic* $\mathbf{A}$ *may be arbitrary (even not computable) and the heuristic* $\mathbf{B}$ *chooses the value of variable at random with equal probabilities. Simplification rules: 1)* Unit clause elimination*: if formula contains a clause with only one literal, then make a substitution that satisfies that clause. 2)* Pure literals rule*: if formula contains a variable that has only positive or only negative occurrences, then substitute it with the corresponding value.*

In Section 3 we will show that we may assume that a drunken algorithm does not use the above simplification rules. Usages of them may be replaced by an appropriate choice of splitting variable.

Running time of a backtracking algorithm for a given sequence of random bits is the number of recursive calls.

3 What Can We Solve by Drunken Algorithms?

In this section we show that it is possible to modify a drunken algorithm in such a way that it will not use pure literals and unit clause elimination rules while its running time is increased only polynomially. We also show that there exists a drunken algorithm that solves satisfiable Tseitin formulas in polynomial time.

Proposition 2. *For any drunken algorithm* $\mathcal{A}$ *there exists another drunken algorithm* $\mathcal{B}$*, that does not use unit clause elimination rule. The running time of algorithm* $\mathcal{B}$ *(for a given sequence of random bits) is at most the running time of* $\mathcal{A}$ *(for the same sequence of random bits) times* n*, where* n *is the number of variables in the input formula.*

Proposition 3. *For any drunken algorithm* $\mathcal{A}$ *there exists another drunken algorithm* $\mathcal{C}$ *that does not use unit clause elimination and pure literals rules. The running time of algorithm* $\mathcal{C}$ *is at most the running time of* $\mathcal{A}$ *times* n^2*, where* n *is the number of variables in the input formula.*

Further we assume that drunken algorithms don't use simplifications rules.

Now we show that drunken algorithms may efficiently solve Tseitin formulas. Tseitin formula is based on a simple connected undirected graph $H(V, E)$ with degree bounded by a constant d. Every edge $e \in E$ has the corresponding propositional variable p_e. There is a function $f : V \to \{0, 1\}$; for every vertex $v \in V$ we put down a formula in CNF that codes an equality $\bigoplus_{u \in V : (u,v) \in E} p_{(u,v)} = f(v)$. ($\oplus$ denotes the summation modulo 2). The conjunction of formulas described above is called Tseitin formula. If $\bigoplus_{v \in V} f(v) = 1$, then Tseitin formula is unsatisfiable.

Indeed, if we sum (modulo 2) all equalities stated in vertices we get $0 = 1$ since every variable has exactly 2 occurrences. If $\bigoplus_{v \in V} f(v) = 0$, then Tseitin formula is satisfiable ([Urq87], Lemma 4.1).

Satisfiable Tseitin formulas may be solved in polynomial time by appropriate drunken algorithm as follows. We assume that the substitution of a variable removes the corresponding edge from the graph. While the graph contains cycles, the drunken algorithm chooses an edge from a cycle for the substitution. Note that after each substitution the current formula remains satisfiable Tseitin formula (if we substitute 0, the function f for the substituted formula is the same as for the original one and if we substitute 1, the function f for substituted formula differs from the original one in 2 ends of the edge). If the graph becomes a tree, then it contains a vertex of degree 1, then the formula contains a unit clause and the whole formula can be solved by application of unit clause elimination rules (we may remove them by Proposition 2).

4 Behavior of Drunken Algorithms on Unsatisfiable Formulas

Behavior of backtracking algorithms on unsatisfiable formulas is closely connected with the resolution proof system. The resolution proof system is used for proving of unsatisfiability of CNF formulas. The proof of unsatisfiability of formula φ in the resolution proof system is a sequence of clauses, every clause in this sequence is either a clause of φ or a result of application of the resolution rule to two previous clauses; and the last clause in the sequence is an empty clause (a contradiction). The resolution of two clauses $(l_1 \vee l_2 \vee \cdots \vee l_n)$ and $(l'_1 \vee l'_2 \vee \cdots \vee l'_m)$ where $l'_m = \neg l_n$ is the clause $(l_1 \vee \cdots \vee l_{n-1} \vee l'_1 \vee \cdots \vee l'_{m-1})$. The proof is called treelike if every inferred clause is used as the premise of the resolution rule at most once.

The running of every drunken algorithm on the unsatisfiable formula corresponds to the splitting tree. Vertices of the tree are marked with variables that are chosen for splitting. There are two outgoing edges from every vertex except leaves; one of the edges is marked with 0, the other edge is marked with 1. In every leaf at least one of clauses of initial formula is refuted. The running time of a drunken algorithm is the size of the splitting tree (note that if formula is unsatisfiable then the algorithm should investigate the whole tree and it's number of steps is the same for all random choices).

The following statement is well known.

Proposition 4. *The running time of a drunken algorithm on unsatisfiable formula is at least the size (number of clauses) of the shortest treelike resolution proof.*

Ben-Sasson and Wigderson in [BSW01] introduced the notion of width of the proof. The width of a clause is the number of literals in it. The width of a CNF formula is the width of its widest clause. The width of a resolution proof is the width of its widest clause.

Theorem 1 ([BSW01], corollary 3.4). *The size of a treelike resolution refutation of the formula φ is at least 2^{w-w_φ}, where w is the minimum width of the resolution refutation of φ and w_φ is the width of φ.*

Let G be a boundary (r, d, c)-expander. We associate a proposition variable with every vertex in set X. Let every vertex y_j in set Y have a CNF formula that depends on variables adjacent to y_j. We denote the formula in the vertex y_j as φ_j. Obviously the width of φ_j is at most d. The conjunction of all formulas that correspond to the vertices Y we denote Φ. For any subset $A \subseteq Y$ the conjunction of all formulas that correspond to the vertices in A we denote as Φ^A.

We say that a variable is *sensible* if by changing its value we change the value of the formula (for every assignment of values of other variables).

Theorem 2. *Let every formula φ_j contain at most k insensible variables; ρ is a partial assignment to variables of X such that formula $\Phi|_\rho$ is unsatisfiable and for any set of vertices $A \subseteq Y$, $|A| < \frac{r}{2}$, the formula $\Phi|_\rho^A$ is satisfiable. Then any resolution proof of $\Phi|_\rho$ has width at least $\frac{(c-k)r}{4} - |\rho|$.*

Proof. We consider Ben-Sason-Wigderson measure μ that is defined on the clauses of resolution proof of $\Phi|_\rho$. $\mu(D)$ is the size of the minimal set of vertices A such that clause D is a semantic implication of $\Phi^A|_\rho$ (it means that every satisfying assignment of $\Phi^A|_\rho$ also satisfies D). The measure μ is semiadditive: if clause D is a resolvent of clauses C_1 and C_2, then $\mu(D) \leq \mu(C_1) + \mu(C_2)$. Since for every set $A \subseteq Y$ such that $|A| < \frac{r}{2}$, formula $\Phi|_\rho^A$ is satisfiable, then the measure of an empty clause is at least $\frac{r}{2}$. Semiadditivity implies that there exists a clause C such that $\frac{r}{2} > \mu(C) \geq \frac{r}{4}$ for r large enough. Let A be the minimal set of vertices such that $\Phi|_\rho^A$ semantically implies C, i.e. $|A| = \mu(C) \geq \frac{r}{4}$. Since G is a (r, d, c)-boundary expander we have $\delta(A) \geq c|A|$. $\delta(A)$ is a set of variables that have exactly one occurrence in the formulas corresponding to the set A. There are at least $(c-k)|A|$ variables among them that are sensible for at least one vertex of A. There are at least $(c-k)|A| - |\rho|$ sensible variables in the formula $\Phi|_\rho^A$. Now we will show that the clause C contains all sensible variables. Suppose for contradiction that there is a variable x_j that is sensible for a vertex $v \in A$ and the clause C doesn't contain x_j. Consider the set $A \setminus \{v\}$. It doesn't semantically imply C, therefore there exists such an assignment that satisfies all formulas for $A \setminus \{v\}$ and doesn't satisfy C. We may change the value of x_j in this assignment in such way that the resulting assignment satisfies all formulas in A and doesn't satisfy C. The later contradicts the fact that C is a semantic implication of A. □

Corollary 2. *The size of the splitting tree of $\Phi|_\rho$ is at least $2^{\frac{(c-k)r}{4} - |\rho| - d}$.*

Proof. Follows from the Theorem 2, Theorem 1 and Proposition 4. □

5 Behaviour of Drunken Algorithms on Satisfiable Formulas

Let G be a bipartite boundary (r, d, c)-expander. Let $c > k + 1$.

Definition 3. *Let $J \subseteq X$, the set of vertices $I \subseteq Y$ is called k-closure of the set J if there is a finite sequence of sets $I_1, I_2, \ldots, I_m$ (we denote $C_\ell = \bigcup_{1 \le i \le \ell} I_i$, $C_0 = \emptyset$), such that the following properties are satisfied:*

- *$I_\ell \subseteq Y$ and $0 < |I_\ell| \le \frac{r}{2}$ for all $1 \le \ell \le m$;*
- *$I_i \cap I_j = \emptyset$ for all $1 \le i, j \le m$;*
- *$|\delta(I_\ell) \setminus (\Gamma(C_{\ell-1}) \cup J)| \le (1+k)|I_l|$; for all $1 \le \ell \le m$;*
- *for all $I' \subseteq Y \setminus C_m$ if $0 < |I'| \le \frac{r}{2}$, then $|\delta(I') \setminus (\Gamma(C_m) \cup J)| > (1+k)|I'|$;*
- *$I = C_m$.*

The set of all k-closures of the set J we denote as $Cl^k(J)$.

Lemma 3. *1. For every set $J \subseteq X$ there exists a k-closure. 2. Let $J_1 \subseteq J_2$, then for every $I_1 \in Cl^k(J_1)$ there exists $I_2 \in Cl^k(J_2)$ such that $I_1 \subseteq I_2$*

Lemma 4 ([AHI05]). *Let $|J| < \frac{(c-k-1)r}{2}$, then for every set $I \in Cl^k(J)$ the inequality $|I| \le (c-k-1)^{-1}|J|$ is satisfied*

We assume that a drunken algorithm creates a splitting tree during the execution. At the beginning it creates the root of the tree that becomes the current vertex. Each vertex of the tree has a current formula, each edge is associated with the assignment of one variable. The path from the root to the vertex defines a partial assignment that is a union of all assignments along this path. The current vertex of the tree becomes a leaf if the current formula is either already satisfied (i.e. all its clauses are satisfied) or contains an empty clause (i.e. a contradiction). In the first case the algorithm prints a satisfying assignment and stops. In the second case the algorithm looks for the closest backtrack point along the path to the root and considers the vertex with that backtrack point as current (in this case we say that the algorithm backtracks). If there are no vertices with a backtrack point, then the algorithm stops and returns "formula is unsatisfiable". If the current formula is not trivially unsatisfiable or satisfiable, then the algorithm chooses the variable for splitting and the value for splitting according to heuristics **A** and **B**, puts a backtrack point in the vertex and creates a descendant that corresponds to the assignment that was chosen; this descendant becomes the current vertex. If the current vertex has a backtrack point, then the algorithm removes this point and creates a descendant corresponding to the assignment that was not investigated in that vertex.

Now we describe a superstructure of drunken algorithms that slightly modifies their behavior on the formula $\Phi_{g(x)=b}$ for several first steps. After this the superstructure finishes its work and the algorithm continues its normal work without modification. We claim that the running time of the algorithm with the superstructure is not increased. (The last statement is not very clear since our algorithm uses random bits. In our case it should be understood in the following way: the original algorithm uses p random bits and the algorithm with the superstructure uses q bits where $q \le p$, and for every string of random bits r of length q there are 2^{p-q} strings of random bits of length p such that the running time of the original algorithm on those strings is at least the running time of the

algorithm with the superstructure on the string r. The above correspondence covers all strings of length p.)

The superstructure has a partial assignment π. If π assigns some value to the variable x, then we call x forced. If a drunken algorithm tries to make an assignment to a forced variable that differs from the value in π, then the superstructure doesn't allow this. In other words, the superstructure cuts off a subtree but we guarantee that the cut subtree is unsatisfiable (it does not contain a satisfying assignment). We also guarantee that while the superstructure is working there are no backtrackings (all backtrackings are left in the cut subtrees).

Let's formally describe the superstructure. Let drunken algorithm $\mathcal{A}$ get as input a satisfiable formula $\Phi_{g(x)=b}$, where G is a (r, d, c)-boundary expander, k is the number of insensible variables of the predicate P and $k + 1 < c$.

1. $J := \emptyset$, $I := \emptyset$, $\rho := \emptyset$ (current substitution)
2. $\pi := \emptyset$ (initially there are no forced variables).
3. While $|J| < \frac{r(c-k-1)}{16d}$ and $|\rho| < n$ do
   (a) If algorithm $\mathcal{A}$ is ready to finish its work or it wants to backtrack, then break.
   (b) Let $\mathcal{A}$ choose a variable x_j for the splitting.
   (c) If variable x_j is forced and π contains assignment $x_j := a$, then $\rho := \rho \cup \{x_j := a\}$. In the splitting tree we add one decender and we do not put a backtracking point.
   (d) Otherwise the variable x_j is not forced, then
      – Let $\mathcal{A}$ chooses value a, then $J := J \cup \{x_j\}$, $\rho := \rho \cup \{x_j := a\}$. We put backtrack point in the current vertex.
      – We extend I to the element of $Cl^k(J)$ (it is possible by the item (2) of Lemma 3).
      – For all variables x_j from $\Gamma(I)$ and $a \in \{0, 1\}$, if the value $x_j = a$ is a semantic implication of formula $\Phi^I_{g(x)=b}|_\rho$, then $\pi := \pi \cup \{x_j := a\}$. (Formally it is possible that the formula $\Phi^I_{g(x)=b}|_\rho$ implies both $x_j = 0$ and $x_j = 1$. In this case we add to π only one of them; later we show that this case is impossible).
      – Create a descendant in the tree that corresponds to the made assignment, this descendant becomes the current vertex.
4. Simulate $\mathcal{A}$ without changes on the formula $\Phi_{g(x)=b}|_\rho$ in the current vertex of the tree.

Let the loop at the 3rd step of the superstructure be executed (up to the end) t times. For $0 \le i \le t$ we denote as J_i, I_i, ρ_i the values of the variables J, I, ρ before the $(i+1)$-th iteration of the loop at the 3rd step. (I_t, J_t, ρ_t are the values after t-th iteration of the loop).

The following Lemma implies that during the work of the superstructure the algorithm does not backtrack.

Lemma 5. *For every $0 \le i \le t$ and for any subset $A \subseteq Y$, such that $|A| \le \frac{r}{2}$, the formula $\Phi^A_{g(x)=b}|_{\rho_i}$ is satisfiable and $I_i = Cl^k(J_i)$.*

Proof. Proof by induction on i. Let's verify the condition for $i = 0$, $\rho_0 = \emptyset$. Proof by contradiction. We consider the smallest set $A \subset Y$, $|A| \leq \frac{r}{2}$ such that the formula $\Phi^A_{g(x)=b}$ is unsatisfiable. Since G is a boundary expander, then $|\delta(A)| \geq c|A|$, and therefore there are at least $(c-k)|A|$ sensible boundary variables for formulas from the set A. By changing of the value of the boundary variable we may change the value of some formula from A, that is this formula (and a vertex) may be removed from A; the latest contradicts to the minimality of A.

The induction step. Proof by contradiction. Consider the minimum set $A \subseteq Y$, $|A| \leq \frac{r}{2}$ such that the formula $\Phi^A_{g(x)=b}|_{\rho_{i+1}}$ is unsatisfiable. Let $A_1 = A \cap I_{i+1}$, $A_2 = A \setminus I_{i+1}$. If A_2 is not empty, then the definition 3 implies $|\delta(A_2) \setminus (\Gamma(I_{i+1}) \cup J)| > (k+1)|A_2|$, therefore we may remove at least one vertex from A_2 since one of the formulas in A_2 may be satisfied by a sensible boundary variable (a sensible boundary variable exists since each vertex has at most k insensible variables) that contradicts to the minimality of A. Therefore $A_2 = \emptyset$ and $A \subseteq I_{i+1}$.

We split A on $A' = A \cap I_i$ and $A'' = A \setminus I_i$. Let $A'' = \emptyset$. By the induction hypothesis the formula $\Phi^{A'}_{g(x)=b}|_{\rho_i}$ is satisfiable since $|A'| \leq |A| \leq \frac{r}{2}$. The formula $\Phi^{A'}_{g(x)=b}|_{\rho_{i+1}}$ is satisfiable since ρ_{i+1} differs from ρ_i in only one assignment and the variables are forced by the substitution π only if their values are semantic implications of the formula $\Phi^{I_i}_{g(x)=b}|_{\rho_i}$. The latest means that it is impossible for the algorithm $\mathcal{A}$ (on the $(i+1)$-the iteration of the loop) to make $\Phi^{I_i}_{g(x)=b}|_{\rho_i}$ unsatisfiable by one assignment.

Let $A'' \neq \emptyset$. $|A''| \leq \frac{r}{2}$ and $A'' \cap I_i = \emptyset$ imply $|\delta(A'') \setminus (\Gamma(I_i) \cup J)| > (k+1)|A''|$, that is the set A'' contains at least two sensible boundary variables (that are not in $\Gamma(I_i) \cup J$), therefore after one assignment there is at least one sensible boundary variable. We can remove at least one vertex from A'' with $\Phi^A_{g(x)=b}|_{\rho_{i+1}}$ remaining unsatisfiable. This contradicts the minimality of A. □

Now we show that the algorithm can't find the satisfying assignment during the work of the superstructure. During the work of the superstructure for every $0 \leq i \leq t$ the inequality $|J_i| \leq \frac{r(c-k-1)}{16d}$ is satisfied. Lemma 4 implies $|I_i| \leq \frac{r}{16d}$, hence $|\Gamma(I_i)| \leq \frac{r}{16}$. The number of variables that were assigned during the work of the superstructure is at most $|\Gamma(I_t)| \cup |J_t| \leq \frac{r}{8}$ ($|J_t| \leq \frac{r}{16}$ since $c \leq d$ in the (r, d, c)-boundary expander). This is not enough to satisfy the formula, since any subset $A \subseteq Y$ of size r contains at least r sensible variables. To satisfy the formula we should assign a value to all sensible variables.

Lemma 6. *Let g be the Goldreich's function based on the (r, d, c)-boundary expander G and the predicate P, which has at most k insensible variables and $c > k + 1$. Let b be the n-th bit string such that the equation $g(x) = b$ has at most $2^{\frac{r(c-k-1)}{64d}}$ solutions. Then with probability $1 - 2^{-\Omega(r)}$ the running time of a drunken algorithm on the formula $\Phi_{g(x)=b}$ is $2^{\Omega(r)}$ (in asymptotic notations c, k and d are considered to be constants).*

Proof. Since the superstructure doesn't increase the running time it is sufficient to estimate the running time of the algorithm with the superstructure. Since

the superstructure works until $|J| \leq \frac{r(c-k-1)}{16d}$ and $|I| \in Cl^k(J)$ we have that Lemma 4 implies $|I| \leq \frac{r}{16d}$. Therefore $|\Gamma(I)| \leq \frac{r}{16}$. Hence during the work of the superstructure the number of assignments made is at most $|\Gamma(I)| + |J| \leq \frac{r}{8}$. The set J corresponds to splittings (to assignments that put a backtrack point). The first substituted values are chosen at random. The substitution of one variable $x_j := a$ is *lucky* for a formula φ if it agrees with at least half of satisfying assignments of φ and *unlucky* otherwise.

During the work of the superstructure a drunken algorithm makes $\frac{r(c-k-1)}{16d}$ assignments choosing the values at random. With probability $\frac{1}{2}$ the chosen value is unlucky, that is, the number of satisfying assignments decreases at least by half. Chernoff bound implies that with probability $1 - 2^{-\Omega(r)}$ there are at least $\frac{r(c-k-1)}{64d}$ unlucky assignments. Thus with probability $1 - 2^{-\Omega(r)}$ after the work of the superstructure the current formula is unsatisfiable; the size of substitution ρ is at most $\frac{r}{8}$. The statement of the Lemma follows from Corollary 2, where we've proved the lower bound for unsatisfiable formulas. □

Theorem 3 (cf. [CEMT09], theorem 4.1). *Let $P_d(x_1, x_2, \ldots, x_d) = x_1 \oplus x_2 \oplus \cdots \oplus x_{d-k} \oplus Q(x_{d-k+1}, \ldots, x_d)$, where Q is an arbitrary predicate of arity k and $k+1 < \frac{d}{4}$. The graph G is obtained randomly in the following way: for every vertex in Y we choose independently at random d edges to X (repetitions are allowed). Then $\mathrm{E}[\#(x, y) \mid g(x) = g(y)] = 2^{(1+2^{-\Omega(d)})n}$, where g is a Goldreich's function based on G and the predicate P_d.*

Now we prove the main theorem:

Theorem 4. *Let $P_d(x_1, x_2, \ldots, x_d) = x_1 \oplus \ldots x_{d-k} \oplus Q(x_{d-k+1}, \ldots, x_d)$, where Q is an arbitrary predicate of arity k and $k + 1 < \frac{d}{4}$. For all d large enough and all n large enough the random graph G with probability at least 0.85 has the following property. For every drunken algorithm $\mathcal{A}$, $\Pr_{y \leftarrow U_n}[\Pr[t^{\mathcal{A}}_{\Phi_{g(x)=g(y)}} > 2^{\Omega(n)}] > 1 - 2^{-\Omega(n)}] > 0.9$, where $t^{\mathcal{A}}_{\Phi}$ denotes the running time of the algorithm $\mathcal{A}$ on the formula Φ.*

Proof. By the corollary 1 the random graph with probability 0.9 is a $(\frac{n}{10d}, d, \frac{1}{4}d)$-boundary expander. Theorem 3 implies that the average number of pairs x and y that $g(x) = g(y)$ is $2^{(1+2^{-\Omega(d)})n}$, where the averaging is on random graphs. The Markov inequality implies that with probability at least 0.95 for the random graph the number of pairs x and y such that $g(x) = g(y)$ is $2^{(1+2^{-\Omega(d)})n}$ (the constant is hidden in $\Omega(d)$). Therefore with probability at least 0.85 the random graph is a boundary expander and the upper bound on the number of pairs with equal values of g holds. We fix such graph G. The Markov inequality implies that for at least 0.9 fraction of strings $y \in \{0,1\}^n$ the following inequality $|g^{-1}(g(y))| < 2^{2^{-\Omega(d)}n}$ is satisfied. The predicate P contains at most k insensible variables (insensible variables are among $x_{d-k+1}, \ldots, x_d$), then Lemma 6 implies that the running time of any drunken algorithm on the formula $\Phi_{g(x)=g(y)}$ is at least $2^{\Omega(n)}$ with probability $1 - 2^{-\Omega(n)}$. □

Acknowledgement. The author thanks Ilya Mironov for bringing attention to [CEMT09], Edward A. Hirsch for fruitful discutions and also thanks Sofia Alexandrova, Anya Luter and Ilya Posov for useful comments that improved the readability of the paper.

References

[ABSRW00] Alekhnovich, M., Ben-Sasson, E., Razborov, A.A., Wigderson, A.: Pseudorandom generators in propositional proof complexity. In: FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Washington, DC, USA, p. 43. IEEE Computer Society, Los Alamitos (2000)

[AHI05] Alekhnovich, M., Hirsch, E.A., Itsykson, D.: Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. J. Autom. Reason. 35(1-3), 51–72 (2005)

[BSW01] Ben-Sasson, E., Wigderson, A.: Short proofs are narrow — resolution made simple. Journal of ACM 48(2), 149–169 (2001)

[CEMT09] Cook, J., Etesami, O., Miller, R., Trevisan, L.: Goldreich's one-way function candidate and myopic backtracking algorithms. In: Reingold, O. (ed.) Theory of Cryptography. LNCS, vol. 5444, pp. 521–538. Springer, Heidelberg (2009)

[DLL62] Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM 5, 394–397 (1962)

[DP60] Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM 7, 201–215 (1960)

[EB05] Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. Theory and Applications of Satisfiability Testing, 61–75 (2005)

[ES03] Een, N., Sorensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)

[Gol00] Goldreich, O.: Candidate one-way functions based on expander graphs. Technical Report 00-090, Electronic Colloquium on Computational Complexity (2000)

[HLW06] Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bulletin of the American Mathematical Society 43, 439–561 (2006)

[MZ06] Mironov, I., Zhang, L.: Applications of SAT solvers to cryptanalysis of hash functions. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 102–115. Springer, Heidelberg (2006)

[NW94] Nisan, N., Wigderson, A.: Hardness vs. randomness. Journal of Computer and System Sciences 49, 149–167 (1994)

[Tse68] Tseitin, G.S.: On the complexity of derivation in the propositional calculus. Zapiski nauchnykh seminarov LOMI 8, 234–259 (1968); English translation of this volume: Consultants Bureau, N.Y., pp. 115–125 (1970)

[Urq87] Urquhart, A.: Hard examples for resolution. J. ACM 34(1), 209–219 (1987)

A SAT Based Effective Algorithm for the Directed Hamiltonian Cycle Problem

Gerold Jäger[1] and Weixiong Zhang[2]

[1] Computer Science Institute
Christian-Albrechts-University of Kiel
D-24118 Kiel, Germany
gej@informatik.uni-kiel.de
[2] Department of Computer Science/Department of Genetics
Washington University
St. Louis, Missouri 63130-4899, United States
weixiong.zhang@wustl.edu

Abstract. The Hamiltonian cycle problem (HCP) is an important combinatorial problem with applications in many areas. While thorough theoretical and experimental analyses have been made on the HCP in undirected graphs, little is known for the HCP in directed graphs (DHCP). The contribution of this work is an effective algorithm for the DHCP. Our algorithm explores and exploits the close relationship between the DHCP and the Assignment Problem (AP) and utilizes a technique based on Boolean satisfiability (SAT). By combining effective algorithms for the AP and SAT, our algorithm significantly outperforms previous exact DHCP algorithms including an algorithm based on the award-winning Concorde TSP algorithm.

1 Introduction

An undirected graph $G = (V, E)$ is *Hamiltonian*, if it contains a *Hamiltonian cycle* (HC), a cycle that visits each vertex exactly once. Given a graph, the *Hamiltonian cycle problem* (*HCP*) is to find a HC or to prove that no HC exists in the graph. The decision version of the HCP is among the first problems that were proven to be $\mathcal{NP}$-complete [20]. HCP is a well-known problem with many applications in different areas, e.g., problems in game theory as the Hamiltonian cycle game [29], the problem of finding a knight's tour on a chessboard [15], and problems in bioinformatics as DNA Physical Mapping [14]. Much research has been done on the HCP in undirected graphs; see [5,6,13,30,32] for reviews. In particular, many heuristics and exact algorithms have been developed for the HCP [1,4,8,28,30,33]. One effective algorithm for the HCP is based on the related Traveling Salesman Problem (TSP) in an undirected weighted graph, which is the problem of finding a HC with minimum total weight. The HCP is also a canonical problem for understanding intrinsic properties of combinatorial problems. One such problem property is the so called *phase transition*. Consider an undirected graph $G_{n,m}$ with m edges randomly chosen from all

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 216–227, 2010.

possible $n(n-1)/2$ edges over n vertices. It is expected that when keeping the number of vertices n a constant while increasing the number of edges m, the probability that a random graph $G_{n,m}$ is Hamiltonian increases from 0 to 1. Surprisingly, the probability of being Hamiltonian for $G_{n,m}$ exhibits a sharp, dramatic transition from 0 to 1, and the transition approximately occurs when $m = \lceil c \cdot n \cdot (\log n + \log\log n)/2 \rfloor$ [23].

In this study we consider the HCP in directed graphs, which we call *directed HCP*, or *DHCP* for short. In contrast to the work on the HCP for undirected graphs, the research on the DHCP is limited [3,22]. Probabilistic heuristics for DHCP were proposed in [1,9]. In 1983, the first exact algorithm for the DHCP was developed by Martello [26]. This algorithm outputs a fixed number h HCs or reports that it cannot find h HCs in a given directed graph. By setting $h = 1$, this gives rise to an algorithm for the DHCP. For the DHCP, a phase transition result similar to that of the HCP has been obtained as well, namely the phase transition appears when $m = \lceil c \cdot n \cdot (\log n + \log\log n) \rfloor$ [27]. Note that the research on the TSP has also alluded to a DHCP algorithm. Using the technique of 2-point reduction, the asymmetric TSP (ATSP) – where the distance from city i to city j may not be necessarily equal to that from j to i – can be converted to the symmetric TSP, with the number of vertices being doubled [18]. Using this transformation, we can determine whether a directed graph is Hamiltonian by solving the symmetric TSP using the renowned Concorde algorithm [2,34], which is the best TSP solver for large TSP instances.

In this paper, we present an effective exact algorithm for the DHCP. In our algorithm, we utilize methods for two well-known combinatorial problems, i.e., Assignment Problem (AP) and Boolean satisfiability (SAT); we therefore denote this algorithm by AP-SAT. Using random graphs and many real world instances, we experimentally compare the AP-SAT algorithm to the DHCP algorithm of Martello [26] and with the TSP based approach that takes advantage of the TSP solver Concorde [2,34]. The results show that the AP-SAT algorithm significantly outperforms these algorithms.

2 The Algorithm

In the following let a directed unweighted graph $G = (V, E)$ be given. For our purpose of solving the DHCP, we consider the problem of determining whether or not there exists a collection of cycles visiting each vertex exactly once. We call this problem *directed Assignment Problem* or *DAP* for short. Our algorithm explores and exploits the intrinsic relationship between the DHCP and the DAP. More exactly, the AP-SAT algorithm searches for a HC in the space of DAP solutions. It first solves the DAP. If the DAP solution forms a HC, or no DAP solution exists, the algorithm terminates. If the DAP solver returns a solution that is not a HC, the algorithm then tries to patch the subcycles in the solution into a HC using the well-known Karp-Steele patching method [21]. If the algorithm has not terminated, these steps are iterated, with the only difference that another DAP solution might be found. For most cases that we considered in this

study, the algorithm can find HCs or determine no solution exists after these two steps. If the patching result is not a HC, the algorithm then attempts to enumerate the DAP solutions by formulating the DAP as a Boolean satisfiability problem and repeatedly solving the problem using a SAT solver and adding constraints to eliminate the DAP solutions that have been encountered [16]. We discuss the details of these steps in the rest of the section.

2.1 Solving the Assignment Problem

Given n vertices and a matrix $C = (c_{ij})_{1 \leq i,j \leq n} \in \mathbb{Z}^{n,n}$ of the costs between pairs of vertices, the Assignment Problem (AP) is to find a vertex permutation π^* such that $\pi^* = \arg\min \left\{ \sum_{i=1}^{n} c_{i,\pi(i)} : \pi \in \Pi_n \right\}$, where Π_n is the set of all permutations of $\{1, \ldots, n\}$. Note that an AP solution can be viewed as a collection of cycles visiting each vertex exactly once. The most efficient AP algorithm is the Hungarian algorithm, which is based on König-Egervary's theorem and has a complexity of $\mathcal{O}(n^3)$. In the AP-SAT algorithm we use the implementation of the Hungarian algorithm by Jonker and Volgenant [19,36].

An instance of DAP can be solved by applying an AP algorithm to the AP instance defined by the matrix $C = (c_{ij})_{1 \leq i,j \leq n}$ with

$$c_{ij} = \begin{cases} 0, & \text{if } (i,j) \in E,\ i \neq j \\ 1, & \text{if } (i,j) \notin E,\ i \neq j \\ 1, & \text{if } i = j \end{cases}$$

where we map an arc in the original graph to an arc of cost 0 in the new complete graph and the remaining cost is set to 1. If the AP algorithm returns a solution with cost 0 on the new complete graph, there is a DAP solution in the original graph, since every arc taken in the AP solution is an arc in the original graph. On the other hand, if it returns a solution of cost greater than 0, there is no DAP solution in the original graph, because at least one arc in the solution does not belong to the original graph.

The first step of the AP-SAT algorithm is this DAP algorithm. Then a HC of G, if one exists, is a solution to the DAP. We have to distinguish three cases at the end of this first step:

- If the AP solution cost is greater than 0, G does not have a HC, and the DHCP instance is solved with no solution.
- If the AP solution cost is 0 and the solution consists of one cycle, we have found a HC, and the DHCP instance is also solved.
- If the AP solution has cost 0 and the AP solution has more than one cycle, we cannot determine, based on the AP solution, if the given G is Hamiltonian such that we continue to the next steps of the AP-SAT algorithm.

2.2 Karp-Steele Patching

If the DAP solution does not provide a definitive answer to our problem, i.e., the case where the AP solution cost is 0 and the AP solution contains more than one

cycle, we continue to search for a HC in G. We first patch the subcycles in an attempt to form a HC, and we use *Karp-Steele patching* (*KSP*) for this purpose, which is an effective ATSP heuristic [11,12,21]. The operation of patching two cycles C_i and C_j in an AP solution is defined as follows: two fixed arcs $(v_i, w_i) \in C_i$ and $(v_j, w_j) \in C_j$ are first deleted and two arcs (v_i, w_j) and (v_j, w_i) joining the two cycles are added. The cost of patching C_i and C_j using (v_i, w_j) and (v_j, w_i) is equal to $\delta(C_i, C_j) = c(v_i, w_j) + c(v_j, w_i) - (c(v_i, w_i) + c(v_j, w_j))$, i.e., $\delta(C_i, C_j)$ is the difference between the total cost of the inserted arcs and the total costs of the deleted arcs. In each step we choose to patch the two cycles that have the largest number of vertices. For these two cycles, the two arcs are chosen in such a way that the patching cost is the minimum among all possible arc pairs. If we have $k \geq 2$ cycles we repeat this patching step $k - 1$ times to form one cycle at the end. We apply KSP to the AP instance, defined in Section 2.1. If the patching procedure provides a HC, the AP-SAT algorithm can be terminated. Otherwise, we continue to the next step.

2.3 Solving Variant APs

DAP may have multiple solutions, and some of the DAP solutions may be HCs. We can increase the chance of finding a HC if we apply the AP step multiple times, since the computational cost of the AP and the KSP algorithms is low. The key is to avoid finding the same DAP solution multiple times. To do this, we slightly alter some of the arc costs of the corresponding AP instance so as to find the other DAP solutions, enhanced by the KSP if needed, to increase the possibility of finding a HC for a given graph. In other words, we add a randomized component of solving multiple variant AP instances to boost the overall chance of finding a HC. Note that in the worst case when the DHCP instance is not Hamiltonian, this procedure will not be productive.

To create a variant AP instance different to that in Section 2.1, we generalize the AP instance as follows. Let $c_{i,j}$ be a non-negative cost value for arc $(i, j) \in E$. Let

$$M := n \cdot \max\{c_{i,j} \mid (i, j) \in E\} + 1$$

i.e., M is greater than n times the largest cost of all the arcs in the given graph G. We then set the remaining cost to M. The AP instance in Section 2.1 is a special case of the generalized version of the AP instance, where all costs $c_{i,j}$, for $(i, j) \in E$, are 0. It is critical to notice that all DAP solutions, including a HC, must have costs less than M. The key idea to create a variant AP instance is to "perturb" the costs of some of the arcs in G such that a new DAP solution, which is different from any of the ones that have been encountered, can be found. In order to possibly reach a new DAP solution, for each arc in the current DAP solution we decrease the probability of its appearance by increasing its cost by 1 and receive a variant AP instance to be solved. As before, if the solution contains a HC, the algorithm terminates; otherwise, the sub-cycles are patched using the KSP to possibly find a HC. We repeat this step multiple times, so that an arc,

which has appeared in many previous DAP solutions, will be very unlikely to appear in the next DAP solution, and an arc, which has never appeared in any previous DAP solution, will be more likely to appear in the next DAP solution. We observed in our experiments that solving variant AP instances n times seems to be a good choice.

2.4 Implicitly Enumerating All DAP Solutions Using SAT

All the AP and patching based steps discussed above may still miss a solution to a DHCP instance. We now consider to implicitly enumerate all DAP solutions with a hope of finding a solution to the DHCP. The idea is to systematically rule out all the DAP solutions that have been discovered so far during the search. To this end, we formulate a DAP as a Boolean satisfiability (SAT) problem and forbid a DAP solution by adding new constraints to the SAT model. This technique of adding new constraints with purpose to enumerate all SAT solutions has been introduced by Jin, Han, and Somenzi [16] for a general SAT problem. Notice that this cannot be easily done under the AP framework because such constraints cannot be properly added to the AP. Moreover, we can take advantage of the research effort that has been devoted to SAT, in particular, we can use an effective SAT solver called MiniSAT [7,35].

In the conjunctive normal form (CNF), a SAT instance over a set of Boolean variables is a conjunction of clauses, each of which is a disjunction of literals which are Boolean variables or their negations. A clause is satisfied, if one of its literals is True, and the instance is satisfied if all its clauses are satisfied. The SAT problem is to find a truth assignment of the variables to satisfy all clauses, if they are satisfiable, or to determine no such assignment exists. SAT was the first problem shown to be $\mathcal{NP}$-complete [20].

We now formulate the DAP in SAT. A solution to a DAP must obey the following restriction:

- For each vertex i, $i = 1, \ldots, n$, exactly one arc (i, j), $i \neq j$, exists in the DAP solution.
- For each vertex i, $i = 1, \ldots, n$, exactly one arc (j, i), $j \neq i$, exists in the DAP solution.

We first introduce an integer decision variable $x_{i,j}$ to arc $(i, j) \in E$ and represent the above constraints in the following integer linear program (ILP).

$$\begin{cases} \sum_{j=1,(i,j)\in E}^{n} x_{i,j} = 1 \text{ for } i = 1, \ldots, n \\ \sum_{i=1,(i,j)\in E}^{n} x_{i,j} = 1 \text{ for } j = 1, \ldots, n \end{cases} \tag{1}$$

where $x_{i,j} \in \{0, 1\}$ for $(i, j) \in E$. Note that we only have to use m variables, one variable for each arc in the graph, which can be substantially smaller than n^2 variables for sparse graphs. We represent the integer linear program (1) by a SAT model similar to [25], where we replace integer variables $\{x_{i,j}\}$ with Boolean variables $\{y_{i,j}\}$. To enforce the $2n$ restrictions in the SAT formulation, we need

to introduce constraints in clauses. One restriction in (1) means that exactly one of the n involved Boolean variables can be set to True and the rest must be False. To represent this, we introduce at most $2n^2$ auxiliary variables $z_1, z_2, \ldots, z_{2n^2}$, with n z's for one restriction. W.l.o.g., consider the first restriction, which has $z_1, z_2, \ldots, z_n$ associated. We use z_k to represent at least one of $y_{1,1}, y_{1,2}, \ldots, y_{1,k}$ is True. Precisely, the z variables are defined as follows.

- $z_1 = y_{1,1}$ or equivalently $(\neg y_{1,1} \vee z_1) \wedge (y_{1,1} \vee \neg z_1)$.
- $z_k = y_{1,k} \vee z_{k-1}$ or equivalently $(z_k \vee \neg y_{1,k}) \wedge (z_k \vee \neg z_{k-1}) \wedge (\neg z_k \vee y_{1,k} \vee z_{k-1})$ for $k = 2, 3, \ldots, n$.

In addition, we need to enforce that only one $y_{1,i}$, for $i = 1, 2, \ldots, n$, can be True. This means, if $y_{1,k}$ is True, none of the $y_{1,i}$, for $i < k$, can be True. This is formulated as

- $\neg z_{k-1} \vee \neg y_{1,k}$ for $k = 2, \ldots, n$.

Finally z_n must be True. The other restrictions in (1) are represented similarly.

The SAT based representation allows us to exclude a DAP solution, which is not a HC, previously found in the search. This can be done by introducing new clauses to explicitly forbidding all subcycles of the solution. Let such a subcycle be $(v_1, \ldots, v_k, v_1)$. We add the following clause

$$\neg y_{v_1,v_2} \vee \ldots \vee \neg y_{v_{k-1},v_k} \vee \neg y_{v_k,v_1} \tag{2}$$

to the current SAT instance. As a result, the updated SAT instance is not satisfiable, meaning that the corresponding DHCP instance is not Hamiltonian, or gives rise to a new DAP solution, as it does not allow the previous DAP solution.

In summary, after the AP- and patching-related steps failed to find a solution, the AP-SAT algorithm transforms the problem instance into a SAT instance. Then it collects all previous DAP solutions, each of which includes at least two subcycles, and excludes these subcycles for each of these DAP solutions by adding new clauses as described above. Then the resulting SAT model is solved using MiniSAT [7,35]. If the SAT model is not satisfiable, then the DHCP algorithm terminates with the result of G being not Hamiltonian. If the SAT model is satisfiable and the solution has only one cycle, the algorithm stops with a solution. If the SAT model is satisfiable, but the solution has more than one subcycle, new clauses are added to the SAT model to rule out this solution, and the algorithm repeats to solve the model. Since there is a finite number of DAP solutions, the algorithm terminates. In the worst case when the DAP solutions contain no HC, the SAT part of the algorithm will enumerate all these DAP solutions.

2.5 General Remarks

1. The AP-SAT algorithm consists of three main components, namely the AP step, the KSP step and the SAT step. It might be interesting to know which of these components is the most important one. For this, we have to distinguish

between completeness and efficacy of the algorithm. The only necessary step for the completeness is the SAT step of Section 2.4. This step without all previous steps leads also to a correct DHCP algorithm. On the other hand, in contrast to AP and KSP, SAT is $\mathcal{NP}$-complete. Thus the AP-SAT algorithm is more effective, if the AP and the KSP steps are called often and the SAT step is called only a few times. For example, if for an instance no DAP solution exists or an existing HC is found by the previous steps, the SAT part is not necessary. Indeed, our further experiments showed that the SAT step is not called for most of the test instances. Regarding the relative time needed by the AP and the KSP steps, we have to consider the density of problem instances. For an instance with a small number of edges, in most cases there is not only no HC solution, but also no DAP solution. In this case the algorithm finishes after the first AP step and does not need any KSP call. On the other hand, an instance with a large number of edges should require many AP steps, as many DAP solutions may exist which are not HCs, and thus probably the first HC solution has to be found by KSP. Also this expected behavior could be verified by experiments: the time for the KSP steps is rather small for instances with a small number of edges, and almost as large as the time for the AP steps for instances with a large number of edges.

2. The AP-SAT algorithm is also able to solve HCP as a special case of DHCP, but it is less effective for this case. The reason is that for the symmetric case, an edge is often present in a DAP solution in combination with its reverse edge, resulting in many small cycles of two vertices in the solution. Thus in general we have to enumerate a large number of DAPs solutions. In the worst case, no HC exists, but all these DAP solutions have to be enumerated, requiring a long running time.
3. We can easily transform the AP-SAT algorithm to enumerate all HCs in a directed graph. This is a well-known problem with many applications, e.g., the problem of finding all knight's tour on a chessboard [15,24]. For algorithms for this problem, see the already mentioned algorithm of Martello [26] and the algorithm of Frieze and Suen [10]. The transformation works as follows. If no or only one HC exists, the algorithm remains the same. Consider now the case that more than one HC exists. If the first HC has been found, the original AP-SAT algorithm terminates in this case. The transformed algorithm at this stage saves the first HC, and then continues to search for the next HC. Note that for the transformed algorithm, the SAT part is always called, if at least one HC exists. Furthermore – like the original AP-SAT algorithm – this transformed algorithm works also for the symmetric case, but is not effective for this case.

3 Experimental Results

We have implemented the AP-SAT algorithm and the DHCP algorithm of Martello [26] in C++ and compared them to an algorithm based on the award-winning Concorde TSP program implemented by Applegate, Bixby, Chvátal and

Cook [2,34]. For the algorithm of Martello we have implemented a version which terminates whenever a HC, if one exists, is found. As subroutines for the AP-SAT algorithm we used the AP solver implemented by Jonker and Volgenant [19,36] and the *MiniSat* SAT solver implemented by Eén and Sörensson [7,35]. For Concorde, a DHCP instance was first transformed to an asymmetric TSP instance by the transformation in Section 2.1 and then to a symmetric TSP instance by the 2-point reduction method [18]. In our experiments, we also tried the 3-point reduction method [20], but got average running times worse than that using the 2-point reduction. After the 2-point reduction, Concorde started with the worst possible solution value as the initial upper bound and was terminated as soon as its lower bound made a HC impossible. All our experiments were carried out on a PC with an Athlon 1900MP CPU with 2GB of memory.

In our experiments we used random asymmetric instances $G_{n,m}$ as test instances, and parameters $n = 100, 200, 400, 800, 1600$ and $m = \lceil c \cdot n \cdot (\log n + \log\log n) \rfloor$ with $c = 0.5, 0.6, \ldots, 1.90, 2.00$, where for $c = 1$ a phase transition is expected. For each n we generated 50 random instances and measured the average time over these instances. The results are summarized in Figure 2, where the x-axis describes the parameter c and the y-axis the average CPU time in seconds. Furthermore, we tested real world and random instances from the DIMACS challenge [17,31] which contains 10 asymmetric problem generators called *amat, coin, crane, disk, rect, rtilt, shop, stilt, super, tmat*. Using each of these generators we generated 24 instances, 10 with 100 vertices, 10 with 316 vertices, three with 1000 vertices, and one with 3162 vertices. To transform asymmetric TSP instances back to DHCP instances, it seems to be reasonable to only keep the arcs of small weights while ignoring the ones with large weights. In other words, to generate a DHCP instance we chose m smallest arcs in the corresponding asymmetric TSP instance. It is interesting to note that the most difficult DIMACS instances appear when the degree parameter c is around 2, which is the value we used in our experiments. Again, for each problem size we measured the average time over these instances. The results for the DIMACS instances are summarized in Figures 1 and 3.

Figure 2 shows that the AP-SAT algorithm and Concorde are more stable than the Martello algorithm: Concorde failed to solve 16 single instances within a given maximal CPU time of 1 hour, whereas the AP-SAT algorithm failed only on 7 single instances (6 of 7 on the *stilt* types from DIMACS). The Martello algorithm was unable to solve most instances with 800 or more vertices because of too large memory requirements. Nevertheless, the Martello algorithm outperformed Concorde on smaller and easier instances, indicating that the former has a worse asymptotic running time. Furthermore, we observed that the AP-SAT algorithm is superior to the other two algorithms; the former outperformed the latter on all problem instances except the *stilt* types from DIMACS, where Concorde is competitive.

The efficacy of the AP-SAT algorithm may be due to the following reasons. Instances with no HC are most likely to have no DAP solution either, and therefore the algorithm terminates after the first AP call. On the other hand, instances

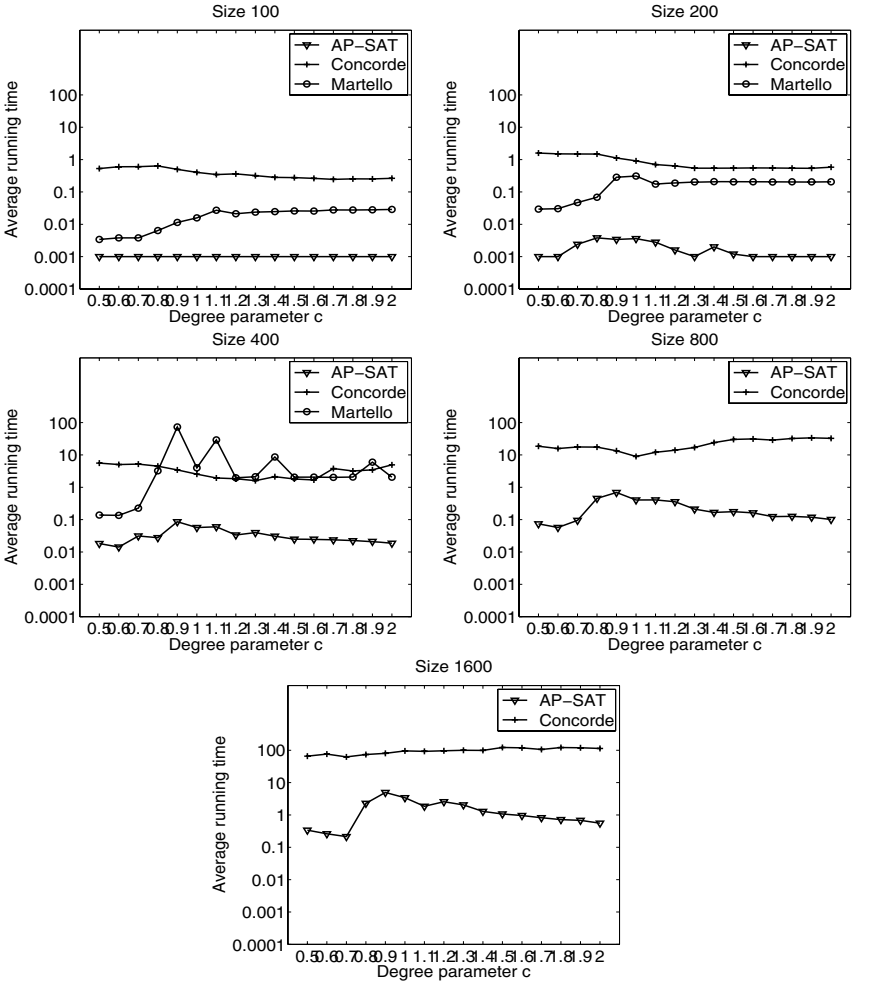


Fig. 1. DIMACS instances, part 1

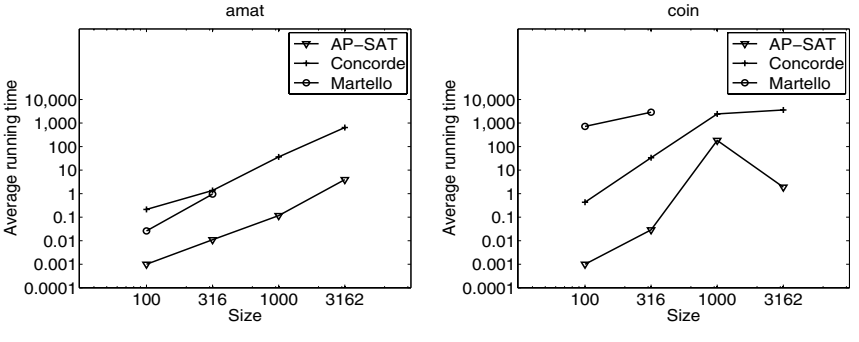


Fig. 2. Random instances

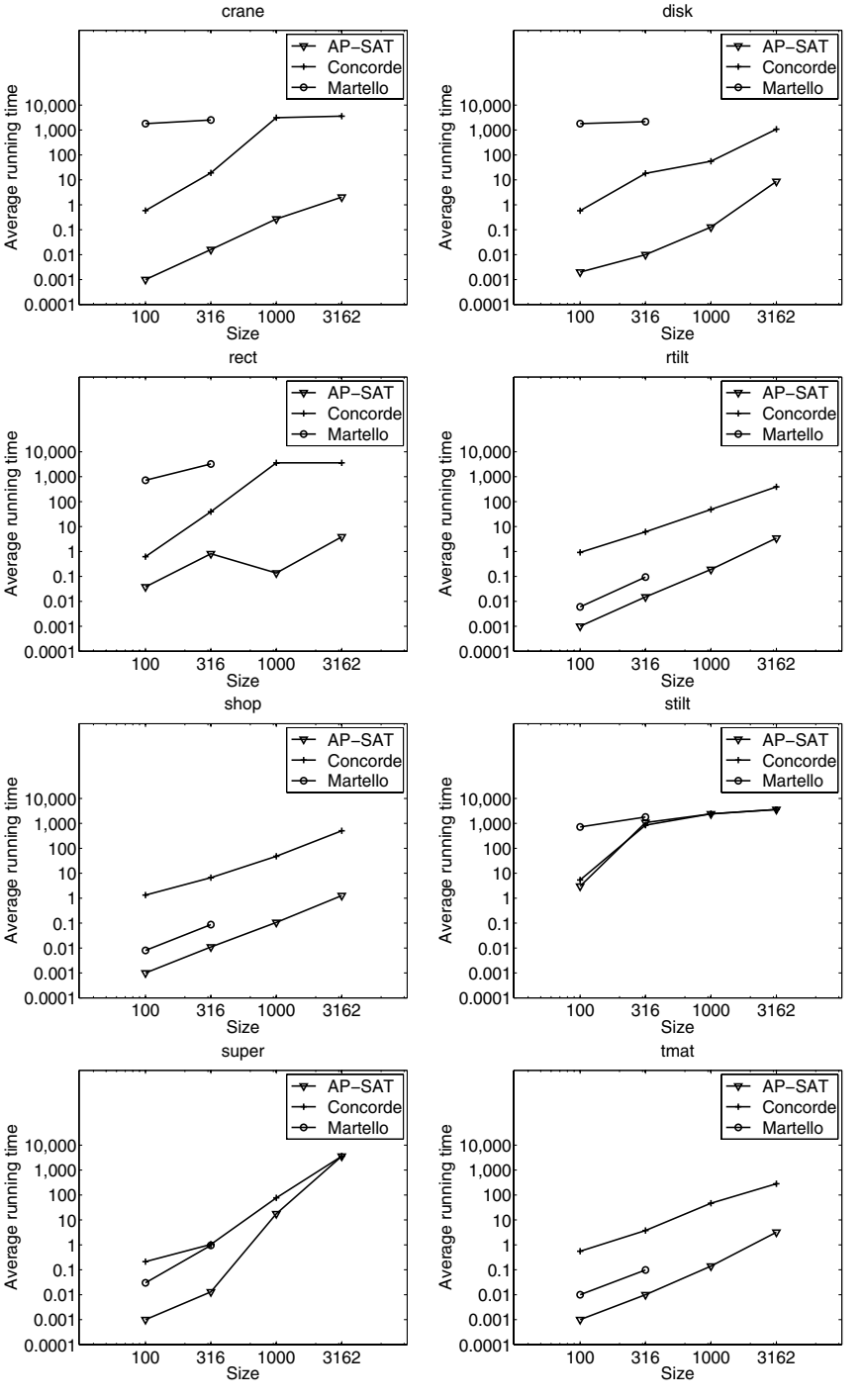


Fig. 3. DIMACS instances, part 2

with a HC are likely to have many HCs, one of which can be found very soon by the AP or KSP steps. The only difficult case is when there are many DAP solutions but none or a very few of them are HCs. Indeed, such cases occur for a few unsolved instances and instances requiring long computation. In such cases, often the SAT part does not terminate in a reasonable amount of time.

Acknowledgement

This research was supported in part by NSF grants IIS-0535257 and DBI-0743797 to Weixiong Zhang.

References

1. Angluin, D., Valiant, L.G.: Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings. J. Comput. System. Sci. 18(2), 155–193 (1979)
2. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem. A Computational Study. Princeton University Press, Princeton (2006)
3. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications, ch. 5. Springer, London (2008), http://www.cs.rhul.ac.uk/books/dbook/
4. Bollobás, B., Fenner, T.I., Frieze, A.M.: An Algorithm for Finding Hamiltonian Paths and Cycles in Random Graphs. Combinatorica 7(4), 327–341 (1987)
5. Christofides, N.: Graph Theory – An Algorithmic Approach. Academic Press, New York (1975)
6. Chvátal, V.: Hamiltonian Cycles. In: Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.) The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization, ch. 11. John Wiley & Sons, Chichester (1985)
7. Eén, N., Sörensson, N.: An Extensible SAT-Solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
8. Frieze, A.M.: Finding Hamiltonian Cycles in Sparse Random Graphs. J. Combin. Theory Ser. B 44, 230–250 (1988)
9. Frieze, A.M.: An Algorithm for Finding Hamilton Cycles in Random Directed Graphs. J. Algorithms 9, 181–204 (1988)
10. Frieze, A.M., Suen, S.: Counting Hamilton cycles in random directed graphs. Random Structures Algorithms 9, 235–242 (1992)
11. Glover, F., Gutin, G., Yeo, A., Zverovich, A.: Construction Heuristics for the Asymmetric TSP. European J. Oper. Res. 129, 555–568 (2001)
12. Goldengorin, B., Jäger, G., Molitor, P.: Tolerance Based Contract-or-Patch Heuristic for the Asymmetric TSP. In: Erlebach, T. (ed.) CAAN 2006. LNCS, vol. 4235, pp. 86–97. Springer, Heidelberg (2006)
13. Gould, R.J.: Updating the Hamiltonian Problem – a Survey. J. Graph Theory 15(2), 121–157 (1991)
14. Grebinski, V., Kucherov, G.: Reconstructing a Hamiltonian Circuit by Querying the Graph: Application to DNA Physical Mapping. IR 96-R-123, Centre de Recherche en Informatique de Nancy (1996)
15. Henderson, R., Apodaca, E.: A Knight of Egodeth: Zen Raptured Quietude. Book Surge Publishing (2008)
16. Jin, H., Han, H., Somenzi, F.: Efficient Conflict Analysis for Finding All Satisfying Assignments of a Boolean Circuit. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 287–300. Springer, Heidelberg (2005)

17. Johnson, D.S., Gutin, G., McGeoch, L.A., Yeo, A., Zhang, W., Zverovich, A.: Experimental Analysis of Heuristics for the ATSP. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and Its Variations, ch. 10. Kluwer, Dordrecht (2002)
18. Jonker, R., Volgenant, A.: Transforming Asymmetric into Symmetric Traveling Salesman Problems. Oper. Res. Lett. 2(4), 161–163 (1983)
19. Jonker, R., Volgenant, A.: A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems. Computing 38, 325–340 (1987)
20. Karp, R.M.: Reducibility Among Combinatorial Problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum, New York (1972)
21. Karp, R.M., Steele, J.M.: Probabilistic Analysis of Heuristics. In: Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.) The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization, ch. 6. John Wiley & Sons, Chicester (1985)
22. Kelly, L.: Hamilton Cycles in Directed Graphs. PhD Thesis, University of Birmingham, United Kingdom (2007)
23. Komlós, M., Szemerédi, E.: Limit Distribution for the Existence of a Hamiltonian Cycle in a Random Graph. Discrete Math. 43, 55–63 (1983)
24. Kyek, O., Parberry, I., Wegener, I.: Bounds on the Number of Knight's Tours. Discrete Appl. Math. 74(2), 171–181 (1997)
25. Lynce, I., Marques-Silva, J.: Efficient Haplotype Inference with Boolean Satisfiability. In: Proc. 21st National Conference on Artificial Intelligence (AAAI). AAAI Press, Menlo Park (2006)
26. Martello, S.: An Enumerative Algorithm for Finding Hamiltonian Circuits in a Directed Graph. ACM Trans. Math. Software 9(1), 131–138 (1983)
27. McDiarmid, C.J.H.: Cluster Percolation and Random Graphs. Math. Program. Stud. 13, 17–25 (1980)
28. Pósa, L.: Hamiltonian Circuits in Random Graphs. Discrete Math. 14, 359–364 (1976)
29. Stojaković, M., Szabó, T.: Positional Games on Random Graphs. Random Structures Algorithms 26(1-2), 204–223 (2005)
30. Vandegriend, B.: Finding Hamiltonian Cycles: Algorithms, Graphs and Performance. Master Thesis, University of Alberta, Canada (1998)
31. 8th DIMACS Implementation Challenge: The Traveling Salesman Problem, http://www.research.att.com/~dsj/chtsp/
32. The Hamiltonian Page by Gutin, G., Moscato, P.: http://www.ing.unlp.edu.ar/cetad/mos/Hamilton.html
33. The Stony Brook Algorithm Repository by Skiena, S.: http://www.cs.sunysb.edu/~{}algorith/files/hamiltonian-cycle.shtml
34. Source code of [2](Concorde), http://www.tsp.gatech.edu/concorde.html
35. Source code of [7] (MiniSat), http://minisat.se
36. Source code of [19], http://www.magiclogic.com/assignment.html

Balancing Bounded Treewidth Circuits*

Maurice Jansen and Jayalal Sarma M.N.

Institute for Theoretical Computer Science,
Tsinghua University, Beijing, China
maurice.julien.jansen@gmail.com, jayalal@tsinghua.edu.cn

Abstract. We use algorithmic tools for graphs of small treewidth to address questions in complexity theory. For both arithmetic and Boolean circuits, we show that any circuit of size $n^{O(1)}$ and treewidth $O(\log^i n)$ can be simulated by a circuit of width $O(\log^{i+1} n)$ and size n^c, where $c = O(1)$, if $i = 0$, and $c = O(\log \log n)$ otherwise. For our main construction, we prove that multiplicatively disjoint arithmetic circuits of size $n^{O(1)}$ and treewidth k can be simulated by bounded fan-in arithmetic formulas of depth $O(k^2 \log n)$. From this we derive an analogous statement for syntactically multilinear arithmetic circuits, which strengthens the central theorem of [14]. As another application, we derive that constant width arithmetic circuits of size $n^{O(1)}$ can be balanced to depth $O(\log n)$, provided certain restrictions are made on the use of iterated multiplication. Also from our main construction, we derive that Boolean bounded fan-in circuits of size $n^{O(1)}$ and treewidth k can be simulated by bounded fan-in formulas of depth $O(k^2 \log n)$. This strengthens in the non-uniform setting the known inclusion that $\mathrm{SC}^0 \subseteq \mathrm{NC}^1$. Finally, we apply our construction to show that Reachability and Circuit Value Problem for some treewidth restricted cases can be solved in LogDCFL.

1 Introduction

It is well-known that many hard graph theoretical problems become tractable when restricted to graphs of bounded treewidth[1]. If a graph with n nodes has bounded treewidth, there always exists a balanced tree decomposition of depth $O(\log n)$. This yields NC-algorithms for many problems, which are known to be NP-complete in general [6].

Consider the following question. Suppose one is given a circuit (Boolean or arithmetic) of size s and bounded fan-in, for which the underlying graph has bounded treewidth. Does this imply, as intuition might suggest, that there must exist an equivalent bounded fan-in circuit of size $poly(s)$ and depth $O(\log s)$? We show that in the Boolean case the situation is as expected, which yields the following theorem:

* This work was supported in part by the National Natural Science Foundation of China Grant 60553001, and the National Basic Research Program of China Grant 2007CB807900, 2007CB807901.

[1] For a definition see Section 2.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 228–239, 2010.

Theorem 1. *The class of languages accepted by non-uniform constant fan-in circuits of polynomial size and bounded treewidth equals non-uniform* NC^1.

Due to a celebrated result of Barrington [4], it is known that NC^1 can be simulated by constant width branching programs, which are skew circuits of constant width. A constant width circuit can be evaluated using $O(1)$ memory, and hence $\mathrm{SC}^0 = \mathrm{NC}^1$ (SC^0 is the class of Boolean functions computable by constant width circuits of *poly* size, c.f. [13]). Theorem 1 strengthens this statement in the non-uniform setting.

For arithmetic circuits, the short answer is that the equivalent circuit need not exist: a depth $O(\log s)$ circuit of bounded fan-in computes a polynomial of degree $s^{O(1)}$, but using repeated multiplication a bounded treewidth circuit of size s can easily compute a polynomial of degree 2^s. We rephrase the question to avoid this triviality.

The class of families of polynomials $\{p_n\}_{n\geq 1}$ of polynomial degree with polynomial size arithmetic circuits is known as VP (See e.g. [8]). In this paper, we let $\mathrm{VP}[tw = O(\log^i n)]$ stand for the class corresponding to polynomial size circuits of treewidth $O(\log^i n)$. Let VNC^1 denote the class corresponding to bounded fan-in arithmetic circuits of depth $O(\log n)$, which due to Brent's result [7] corresponds to poly-size arithmetic formulas. Our question becomes the following: is $\mathrm{VP}[tw = O(1)] \subseteq \mathrm{VNC}^1$?

One reason for considering this question, is that in case of an affirmative (and effective) answer it could be a useful tool for circuit building. Namely, one could during the design stage focus on using any of the well-known classes of bounded treewidth, and be guaranteed a nicely balanced formula can be obtained afterwards. Another reason, more on the complexity theoretic side, is that a flexible parameter like treewidth contributes to obtaining a more refined structural understanding of the difference between formulas and circuits, since it provides hierarchies of classes which bridge the two notions. Regarding this, it is a major open problem whether VNC_1 and VP are distinct. We only know of a separation in the multilinear world, due to the result by Raz [17].

Considering the notion of restricted treewidth circuits will also shed some light on issues regarding bounded width circuits. Arithmetic circuits of bounded treewidth provide a common generalization of formulas and of circuits of bounded width. One has the hierarchy of classes $\{\mathrm{VSC}^i\}_{i\geq 0}$, where VSC^i corresponds to arithmetic circuits of width $O(\log^i n)$ and size $n^{O(1)}$, for $i \geq 0$. This hierarchy is known to be sandwiched in between VNC_1 and VP. We prove the following (and the Boolean analogue):

Theorem 2

1. $\mathrm{VSC}^0 \subseteq \mathrm{VP}[tw = O(1)] \subseteq \mathrm{VSC}^1$.
2. $\mathrm{VSC}^i \subseteq \mathrm{VP}[tw = O(\log^i n)] \subseteq \mathrm{VSC}^{i+1}[size = n^{O(\log\log n)}]$, *for any* $i \geq 1$.

Arithmetic circuit width is a fundamental notion, but it is still ill-understood in its relation to other resources. We currently do not know of any "natural" problems characterized by bounded width arithmetic circuit classes. However,

as of recently, the notion has gained renewed attention from several researchers. It more or less embodies a notion of space in the arithmetic setting (See [15]). Mahajan and Rao [14] study the class $\mathrm{VSC}^0[deg = n^{O(1)}]$, which is obtained from VSC^0 by requiring formal degrees of circuits to be $n^{O(1)}$. Arvind, Joglekar, and Srinivasan [2] give lower bounds for *monotone* arithmetic circuits of constant width.

To make progress on the basic question whether $\mathrm{VP}[tw = O(1)] \subseteq \mathrm{VNC}^1$, we show that *multiplicatively disjoint*[2] circuits of size $n^{O(1)}$ and bounded treewidth can be simulated by bounded fan-in formulas of depth $O(\log n)$. In our notation this is stated as follows:

Theorem 3. $\text{md-VP}[tw = O(1)] = \mathrm{VNC}^1$.

Notice that without the treewidth restriction multiplicatively disjoint circuits of size $n^{O(1)}$ are known to compute all of VP [16]. From the above result, we derive the analogous statement for *syntactically multilinear* circuits. The resulting formulas will be syntactically multilinear (denoted with the prefix sm-) as well. This implies the lower bounds by Raz [17] hold all the way up to syntactically multilinear bounded treewidth circuits. We prove

Theorem 4. $\text{sm-VP}[tw = O(1)] = \text{sm-VNC}^1$.

Theorem 4 strengthens the main result of Mahajan and Rao [14], which states that *poly* size syntactically multilinear circuits of constant width can be simulated by *poly* size circuits of *log* depth (but it was explicitly left open whether the latter could be ensured to be syntactically multilinear). Considering the more general notion of treewidth in a way simplifies the proof due to the help of well-established algorithmic tools. We also remark that Theorem 3 strengthens Theorem 4 in [10], which states that $\text{md-VSC}^0 = \mathrm{VNC}^1$.

In [14] the fundamental question is raised whether $\mathrm{VSC}^0[deg = n^{O(1)}] \subseteq \mathrm{VNC}^1$. As was mentioned, they show this holds under the restriction of syntactic multilinearity. To make progress, we demonstrate a different restriction under which an efficient simulation by arithmetic $O(\log n)$ depth formulas is achievable. We apply Theorem 3 to give the following result for circuits with *bounded iterated multiplication chains* (For a definition see Section 4):

Theorem 5. *Constant width arithmetic circuits of size $n^{O(1)}$ with constant bounded iterated multiplication chains can be simulated by fan-in two arithmetic formulas of depth $O(\log n)$.*

As two additional applications of the above results, we consider the Circuit Value Problem (CVP) and the Reachability problem. Given the encoding of a Boolean circuit C and an input x the Circuit Value Problem is to test if $C(x) = 1$ or not. The general version of this problem is known to be P-complete. Several variants of this has been studied (See [9,3,12] and the references therein).

Given a (directed or undirected) graph $G = (V, E)$ and $s, t \in V$, reachability asks to test if t is reachable from s in G. reachability captures space

[2] We indicate this with the prefix md-. See Section 2 for a definition.

bounded computation in a natural way. For directed graphs it is complete for NL [11,19]. The case of undirected graphs was settled recently by Reingold [18] by giving a log-space algorithm for the problem. This shows the problem is complete for L. There has been extensive research aimed at settling the complexity of testing reachability on restricted graphs (see [1] and references therein).

LogCFL and LogDCFL are the classes of languages that are logspace many-one reducible to non-deterministic and deterministic context-free languages, respectively. LogDCFL can be also characterized as the class of languages that can be recognized by a logspace Turing machine that is also provided with a stack, which runs in polynomial time. It follows by definition that $\mathrm{L} \subseteq \mathrm{LogDCFL} \subseteq \mathrm{LogCFL}$ and $\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{LogCFL}$. However, it is unknown how NL and LogDCFL can be compared. In essense, this asks for a trade-off, trading non-determinism with stack access. Given that directed reachability is an NL-complete problem, giving a LogDCFL upper bound achieves such a trade-off for a restricted class of NL-computations.

We prove the following theorem for the CVP and obtain a corollary for Reachability.

Theorem 6. CVP *for bounded treewidth and bounded fan-in circuits when the input also contains the tree decomposition, is in* LogDCFL.

Corollary 1. Reachability *for directed acyclic graphs of bounded treewidth and in-degree is in* LogDCFL*, provided the tree decomposition is given at the input.*

2 Preliminaries

We briefly recall basic circuit definitions. A *Boolean circuit* is a directed acyclic graph, with labels $\{0, 1, x_1, \ldots, x_n, \wedge, \vee, \neg\}$ on its nodes. Nodes with label from $\{0, 1, x_1, \ldots, x_n\}$ are called *input gates*, and designated nodes of zero out-degree are called the *output gates*. The *fan-in* of a gate is its in-degree. Formulas are circuits for which the out-degree of each gate is at most one. For size of a circuit we count the number of non-input gates. The depth is measured as the length of a longest directed path. Fan-in is assumed to be bounded. As in [14,10], when we speak about the width of a circuit, we assume the circuit is layered, and it is taken to be the maximum number of nodes on a layer. Let us emphasize that we allow input gates (constant or variable labeled) to appear at all layers. The class NC^1 is the class of boolean functions on n bits which can be computed by boolean circuits of depth $O(\log n)$ and size $n^{O(1)}$. SC^i denotes the class of functions computed by polynomial size circuits of width $O(\log^i n)$.

For *arithmetic circuits* over a ring R, nodes are labeled by ring constants, formal variables from a set X, and $\{+, \times\}$. We assume that the fan-in is bounded by two. The output of an arithmetic circuit is a polynomial in the ring $R[X]$, defined in the obvious way. The size of a circuit is taken to be the number of $\{+, \times\}$-gates. For a circuit Φ with designated output gate f, the polynomial computed by the output gate is denoted with $\lceil \Phi \rceil$. We denote the set of variables

used in Φ by $Var(\Phi)$. Similarly we use $Var(p)$, if p is a polynomial. Note that $Var(\lceil \Phi \rceil) \subseteq Var(\Phi)$. We call a polynomial f *multilinear* in some subset of the variables S, if the individual degree is at most one in f, for each variable in S (even if f has a constant term). An arithmetic circuit is called *syntactically multilinear* if for each multiplication gate the subcircuits originated at its inputs carry disjoint sets of variables. For a *multiplicatively disjoint* circuit, for every gate $f = g \times h$, the sub-circuits rooted at g and h are disjoint (as graphs). The *formal degree* of a circuit is defined inductively by taking variable and constant labeled gates to be of degree one. For addition gates one takes the maximum of the degrees of its inputs. For multiplication gates one takes the sum of the degrees. The degree of the circuit is taken to be the maximum degree of a gate.

For a *p-family of polynomials* $\{f_m\}_{m\geq 1}$, we have $f_m \in R[x_1, x_2, \ldots, x_{p(m)}]$, and $\deg(f_m) \leq q(m)$, for some polynomials p and q. Arithmetic circuit classes contain p-families. VP and VP_e are the classes of p-families computable by arithmetic circuits and formulas, respectively, of size $n^{O(1)}$ (See e.g. [8]). For $i \geq 0$, VSC^i is the class of all p-families computable by arithmetic circuits of width $O(\log^i n)$ and size $n^{O(1)}$. In [14] the class a-sSC^i is considered, which corresponds to width $O(\log^i n)$ circuits of size and formal degree $n^{O(1)}$. We will denote this class by $\mathrm{VSC}^i[deg = n^{O(1)}]$. The class VNC^i is the set of all p-families computable by arithmetic circuits of depth $O(\log^i n)$ and size $n^{O(1)}$.

Next we define various graph parameters. The *width* of a layered graph is the maximum number of vertices in any particular layer. A *tree decomposition* of a graph $G = (V, E)$ is given by a tuple $(T, (X_d)_{d\in V[T]})$, where T is a tree, each X_d is a subset of V called a *bag*, satisfying 1) $\bigcup_{d\in V[T]} X_d = V$, 2) For each edge $(u, v) \in E$, there exists a tree node d with $\{u, v\} \subseteq X_d$, and 3) For each vertex $u \in V$, the set of tree nodes $\{d : u \in X_d\}$ forms a connected subtree of T. Equivalently, for any three vertices $t_1, t_2, t_3 \in V[T]$ such that t_2 lies in the path from t_1 to t_3, it holds that $X_{t_1} \cap X_{t_3} \subseteq X_{t_2}$.

The *width* of the tree decomposition is defined as $\max_d |X_d| - 1$. The *treewidth* $\mathrm{tw}(G)$ of a graph G is the minimum width of a tree decomposition of G. For a rooted tree T, let $X_{\leq t} = \cup_{u\in S_t} X_u$, with $S_t = \{u : u = t \text{ or } t \text{ is an ancestor of } u\}$.

Lemma 1. *(Theorem 4.3 in [6]) Let $G = (V, E)$ be a graph with $|V| = n$ and treewidth at most k. Then G has a tree decomposition $(T, (X_d)_{d\in V[T]})$ of width $3k + 2$ such that T is a binary tree of depth at most $2\lceil \log_{\frac{5}{4}} n \rceil$.*

The following proposition is left as an easy exercise:

Proposition 1. *A leveled graph G of width k has treewidth at most $2k - 1$.*

3 Arithmetic Circuits of Bounded Treewidth

The treewidth of a circuit with underlying graph G is defined to be $\mathrm{tw}(G)$. Note that a circuit has treewidth 1 if and only if it is a formula. We introduce the class $\mathrm{VP}[tw = O(\log^i n)]$ as the class of p-families of polynomials $\{f_n\}_{n\geq 1}$ that can be computed by fan-in two arithmetic circuits of size $n^{O(1)}$ and treewidth

$\mathrm{O}(\log^i n)$. As Theorem 2 states, these classes interleave (roughly) with the VSC^i classes. We postpone the proof of Theorem 2 as it uses developments of our main construction.

Theorem 7. *For any multiplicatively disjoint arithmetic circuit Φ of size s and treewidth k, there exists an equivalent formula Γ of size at most $s^{O(k^2)}$.*

Proof. Let Φ be a multiplicatively disjoint circuit of size s, and let $(T, (X_t)_{t\in V[T]})$ be a tree decomposition of Φ of width k. By Lemma 1, we can assume that T is a rooted binary tree of depth $d = O(\log s)$. We first preprocess T and Φ using Proposition 2 (Proof will appear in full version).

Proposition 2. *For every circuit Φ of size s, that has a tree decomposition $(T, (X_t)_{t\in V[T]})$ of width k and depth d, there exists a circuit Φ' of size at most $2s$, for which $\lceil\Phi\rceil = \lceil\Phi'\rceil$, with tree decomposition $(T', (X'_t)_{t\in V[T']})$ of width at most $k' = 3k+2$ and depth at most d, so that for any $t \in T'$, for any non-input gate $g \in X'_t$ with inputs g_1 and g_2, either both $g_1, g_2 \in X'_t$ or both $g_1, g_2 \notin X'_t$. In the latter case it holds that $g_1 \notin X'_{\leq t}$ iff $g_2 \notin X'_{\leq t}$.*

We assume wlog. that Φ and $(T, (X_t)_{t\in V[T]})$ satisfy the conditions of Proposition 2, as the increase in k and s due to preprocessing does not affect the bound we are aiming for. For any tree node $t \in T$ and $f \in X_t$, we define a circuit Φ_t, which is obtained from the subgraph $\Phi[X_{\leq t}]$, by turning all $g \in X_t$ that take both inputs from gates not in $X_{\leq t}$ into input gates with label z_g. For any $f \in X_t$, let $\Phi_{t,f}$ be the subcircuit of Φ_t rooted at gate f. At most $k+1$ new z-variables will be used at the tree node t. Crucially, observe that, since Φ is multiplicatively disjoint, any gate in $\Phi_{t,f}$ computes a polynomial that is multilinear in $\overline{z}$.

We will process the tree decomposition going bottom up. At a node t, we want to compute for each $f \in X_t$ a formula $\Gamma_{t,f}$ equivalent to $\Phi_{t,f}$. Wlog. we assume that the output gate of Φ is contained in X_r, for the root r of T. Hence, when done, we have a formula equivalent to Φ. In order to keep the size of the computed formulas properly bounded, we require a constant bound on the number of appearances of a z-variable in $\Gamma_{t,f}$. We achieve this by brute-force with Proposition 3, at the cost of blowing up the size by a factor of 2^{k+1}. To verify its correctness, observe that the lhs. and rhs. are multilinear polynomial in $F[\overline{x}][z_1, z_2, \ldots, z_{k+1}]$ taking identical values on $\{0,1\}^{k+1}$, and hence must be identical.

Proposition 3. *For any $f(\overline{x}, z_1, z_2, \ldots, z_{k+1})$ that is multilinear in $\overline{z}$, we have that $f = \sum_{b\in\{0,1\}^{k+1}} \left(\prod_{i\in[k+1]} (1-z_i)^{1-b_i} z_i^{b_i}\right) f(\overline{x}, b_1, b_2, \ldots, b_{k+1})$.*

The recursive procedure for computing the desired formula equivalent to $\Phi_{t,f}$ is given by Algorithm 1. Formally, for any $t \in T$, and $f \in X_t$, let $\Gamma_{t,f}$ be the formula output by the procedure call $Traceback(t, f)$. The following lemma proves its correctness:

Lemma 2. *For any $t \in T$, and any $f \in X_t$, $\lceil\Gamma_{t,f}\rceil = \lceil\Phi_{t,f}\rceil$.*

Proof. The proof will proceed by structural induction both on T and Φ. The statement can be easily verified for the two base cases: if t is a leaf of T, or f is an input gate in Φ. For the induction step, suppose t has children t_0 and t_1, and say $f = f_0 \circ f_1$, with $\circ \in \{+, \times\}$. We ignore line 17 of the procedure $Traceback$, since it does not modify the output of the computed formula.

In case both $f_0, f_1 \in X_t$, by induction hypothesis, $\lceil \Gamma_{t,f_0} \rceil = \lceil \Phi_{t,f_0} \rceil$ and $\lceil \Gamma_{t,f_1} \rceil = \lceil \Phi_{t,f_1} \rceil$. Observe that in this case $Traceback(t, f)$ returns $\Gamma_{t,f_0} \circ \Gamma_{t,f_1}$, so $\lceil \Gamma_{t,f} \rceil = \lceil \Gamma_{t,f_0} \circ \Gamma_{t,f_1} \rceil = \lceil \Gamma_{t,f_0} \rceil \circ \lceil \Gamma_{t,f_1} \rceil = \lceil \Phi_{t,f_0} \rceil \circ \lceil \Phi_{t,f_1} \rceil = \lceil \Phi_{t,f} \rceil$.

Now assume not both $f_0, f_1 \in X_t$. By Proposition 2, this means $f_0 \notin X_t$ and $f_1 \notin X_t$. Furthermore, we either have $f_0, f_1 \in X_{\leq t}$, or $\{f_0, f_1\} \cap X_{\leq t} = \emptyset$. In the latter case, $\lceil \Phi_{t,f} \rceil = z_f$, which is exactly what is returned by $Traceback(t, f)$. In the former case, say $f_0 \in X_{\leq t_{i_1}}$ and $f_1 \in X_{\leq t_{i_2}}$, for $i_1, i_2 \in \{0, 1\}$. Observe that by the tree decomposition properties $f \in X_{t_{i_1}}$, which makes the call of $Traceback(t_{i_1}, f)$ on line 11 valid. Note that $f_0 \notin X_{\leq t_{i_2}}$ and $f_1 \notin X_{\leq t_{i_1}}$, if $i_1 \neq i_2$. Hence, by the tree decomposition properties, if $i_1 \neq i_2$, there would exist a node t' with t_1 as ancestor such that $f, f_0 \in X_{t'}$, but $f_1 \notin X_{t'}$. Due to Proposition 2 this case does not arise.

The algorithm first computes $\Gamma = Traceback(t_{i_1}, f)$. By the induction hypothesis $\lceil \Gamma \rceil = \lceil \Phi_{t_{i_1},f} \rceil$. In $\Phi_{t_{i_1},f}$, whenever a gate g takes an input from a gate not in $X_{\leq t_{i_1}}$, i.e. by Proposition 2 this means both its inputs are not in $X_{\leq t_{i_1}}$, it appears as input node with label z_g. However, for the circuit $\Phi_{t,f}$ node g roots $\Phi_{t,g}$. Observe that this means that substituting $\lceil \Phi_{t,g} \rceil$ for each $z_g \in Var(\lceil \Phi_{t_{i_1},f} \rceil)$ in $\lceil \Phi_{t_{i_1},f} \rceil$ yields $\lceil \Phi_{t,f} \rceil$. Observe that the tree decomposition properties give us that $g \in X_t$, whenever we make the call on line 13 to compute Γ', and hence that this call is valid. By the induction hypothesis, $\lceil \Gamma' \rceil = \lceil \Phi_{t,g} \rceil$. Hence replacing, for all $z_g \in Var(\lceil \Gamma \rceil)$, each gate in Γ labeled with z_g by the formula Γ' gives a new formula Γ satisfying $\lceil \Gamma \rceil = \lceil \Phi_{t,f} \rceil$. □

We must bound the size of the formula $\Gamma_{t,f}$. The proof of the following lemma will appear in the full version of the paper.

Lemma 3. *Let $t \in T$ be a node at height h, then for any $f \in X_t$, $\Gamma_{t,f}$ has at most $\alpha^h 2^{k+1}$ many gates, where $\alpha = 2^{3k^2+9k+6}$.*

Since T has depth $O(\log s)$, we conclude the final formulas given at the root of T will be of size $s^{O(k^2)}$. □

The proof of Theorem 3 is now clear. Trivially $\text{VP}_e \subseteq \text{md-VP}[tw = O(1)]$. The converse follows from Theorem 7. Now use the fact that $\text{VP}_e = \text{VNC}^1$ [7].

3.1 Proof of Theorem 4

Observe that $\text{sm-VNC}^1 \subseteq \text{sm-VP}_e \subseteq \text{sm-VP}[tw = O(1)]$. For the other direction, let Φ be a syntactically multilinear circuit of treewidth k. We first modify it so that any gate g computing a field constant α is replaced by an input gate g' labeled with α. This can be done by removing edges fanning into g and

Algorithm 1. Recursive procedure for computing $\Gamma_{t,f}$

procedure $Traceback(t \in T,\ f \in X_t)$
if t is a leaf or f is an input gate in Φ **then**
  return a formula equivalent to $\Phi_{t,f}$ of size at most 2^{k+1} computed by 'brute force'.
else
  let t_0 and t_1 be the children of t in T, and say $f = f_0 \circ f_1$, with $\circ \in \{+, \times\}$.
  if both f_0 and f_1 are in X_t **then**
    let $\Gamma = Traceback(t, f_0) \circ Traceback(t, f_1)$.
  else
    // Neither f_0 nor f_1 is in X_t, by pre-processing.
    If f_0 and f_1 are not in $X_{\leq t}$ **return** a single node with label z_f. Otherwise, say $f_0 \in X_{\leq t_{i_1}}$ and $f_1 \in X_{\leq t_{i_2}}$, for $i_1, i_2 \in \{0, 1\}$.
    $\Gamma = Traceback(t_{i_1}, f)$.
    for all $z_g \in Var(\lceil \Gamma \rceil)$ **do**
      let $\Gamma' = Traceback(t, g)$.
      replace any gate in Γ labeled with z_g by the formula Γ'.
    end for
  end if
  Process Γ to make any z-variable occur at most 2^{k+1} times using Proposition 3.
  return Γ.
end if

relabeling. Hence the treewidth of the modified circuit is at most k. Next, any gate g labeled with a field constant α, with edges going to gates $f_1, f_2, \ldots, f_m$, is replaced by m separate copies of $g_1, g_2, \ldots, g_m$, each labeled with α, where we add edges (g_i, f_i), for all $i \in [m]$. This does not increase the treewidth, as it can be thought of as a two step procedure, neither of which increases treewidth: first removing the vertex g and attached edges, secondly, adding back the isolated copies. Observe that now we have obtained an equivalent circuit Φ' that is multiplicatively disjoint. Namely, for purpose of contradiction, suppose there exists a multiplication gate $f = f_1 \times f_2$ such that both f_1 and f_2 are reachable from some gate h. Then there exists such an h for which the paths to f_1 and f_2 are edge disjoint. For this h, since Φ' is syntactically multilinear, there cannot be variables in the subcircuit Φ'_h. Hence h is a gate computing a constant. Since the paths to f_1 and f_2 are edge disjoint, h must have out-degree at least two. This contradicts the fact that any gate computing a constant in Φ' has out degree one. The statement sm-VP$[tw = O(1)] \subseteq$ VNC$_1$ now follows from Theorem 7 and the fact that VP$_e$ = VNC^1 [7].

To get the strengthened conclusion that sm-VP$[tw = O(1)] \subseteq$ sm-VNC$_1$, we will now indicate how to modify Algorithm 1 to ensure syntactic multilinearity. We use the notation of the proof of Theorem 7. Assume we have done preprocessing as indicated above. We know each circuit $\Phi_{t,f}$ is syntactically multilinear, for all $t \in T$, and $f \in X_t$. The goal is to establish inductively that each $\Gamma_{t,f}$ is syntactically multilinear, for all $t \in T$, and $f \in X_t$.

At the base case, i.e. line 3 of Algorithm 1, we can simply enforce the condition by brute force. At line 7, by induction Γ_{t,f_0} and Γ_{t,f_1} are syntactically multilinear. If $\circ = +$, then so is Γ. In case $\circ = \times$, whenever the formulas Γ_{t,f_0} and Γ_{t,f_1} share a variable α, since we know $\lceil \Gamma \rceil = \lceil \Phi_{t,f} \rceil$ is multilinear, α does not appear in at least one of the polynomials $\lceil \Gamma_{t,f_0} \rceil$ and $\lceil \Gamma_{t,f_1} \rceil$. Setting α to zero in the corresponding formula ensures Γ is syntactically multilinear.

We now argue how to correctly deal with the substitution on line 14, and the processing of z variables on line 17. Consider Γ as computed on line 11. We want to ensure it is in the following standard form: $\sum_{a\in\{0,1\}^{k+1}} \left(\prod_{i\in[k+1]} z_i^{a_i}\right) f_a(\overline{x})$, for certain polynomials $f_a \in F[X]$. For this we use the following modification of Proposition 3, which is obtained by multiplying out the factors $\prod_{i\in[k+1]}(1 - z_i)^{1-b_i} z_i^{b_i}$. For $a, a' \in \{0,1\}^{k+1}$, we say $a' \leq a$ iff $\{i : a'_i = 1\} \subseteq \{i : a_i = 1\}$. We denote the size of $\{i : a'_i = 1\}$ by $|a'|$. We leave the proof of the following proposition as an exercise:

Proposition 4. *Let $f(\overline{x}, z_1, z_2, \ldots, z_{k+1})$ be given that is multilinear in $\overline{z}$. Write $f(\overline{x}, z_1, z_2, \ldots, z_{k+1}) = \sum_{a\in\{0,1\}^{k+1}} \left(\prod_{i\in[k+1]} z_i^{a_i}\right) coef(f, z_1^{a_1} z_2^{a_2} \ldots z_{k+1}^{a_{k+1}})$, then it holds that $coef(f, z_1^{a_1} z_2^{a_2} \ldots z_{k+1}^{a_{k+1}}) = \sum_{a'\leq a} (-1)^{|a|-|a'|} f(\overline{x}, a')$.*

If we use the above proposition to process z-variables on line 17, then by induction, Γ on line 11 will indeed have the required form, or for simplicity one can also assume we do an extra step of z-variable processing. That is, assume we apply above proposition to get Γ in the required form. This requires at most $(2^{k+1})^2$ copies of Γ and blows up Γ by an inconsequential factor of $2^{O(k)}$. Observe that this leaves Γ syntactically multilinear.

Now consider line 14. First of all, any $z_g \in Var(\Gamma) \backslash Var(\lceil \Gamma \rceil)$ can be set to zero in Γ. For the remaining z-variables, we claim that for any pair $z_g, z_h \in Var(\lceil \Gamma \rceil)$, whenever $\Gamma_{t,g}$ and $\Gamma_{t,h}$ share a variable α, then $coef(\lceil \Gamma \rceil, m) = 0$, for any multilinear monomial m in the z-variables of Γ that contains both z_g and z_h. Hence we can remove these terms from the standard form of Γ, and avoid multilinearity conflicts among products between each of the substituted formulas.

We will verify this claim using the notion of a *proof tree*. A proof tree rooted at a gate g in a circuit C is any tree obtained by recursively selecting gates, starting with g, as follows: 1) at an addition gate select exactly one of its children, and 2) at a multiplication gate select both children. We will consider proof trees of $\Phi_{t_{i_1},f}$ rooted at f. For a subset Z of z-variables in $\Phi_{t_{i_1},f}$, we let $\mathrm{PTree}(Z)$ stand for the collection of proof trees rooted at f that have precisely the z-variables in Z appearing at its leaves. Given $T \in \mathrm{PTree}(Z)$, let $p(T)$ denote the product of all X variables appearing in T. The following proposition is easily proved by structural induction on the circuit $\Phi_{t_{i_1},f}$.

Proposition 5. *For any multilinear monomial m in z-variables used in $\Phi_{t_{i_1},f}$, it holds that $coef(\lceil \Phi_{t_{i_1},f} \rceil, m) = \sum_{T\in PTree(Z)} p(T)$, where Z is the set of z-variables of m.*

Recall that by induction $\lceil \Gamma \rceil = \lceil \Phi_{t_{i_1},f} \rceil$. Now consider any multilinear monomial m in z-variables of $\lceil \Phi_{t_{i_1},f} \rceil$ with both z_g and z_h in it, where $\Gamma_{t,g}$ and $\Gamma_{t,h}$ share a variable α. For purpose of contradiction suppose $coef(\lceil \Phi_{t_{i_1},f} \rceil, m) \neq 0$. By Proposition 5 this means there exists a proof tree in $\Phi_{t_{i_1},f}$ rooted at f that contains both z_g and z_h. This implies g and h are reachable from a single multiplication gate r in $\Phi_{t_{i_1},f}$, and hence also in $\Phi_{t,f}$. Observe that our construction satisfies the property that for any $t \in V[T]$ and $f \in X_t$, $Var(\Gamma_{t,f}) \subseteq Var(\Phi_{t,f})$. Hence α appears in both $\Phi_{t,g}$ and $\Phi_{t,h}$. Observe that both α's must be reachable from r in $\Phi_{t,f}$. This contradicts the fact that $\Phi_{t,f}$ is syntactically multilinear.

Similarly, one can verify that whenever for a variable $z_g \in Var(\lceil \Gamma \rceil)$, the formula $\Gamma_{t,g}$ contains a variable α, then $coef(\lceil \Gamma \rceil, m)$ does not contain α for any monomial m containing z_g. Hence any occurrence of α in the formula $\sum_{a' \leq a} (-1)^{|a|-|a'|} \Gamma(\overline{x}, a')$ used to compute $coef(\lceil \Gamma \rceil, m)$ can be replaced by zero.

We conclude that under above modifications, Algorithm 1 yields a syntactically multilinear formula $\Gamma_{t,f}$ equivalent to $\Phi_{t,f}$. The proof is completed with the observation of [14] that Brent's construction [7], which shows $\mathrm{VP}_e \subseteq \mathrm{VNC}^1$, preserves syntactic multilinearity. □

3.2 Evaluation over a Finite Field and Boolean Implications

The observation is that Algorithm 1, when applied over $GF(2)$ to an arbitrary n-input arithmetic circuit Φ, will result in a formula Γ such that for any $a \in GF(2)^n$, $\lceil \Phi \rceil(a) = \lceil \Gamma \rceil(a)$. For this, no assumptions regarding the multiplicative disjointness of Φ is needed. One can prove this condition using structural induction similarly as in Lemma 2. For the processing of the z-variables on line 17, observe that we have the following adaption of Proposition 3:

Proposition 6. *Let $f(x_1, \ldots, x_n, z_1, \ldots, z_{k+1})$ be a polynomial over $GF(2)$, and let $g = \sum_{b \in \{0,1\}^{k+1}} \left(\prod_{i \in [k+1]} (1 - z_i)^{1-b_i} z_i^{b_i} \right) f(x_1, x_2, \ldots, x_n, b_1, b_2, \ldots, b_{k+1})$. Then for any $a \in GF(2)^{n+k+1}$, $f(a) = g(a)$.*

One can generalize this to an arbitrary finite field F of size q, by similarly using brute force on line 17 of Algorithm 1 to make sure any z-variables appears at most q^{k+1} times in Γ. Consequently, we have the following theorem:

Theorem 8. *Let F be a finite field, and let $q = |F|$. For any arithmetic circuit Φ over F of size s and treewidth k, there exists a formula Γ over F of size at most $s^{O(k^2 \log q)}$ such that Φ and Γ evaluate to identical values for inputs in F.*

A proof of the following proposition will appear in the full version.

Proposition 7. *For every Boolean circuit C of fan-in two and treewidth k, there is an arithmetic circuit C' over $GF(2)$ of treewidth $3k$ such that $\forall x \in \{0,1\}^n$, $C(x) = 1$ if and only if $C'(x) = 1$.*

Proof of Theorem 1. Given a Boolean circuit of size s and treewidth k of bounded fan-in (wlog. assume fan-in two), first convert it into an arithmetic circuit over $GF(2)$ using Proposition 7. Now apply Theorem 8 to obtain an arithmetic formula Γ over $GF(2)$ of size $s^{O(k^2)}$. Balance this formula down to depth $O(k^2 \log s)$ using [7]. Now do the reverse construction of arithmetization and code out an $\{\wedge, \vee, \neg\}$-formula computing the same function. The final circuit has depth $O(k^2 \log s)$. Thus we have proven Theorem 1. □

We can use a similar reduction to derive a Boolean analogue of Theorem 2. The proof will appear in the full version of the paper. Let TWC^i denote the class of Boolean functions computed by Boolean circuits of treewidth $O(\log^i n)$.

Theorem 9. *The following two statements hold in the non-uniform setting: 1)* $\mathrm{SC}^0 \subseteq \mathrm{TWC}^0 \subseteq \mathrm{SC}^1$*, and 2)* $\forall i \geq 1, \mathrm{SC}^i \subseteq \mathrm{TWC}^i \subseteq \mathrm{SC}^{i+1}[size = n^{O(\log \log n)}]$.

4 Constant Width Circuits

We make the following definition: an *iterated multiplication chain* of length ℓ in a circuit Φ is given by a sequence of gates $g_0, g_1, \ldots, g_\ell$, where all are multiplication gates, except possibly g_0, such that both inputs of g_i are reachable from g_{i-1}, for all $i \in [\ell]$. We denote the length of a longest iterated multiplication chain in Φ by $\mathcal{M}(\Phi)$. Note that if $\mathcal{M}(\Phi) = 0$, then Φ is multiplicatively disjoint. The following theorem will be proved in the full version of the paper:

Theorem 10. *For any leveled arithmetic circuit Φ of size s and width w, there exists equivalent formula of depth $d = O(w^4 \mathcal{M}(\Phi) \log s)$ and size at most 2^d.*

Theorem 5 immediately follows from Theorem 10. Note that conversely one has the inclusion $\mathrm{VNC}^1 \subseteq \mathrm{VSC}^0[\mathcal{M} = O(1)]$, due to [5].

5 Application to Circuit Evaluation and Reachability

We use Proposition 7 to reduce the proof of Theorem 6 to the following proposition. We note this reduction can be computed within *logspace*.

Proposition 8. *Given an arithmetic circuit C over $GF(2)$ together with its tree decomposition $(T, (X_d)_{d \in V[T]})$ of constant width k and an input $x \in \{0,1\}^n$, testing whether $C(x) = 1$ can be done in* LogDCFL.

Proofsketch. The proof proceeds by analyzing *Traceback*. We are given the circuit C and an input x. We replace each gate of C labeled by x_i with its Boolean value. Next we run *Traceback* to compute an equivalent formula. We claim this can be implemented in *poly*-time and $O(\log n)$ workspace, provided we use a stack (whose space usage is not counted towards the space bound). This claim will be substantiated in the paper's full version. □

A proof of following proposition will appear in the full version of the paper. Corollary 1 follows from it.

Proposition 9. *Given a DAG $G = (V, E)$ of bounded treewidth and bounded in-degree and $s, t \in V$, we can obtain a circuit C of bounded treewidth and an input x such that $C(x) = 1$ if and only if t is reachable from s in the graph G.*

References

1. Allender, E.: Reachability problems: An update. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 25–27. Springer, Heidelberg (2007)
2. Arvind, V., Joglekar, P., Srinivasan, S.: On lower bounds for constant width arithmetic circuits. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 637–646. Springer, Heidelberg (2009)
3. Barrington, D., Lu, C.-J., Miltersen, P., Skyum, S.: On monotone planar circuits. In: IEEE Conference on Computational Complexity, pp. 24–31 (1999)
4. Barrington, D.M.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In: Proc. 18th STOC, pp. 1–5 (1986)
5. Ben-Or, M., Cleve, R.: Computing algebraic formulas using a constant number of registers. In: Proc. 20th STOC, pp. 254–257 (1988)
6. Bodlaender, H.L.: NC-algorithms for graphs with small treewidth. In: Nagl, M. (ed.) WG 1989. LNCS, vol. 411, pp. 1–10. Springer, Heidelberg (1990)
7. Brent, R.: The parallel evaluation of general arithmetic expressions. J. Assn. Comp. Mach. 21, 201–206 (1974)
8. Bürgisser, P., Claussen, M., Shokrollahi, M.: Algebraic Complexity Theory. Springer, Heidelberg (1997)
9. Goldschlager, L.M.: The monotone and planar circuit value problems are logspace complete for P. SIGACT News 9(2), 25–29 (1977)
10. Jansen, M., Rao, B.: Simulation of arithmetical circuits by branching programs with preservation of constant width and syntactic multilinearity. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) Computer Science - Theory and Applications. LNCS, vol. 5675, pp. 179–190. Springer, Heidelberg (2009)
11. Jones, N.: Space-bounded reducibility among combinatorial problems. J. Comp. Sys. Sci. 11, 68–85 (1975); Corrigendum. J. Comp. Sys. Sci. 15, 241 (1977)
12. Limaye, N., Mahajan, M., Sarma, J.: Upper bounds for monotone planar circuit value and variants. Computational Complexity 18(3), 377–412 (2009)
13. Mahajan, M.: Polynomial size log depth circuits: between NC^1 and AC^1. Technical Report 91, BEATCS Computational Complexity Column (2007)
14. Mahajan, M., Rao, B.: Arithmetic circuits, syntactic multilinearity, and the limitations of skew formulae. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 455–466. Springer, Heidelberg (2008)
15. Mahajan, M., Rao, B.: Small-space analogues of Valiant's classes. In: Gębala, M. (ed.) FCT 2009. LNCS, vol. 5699, pp. 455–466. Springer, Heidelberg (2009)
16. Malod, G., Portier, N.: Characterizing valiant's algebraic complexity classes. J. Complex. 24(1), 16–38 (2008)
17. Raz, R.: Separation of multilinear circuit and formula size. In: Proc. 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 344–351 (2004)
18. Reingold, O.: Undirected st-connectivity in log-space. In: Proc. 37th Annual ACM Symposium on the Theory of Computing, pp. 376–385 (2005)
19. Savitch, W.: Maze recognizing automata and nondeterministic tape complexity. J. Comput. Syst. Sci. 7(4), 389–403 (1973)
20. Valiant, L., Skyum, S., Berkowitz, S., Rackoff, C.: Fast parallel computation of polynomials using few processors. SIAM J. Comput. 12, 641–644 (1983)

Obtaining Online Ecological Colourings by Generalizing First-Fit

Matthew Johnson[1,⋆], Viresh Patel[1,⋆⋆], Daniël Paulusma[1,⋆⋆⋆], and Théophile Trunck[2]

[1] Department of Computer Science, Durham University,
Science Laboratories, South Road, Durham DH1 3LE, U.K
{matthew.johnson2,viresh.patel,daniel.paulusma}@dur.ac.uk
[2] Ecole Normale Supérieure de Lyon,
46 Allée d'Italie, 69364 Lyon Cedex 07, France
theophile.trunck@ens-lyon.fr

Abstract. A colouring of a graph is ecological if every pair of vertices that have the same set of colours in their neighbourhood are coloured alike. We consider the following problem: if a graph G and an ecological colouring c of G are given, can further vertices added to G, one at a time, be coloured using colours from some finite set C so that at each stage the current graph is ecologically coloured? If the answer is yes, then we say that the pair (G, c) is ecologically online extendible. By generalizing the well-known First-Fit algorithm, we are able to characterize when (G, c) is ecologically online extendible. For the case where c is a colouring of G in which each vertex is coloured distinctly, we give a simple characterization of when (G, c) is ecologically online extendible using only the colours of c, and we also show that (G, c) is always online extendible if we permit ourselves to use one extra colour. We also study (off-line) ecological H-colourings where the colouring must satisfy further restrictions imposed by some fixed pattern graph H. We characterize the computational complexity of this problem. This solves an open question posed by Crescenzi et al.

1 Introduction

One of the goals of social network theory is to determine patterns of relationships amongst actors in a society. Social networks can be represented by graphs, where vertices of the graph represent individuals and edges represent relationships amongst them. One way to study patterns of relationships in such networks is to assign *labels* in such a way that those who are assigned the same label have similar sorts of relationships within the network; see e.g. Hummon and Carley [9]. Several graph-theoretic concepts such as ecological colourings [1], role assignments [6] and perfect colourings [2], have been introduced to facilitate the study of social networks in this way.

This paper focuses on ecological colourings. The term "ecological" is derived from certain models of population *ecology* in which individuals are assumed to be determined by their environment. For example, in biology, features of a species' morphology are

⋆ Supported by EPSRC Grant EP/E048374/1.
⋆⋆ Supported by EPSRC Grant EP/F064551/1.
⋆⋆⋆ Supported by EPRSC Grant EP/G043434/1.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 240–251, 2010.

usually defined in relation to the way such a species interacts with other species. Also, some network theories of attitude formation assume that one's attitude is predicted by the combination of the attitudes of surrounding individuals [3,5].

We introduce some basic notation and terminology. Throughout the paper, all graphs are undirected and without loops or multiple edges unless otherwise stated. We denote the vertex and edge sets of a graph G by V_G and E_G respectively. An edge between u and v is denoted (u, v). The *neighbourhood* of u in G is denoted $N_G(u) = \{v \mid (u, v) \in E_G\}$. For a subset $S \subseteq V_G$ and a function c on V_G (for example, a colouring of the vertices), we use the short-hand notation $c(S)$ for the set $\{c(u) \mid u \in S\}$. The *colourhood* of a vertex v in a graph G with colouring c is defined to be $c(N_G(v))$. For a set C, we write $A + x$ to denote $A \cup \{x\}$ for some subset $A \subseteq C$ and $x \in C$.

Ecological colourings were introduced by Borgatti and Everett in [1] to analyse power in experimental exchange networks. Formally, an *ecological colouring* of a graph $G = (V, E)$ is a vertex mapping $c : V \to \{1, \ldots\}$ such that any $u, v \in V$ with the same *colourhood*, i.e. with $c(N(u)) = c(N(v))$, have the same colour $c(u) = c(v)$. Note that such a colouring does not have to be *proper*, i.e. two adjacent vertices may receive the same colour. This reflects that two individuals that play the same role in their environment might be related to each other. See Fig. 1 for an example of a proper ecological colouring.

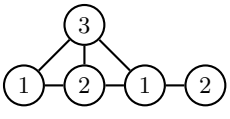


Fig. 1. A proper ecological colouring that is also an ecological K_3-colouring

One of the appealing features of ecological colourings is a result of Crescenzi et al. [4]. In order to state the result precisely, we need to introduce some terminology. A *twin-free* graph (also known as a neighbourhood-distinct graph) is a graph in which no two vertices have the same neighbourhood (including empty neighbourhoods). A graph G that is not twin-free can be made twin-free as follows: whenever we find a pair of vertices u and v for which $N_G(u) = N_G(v)$, we delete one of them until no such pair remains. It is easy to check that the resulting graph is independent of the order in which vertices are deleted and is twin-free; it is called the *neighbourhood graph* of G and is denoted by G_N. The main result of Crescenzi et al. [4] states that an ecological colouring of a graph G using exactly k colours can be found in polynomial time for each $1 \leq k \leq |V_{G_N}|$ and does not exist for $k \geq |V_{G_N}| + 1$.

Our motivation for studying online ecological colourings. In static optimization problems, one is often faced with the challenge of determining efficient algorithms that solve a particular problem optimally for any given instance of the problem. In the area of *dynamic* optimization the situation is more complicated: here, one often lacks knowledge of the complete instance of the problem.

This paper studies ecological colourings for dynamic networks. Gyárfás and Lehel [7] introduced the concept of *online colouring* to tackle dynamical storage allocations. An *online colouring algorithm* irrevocably colours the vertices of a graph one by one, as

they are revealed, where determination of the colour of a new vertex can only depend on the coloured subgraph induced by the revealed vertices. See [10] for a survey on online colouring.

Perhaps the most well-known online colouring algorithm is FIRST-FIT. Starting from the empty graph, this algorithm assigns each new vertex the least colour from $\{1, 2, \ldots\}$ that does not appear in its neighbourhood. It is easy to check that an ecological colouring is obtained at each stage and hence FIRST-FIT is an example of an *on-line ecological colouring algorithm*. Note, however, that it may use an unbounded number of colours. If we wish to use at most k colours when we start from the empty graph, then we can alter FIRST-FIT so that each new vertex v is assigned, if possible, the least colour in $\{1, 2, \ldots, k-1\}$ not in the colourhood of v, or else v is coloured k. We call the modified algorithm k-FIRST-FIT. It gives a colouring that is ecological but not necessarily proper (cf. 1-FIRST-FIT which assigns all vertices the same colour).

A natural situation to consider is when we are given a nonempty *start graph* $G_0 = G$, the vertices of which are coloured by an ecological colouring c. At each stage i, a new vertex v_i is added to G_{i-1} (the graph from the previous stage) together with (zero or more) edges between v_i and the vertices of G_{i-1}, to give the graph G_i. Knowledge of G_i is the *only* information we have at stage i. Our task is to colour the new vertex v_i at each stage i, without changing the colours of the existing vertices, to give an ecological colouring of G_i. If there exists an online colouring algorithm that accomplishes this task using some colours from a finite set $C \supseteq c(V_G)$, we say that the pair (G, c) is *(ecologically) online extendible* with C. Sometimes we do not give C explicitly and say simply that (G, c) is *(ecologically) online extendible*. Motivated by our observation that colourings obtained by FIRST-FIT and k-FIRST-FIT are ecological, we examine which pairs (G, c) are online extendible.

Our motivation for studying ecological H-colourings. In order to analyse the salient features of a large network G, it is often desirable to compress G into a smaller network H in such a way that important aspects of G are maintained in H. Extracting relevant information about G becomes much easier using H. This idea of compression is encapsulated by the notion of *graph homomorphisms*, which are generalizations of graph colourings. Let G and H be two graphs. An *H-colouring* or *homomorphism* from G to H is a function $f : V_G \to V_H$ such that for all $(u, v) \in E_G$ we have $(f(u), f(v)) \in E_H$. An *ecological* H-colouring of G is a homomorphism $f : V_G \to V_H$ such that, for all pairs of vertices $u, v \in V_G$, we have $f(N_G(u)) = f(N_G(v)) \implies f(u) = f(v)$. See Fig. 1 for an example of an ecological K_3-colouring, where K_3 denotes the complete graph on $\{1, 2, 3\}$. The ECOLOGICAL H-COLOURING problem asks if a graph G has an ecological H-Colouring. Classifying the computational complexity of this problem is our second main goal in this paper. This research was motivated by Crescenzi et al. [4] who posed this question as an interesting open problem.

Our results and paper organisation. In Section 2, we characterize when a pair (G, c), where G is a graph and c is an ecological colouring of G, is online extendible with a fixed set of colours C. We then focus on the case where each vertex of a k-vertex graph G is coloured distinctly by c. We show that such a pair (G, c) is always online extendible with $k+1$ colours, and give a polynomial-time online colouring algorithm for achieving this. We show that this result is tight by giving a simple characterization of exactly

which (G, c) are not ecologically online extendible with k colours. This characterization can be verified in polynomial time. In Section 3, we give a complete answer to the open problem of Crescenzi et al. [4] and classify the computational complexity of the ECOLOGICAL H-COLOURING problem. We show that if H is bipartite or contains a loop then ECOLOGICAL H-COLOURING is polynomial-time solvable, and is NP-complete otherwise. Section 4 contains the conclusions and open problems.

2 Online Ecological Colouring

We first give an example to demonstrate that not all pairs (G, c) are online extendible. Consider the ecologically coloured graph in Fig. 2(i). Suppose that a further vertex is added as shown in Fig. 2(ii). Its colourhood is $\{1, 3, 4\}$ so it must be coloured 2 to keep the colouring ecological (since there is already a vertex with that colourhood). Finally suppose that a vertex is added as shown in Fig. 2(iii). Its colourhood is $\{2, 3, 4\}$ so it must be coloured 1. But now the two vertices of degree 2 have the same colourhood but are not coloured alike so the colouring is not ecological.

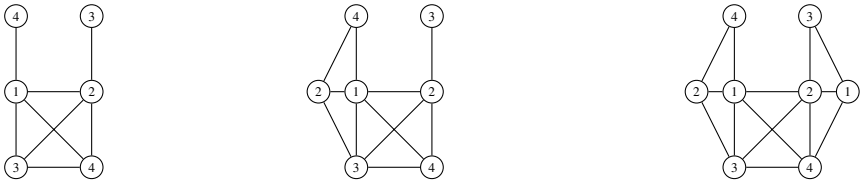


Fig. 2. A pair (G, c) that is not online extendible

We also give an example of a pair (G, c) that is online extendible but for which we cannot use FIRST-FIT or k-FIRST-FIT. Let G be the path $v_1v_2v_3v_4$ on four vertices coloured $abcd$. We will show in Theorem 2 that (G, c) is online extendible (even if we are forced to use only colours from $\{a, b, c, d\}$). However, FIRST-FIT or k-FIRST FIT (arbitrary k) cannot be used with any ordering of $\{a, b, c, d\}$. To see this, add a new vertex adjacent to v_1 and v_3. Any correct online colouring algorithm must colour it b. So if the algorithm is FIRST-FIT, b is before d in the ordering of the colours. Next add a new vertex adjacent to v_3. If this vertex is not coloured d then the colouring will not be ecological, but FIRST-FIT will not use d as b (or possibly a) is preferred.

Let us now describe our general approach for obtaining online ecological colourings when they exist. As before, let G be a graph with an ecological colouring c and let C be a set of colours where $C \supseteq c(V_G)$. What we would like to do is to write down a set of instructions: for each subset $A \subseteq C$, a colour x should be specified such that whenever a vertex is added and its colourhood is exactly A, we will colour it x. We would like to construct a fixed set of instructions that, when applied, always yields ecological colourings. We make the following definitions.

(i) A *rule* on C is a pair $A \subseteq C$ and $x \in C$ and is denoted $A \to x$.
(ii) A rule $A \to x$ *represents* a vertex v in G if v has colourhood A and $c(v) = x$.
(iii) The set of rules that represent each vertex of G is said to be *induced* by (G, c) and is denoted $R_{(G,c)}$.
(iv) A set of rules R on C is *valid* for (G, c) if $R \supseteq R_{(G,c)}$ and R contains at most one rule involving A for each subset $A \subseteq C$.
(v) A set of rules R on C is *full* if R contains exactly one rule involving A for each subset $A \subseteq C$.

Notice that a full set of rules R constitutes an online colouring algorithm: if v is a newly revealed vertex with colourhood A and $A \to x$ is the unique rule for A in R, then v is coloured x by R. Notice also that the k-FIRST-FIT algorithm can be written down as the full set of rules

$$R^k_{FF} = \{A \to \min\{y \geq 1 \mid y \notin A\} \mid A \subset \{1, \ldots, k\}\} \cup \{\{1, \ldots, k\} \to k\},$$

that is, the k-FIRST-FIT algorithm assigns colours to new vertices purely as a function of their colourhoods. In this way, the notion of rules generalises FIRST-FIT. There is no reason a priori that a general online colouring algorithm should follow a set of rules; however, one consequence of Theorem 1 below is that every online *ecological* colouring algorithm can be assumed to follow a set of rules.

While a full set of rules R gives an online colouring algorithm, it does not guarantee that each colouring will be ecological. For this, we must impose conditions on R. The following observation, which follows trivially from definitions, shows that having a valid set of rules for a coloured graph ensures that it is ecologically coloured. We state the observation formally so that we can refer to it later.

Observation 1. *Let $G = (V, E)$ be a graph with colouring c. Let R be a valid set of rules on some $C \supseteq c(V)$ for (G, c). Then c is an ecological colouring of G.*

Proof. Suppose c is not ecological. Then there are two vertices coloured x and y, $x \neq y$, which both have colourhood $A \subseteq c(V) \subseteq C$. Then the set of rules induced by (G, c) contains two rules $A \to x$ and $A \to y$. Since R is valid for (G, c), it must contain the rules induced by (G, c), but this contradicts that R must contain at most one rule for each $A \subseteq C$. □

Note, however, that if we have a valid and full set of rules R on C for (G, c) and further vertices are added and coloured according to the rules, R might not necessarily remain valid for the new graph, that is, R might not be a superset of the induced rules for the new graph. Let us see what might happen. Suppose that a new vertex u is added such that the colours in its neighbourhood are B and that, according to a rule $B \to y$ in R, it is coloured y. Now consider a neighbour v of u. Suppose that it had been coloured x at some previous stage according to a rule $A \to x$ in R. But now the colour y has been added to its colourhood. So R is valid for the altered graph only if it contains the rule $A + y \to x$. This motivates the following definition. Let R be a set of rules on $C \supseteq c(V_G)$. We say that R is a *good* set of rules on C if for any $A, B \subseteq C$ and $x, y \in C$ the following holds:

$$\text{if } (A \to x) \in R \text{ and } (B \to y) \in R \text{ and } x \in B \text{ then } (A + y \to x) \in R.$$

It is an easy exercise to check that the rules R^k_{FF} for k-FIRST-FIT are good.

We are now able to present the main results of this paper. First we characterize when a pair (G, c) is online extendible.

Theorem 1. *Let G be a graph with ecological colouring c. Then (G, c) is online extendible with a finite set C if and only if there exists a set of rules that is valid for (G, c), good, and full on C', where C' is a set of colours satisfying $C \supseteq C' \supseteq c(V_G)$.*

The purpose of C' in the statement of Theorem 1 is to account for the possibility that some of the colours of C may, under all circumstances, not be required.

Proof. ($\Longrightarrow$) If (G, c) is online extendible with finite C, then there exists, by definition, an algorithm α that can be used to obtain an ecological colouring of any graph constructed by adding vertices to G. We shall show that, by carefully choosing how to add vertices to G and colouring them with α, we can obtain a graph which induces a set of rules that is valid for (G, c), good and full on some set $C' \subseteq C$.

First, we describe one way in which we add vertices. If a graph contains a vertex u coloured x with colourhood A, then the set of rules induced by the graph includes $A \to x$. To *protect* that rule means to add another vertex v with the same neighbourhood (and thus also the same colourhood) as u, to colour it x (as any correct algorithm must), and to state that no further vertices will be added that are adjacent to v. Hence all future graphs obtained by adding additional vertices will also induce the rule $A \to x$.

We use this method immediately: we protect each of the rules induced by (G, c). In this way, we ensure that the set of induced rules for any future graph is valid for (G, c).

As long as the set of rules R induced by the current graph G^* is not full for the set of colours C^* used on G^*, we add a new vertex as follows:

> Let $B \subseteq C^*$ be a set for which R does not contain a rule. Add to G^* a new vertex u with colourhood B and use the algorithm α to obtain an ecological colouring. Add vertices to protect any rule induced by the new graph not in R.

Note that it is possible to add such a vertex u without making it adjacent to vertices that have been used for protection. There is at least one rule induced by the new graph not induced by the previous graph, namely $B \to y$, where y is the colour α assigns to u. So if we continue in this way, the number of rules will increase and eventually a full set of rules will be obtained for some set $C' \subseteq C$ (since, by definition, α only uses colours from such a set). Let G_F be the graph for which the induced rules are full. It only remains to prove that these rules are good.

If the rules are not good, then they include rules $A \to x$, $B \to y$, $A + y \to z$ such that $x \in B$ and $x \neq z$. Let u be a vertex in G_F coloured x with colourhood A. Choose a set of vertices $S \ni u$ coloured B such that each vertex, except possibly u, is not one that was created to protect a rule. Add a new vertex adjacent to the vertices of S. This must be coloured y by α. But now the neighbourhood of u is $A + y$ and the colouring is not ecological (since no vertex has been added adjacent to the vertex protecting the rule $A + y \to z$); this contradicts the definition of α.

($\Longleftarrow$) Suppose R is a set of rules that is valid for (G, c), good, and full on some C' where $C \supset C' \supseteq c(V)$. Set $G_0 = G$ and suppose, in the online process, vertices $v_1, \ldots, v_r$ are added one at a time to obtain graphs $G_1, \ldots, G_r$. Colouring each new

vertex according to R, we obtain the colouring c_i for G_i. We must show that c_i is an ecological colouring of G_i. By Observation 1, we can do this by proving inductively that R is valid for each (G_i, c_i).

We have that R is valid for (G_0, c). Assume, for induction, that R is valid for (G_{i-1}, c_{i-1}). Let B be colourhood of v_i in G_i. If R contains the rule $B \to y$, then v_i is coloured y, giving the colouring c_i of G_i. We must check that there are rules in R to represent each vertex of G_i. Clearly the rule $B \to y$ represents v_i. Let $v \neq v_i$ be a vertex of G_i. Suppose, as a vertex of G_{i-1}, it is represented by the rule $A \to x$. If v is not adjacent to v_i, then v is still represented by the rule $A \to x$ in G_i. If v is adjacent to v_i, then v is represented by the rule $A + y \to x$; this rule is present in R since R is good and contains the rules $A \to x$ and $B \to y$, where $x \in B$. □

Corollary 1. *Fix a colour set C. Given an input graph $G = (V, E)$ together with an ecological colouring c, where $c(V) \subseteq C$, the problem of deciding if (G, c) is ecologically online extendible with C is solvable in polynomial time.*

Proof. Suppose $|C| = \ell$ for some fixed integer ℓ. We enumerate all full sets of rules on all $C' \subseteq C$ and check whether they are good and valid for (G, c). The number of sets of rules to be checked depends only on ℓ and is independent of $|G|$, and checking a set of rules requires polynomial time. □

So far, we have not been able to prove the computational complexity of the above decision problem if C is part of the input. A natural question to start with is to consider the case in which all vertices of G have distinct colours. Thus we assume that G is twin-free else the colouring would not be ecological. Theorem 2 solves this case by showing that any such pair (G, c) is online extendible using one extra colour in addition to $c(V)$. We show in the second part of this theorem that the above is tight by characterizing those pairs (G, c) for which we always need the extra colour. The simple necessary and sufficient conditions in our characterization can easily be checked in polynomial time.

Theorem 2. *Let $G = (V, E)$ be a twin-free graph on k vertices and let c be a colouring of G with $|c(V)| = k$ (thus c is an ecological colouring of G).*

1. *(G, c) is online extendible with $c(V)$ and one extra colour.*
2. *(G, c) is online extendible with $c(V)$ if and only if G contains a vertex u^* such that*
   - *(i) the neighbourhood of u^* is maximal in G, that is $N(u^*)$ is not a proper subset of $N_G(v)$ for all $v \in V$, and*
   - *(ii) the graph $G - u^*$ is twin-free.*

The smallest twin-free graph that does not satisfy the two conditions (i) and (ii) in Theorem 2 is a graph on two components, one of which is an isolated vertex and the other is an edge. The smallest connected twin-free graph that does not satisfy these two conditions is obtained from a complete graph on four vertices u_1, u_2, u_3, u_4 after adding two new vertices v_1, v_2 with edges (u_1, v_1), (u_2, v_1), (u_3, v_2), (u_4, v_2) and (v_1, v_2). We can construct an infinite family of such examples as follows. Take two disjoint copies, H and H', of the complete graph K_{2n} on $2n$ vertices with a perfect matching removed. Let $(v_1, w_1), (v_2, w_2), \ldots, (v_n, w_n)$ be the perfect matching removed from H, and let $(v'_1, w'_1), (v'_2, w'_2), \ldots, (v'_n, w'_n)$ be the perfect matching removed from H'.

Let G be the graph obtained by adding the edges $(v_1, v_1'), (v_2, v_2'), \ldots, (v_n, v_n')$ to $H \cup H'$. Clearly, the vertices with maximal neighbourhoods are $v_1, \ldots, v_n, v_1', \ldots, v_n'$, but removing v_i (resp. v_i') from G leaves twins v_i', w_i' (resp. v_i, w_i).

Proof. We restate that G is a twin-free graph on k vertices and that c is an ecological colouring of G with $|c(V)| = k$. Define $C := c(V) = \{1, 2, \ldots, k\}$. To prove each part of the theorem, we must find a valid, good, full set of rules R for (G, c). We know that R must contain rules that represent each vertex of G; we must describe how to define the remaining rules. Here is a useful technique.

Let $\mathcal{A}$ contain the subsets $A \subseteq C$ for which $R_{(G,c)}$ contains a rule involving A. To *propagate* $R_{(G,c)}$ apply the following to obtain a set of rules $R^*_{(G,c)}$:

- for each $1 \leq i \leq |C|$, fix an ordering for the collection of sets in $\mathcal{A}$ of cardinality i;
- for each subset $A \subseteq C$, let, if possible, A^* be the smallest member of $\mathcal{A}$, first ordered, that is a superset of A (possibly $A^* = A$). If A^* exists and $A^* \to x$ is a rule in $R_{(G,c)}$, then add $A \to x$ to $R^*_{(G,c)}$.

We make two claims. The first is a simple observation.

Claim 1. We have that $R^*_{(G,c)}$ is valid for (G, c). Furthermore $R^*_{(G,c)}$ is a full set of rules on C if and only if $R_{(G,c)}$ contains a rule $C \to x$ for some x.

Claim 2. We have that $R^*_{(G,c)}$ is good.

We prove Claim 2 as follows. If $R^*_{(G,c)}$ is not good, then there are rules $A \to x$ and $B \to y$ in $R^*_{(G,c)}$, where $x \in B$, but $A + y \to x$ is not in $R^*_{(G,c)}$. By definition, $R_{(G,c)}$ contains a rule $A^* \to x$. Notice that A^* is the set of colours used on the neighbours of the vertex in G coloured x. Similarly $R_{(G,c)}$ must contain a rule $B^* \to y$, where $x \in B \subseteq B^*$ and B^* is the set of colours used on the neighbours of the vertex in G coloured y. So the vertices in G coloured x and y are adjacent and so $y \in A^*$. But then A^* contains $A + y$ so we must have $A^* = (A + y)^*$. Thus $A + y \to x$ is in $R^*_{(G,c)}$. This proves Claim 2.

We now prove the first part of the theorem. Let G' be obtained from G by adding a new vertex v^* adjacent to all existing vertices and to itself (we could avoid having a loop by adding two new vertices adjacent to every vertex in G and each other; but allowing the loop makes the analysis a little tidier). Colour v^* with colour $k + 1$ to obtain a colouring c' of G', and write $C' = \{1, \ldots, k + 1\}$. Note that G' is twin-free.

As $R^*_{(G',c')}$ contains a rule involving C', Claim 1 tells us that it is a full and valid set of rules on C' for (G', c'). By Claim 2, $R^*_{(G',c')}$ is also good. It remains only to show that $R^*_{(G',c')}$ is valid for (G, c).

Note that each vertex v of G has the colour $k+1$ in its G'-neighbourhood. Therefore, as a vertex of G', v is represented in $R_{(G',c')}$ by a rule $A + (k + 1) \to x$ (where A is the set of colours in the G-neighbourhood of v). Observe that, since A^* is a minimal set containing A that is involved in a rule of $R_{(G',c')}$, and since all rules $B \to y$ in $R_{(G',c')}$ satisfy $k + 1 \in B$, we have $A^* = A + (k + 1)$. Thus $R^*_{(G',c')}$ contains the rule $A \to x$, which represents the vertex v of G. This is true for all vertices of G, and so $R^*_{(G',c')}$ is also valid for (G, c). Thus $R^*_{(G',c')}$ is a full set of rules on $C' = \{1, \ldots, k + 1\}$ that is good and valid for (G, c). Thus (G, c) is online extendible with $\{1, \ldots, k + 1\}$ by Theorem 1. This completes the proof of the first part of Theorem 2.

Now we prove the second part of the theorem.

($\Longrightarrow$) We begin by showing that if G contains a vertex u^* such that $G - u^*$ is twin-free and the neighbourhood of u^* in G is maximal (that is, it is not a proper subset of the neighbourhood of another vertex in G), then (G, c) is online extendible with $c(V) = C = \{1, \ldots, k\}$. If we can construct a full set of rules on C that is good and valid for (G, c) then we are done by Theorem 1.

We may assume that u^* is coloured k. Let G' be obtained from G by adding edges to G so that u^* is adjacent to every vertex in G, including itself. Note that G' is twin-free: u^* is the only vertex adjacent to every vertex in the graph and if two other vertices both have neighbourhoods $A + u^*$, then in G one must have neighbourhood A and the other $A + u^*$, contradicting that $G - u^*$ is twin-free.

Let $R^*_{(G',c)}$ be obtained from $R_{(G',c)}$ by propagation. As $R_{(G',c)}$ contains the rule $C \to k$, we have that $R^*_{(G',c)}$ is a full set of rules on C that is valid for (G', c) by Claim 1 and that is good for (G', c) by Claim 2.

It remains only to show that $R^*_{(G',c)}$ is valid for (G, c). Note that for each vertex $v \neq u^*$ of G, if $c(N_G(v)) = A$, then $R_{(G',c)}$ contains the rule $A + k \to x$. Also $R^*_{(G',c)}$ contains the rule $A \to x$ as $A^* = A + k$ (since A^* is a minimal superset of A and must contain k). In G, the set of colours in the neighbourhood of v is either A or $A + k$; in either case there is a rule in $R^*_{(G',c)}$ to represent it.

Let B be the colours in the neighbourhood of u^* in G. Then $B^* = C$ as, by the maximality of B, there is no other superset of B involved in a rule of $R_{(G',c)}$. Since $R_{(G',c)}$ contains $C \to k$, $R^*_{(G',c)}$ contains $B \to k$ which represents u^*. So $R^*_{(G',c)}$ is valid for (G, c) as required.

($\Longleftarrow$) Suppose that for every vertex u^* of G, either $G - u^*$ is not twin-free or the neighbourhood of u^* in G is not maximal. We show that (G, c) is not online extendible with $C = \{1, \ldots, k\}$.

Suppose, for a contradiction, that there is an online algorithm to extend (G, c). Add vertex v to G adjacent to all vertices in G to form G_1. Without loss of generality, our algorithm assigns colour k to v to give us a colouring c_1 of G_1. Let u be the vertex of $G_0 := G$ that is coloured k. There are two cases to consider: either $G_0 - u$ is not twin-free or $N_{G_0}(u)$ is not maximal.

Suppose $G_0 - u$ has twins, that is two vertices a and b with the same neighbourhood (in $G_0 - u$). The colouring $c_0 = c$, and therefore c_1, colours a and b differently; however we have $c_1(N_{G_1}(a)) = c_1(N_{G_1}(b))$, a contradiction.

Suppose $N_{G_0}(u)$ is not maximal; suppose $S = N_{G_0}(u)$ and $T = N_{G_0}(u')$, where $T = S \cup \{t_1, \ldots, t_r\}$. Let $N_i = N_{G_0}(t_i)$. (Note that $r \neq 0$ since $G_0 = G$ is twin-free.) Add vertices $w_1, \ldots, w_r$ to G_1 one at a time, where w_i is adjacent to each vertex in $N_i \cup \{u\}$. Our online algorithm is forced to assign the colour of t_i to w_i (since they have the same colours in their neighbourhoods). Let G_{r+1} be the graph obtained after addition of $w_1, \ldots, w_r$ and let c_{r+1} be its colouring. In G_{r+1}, c_{r+1}, we find that u and u' have the same set of colours in their neighbourhoods but are coloured differently (since they were coloured differently by c_0). This is a contradiction. □

Now we show that the online algorithms implied by Theorem 2 run in polynomial time. Let G be a twin-free graph with ecological colouring c. We compute the sets A^* required for the propagation. When a new vertex v_i is presented and needs to be coloured,

we first determine the set of colours A in the neighbourhood of v_i. We then compute A^* by checking whether A is a subset of any member of $\mathcal{A}$, and if so, finding the smallest and first ordered such member of $\mathcal{A}$. This determines the rule by which v_i should be coloured and can be done in time polynomial in the size of G.

3 Ecological H-Colouring

Crescenzi et al. [4] mention that ECOLOGICAL K_3-COLOURING is NP-complete and ask if the computational complexity of ECOLOGICAL H-COLOURING can be classified. We classify the computational complexity of the ECOLOGICAL H-COLOURING problem for all fixed target graphs H.

Before doing this, we must introduce some further terminology. Given a graph H on k vertices, we define the *product graph* H^k. The vertex set of H^k is the Cartesian product

$$V_{H^k} = \underbrace{V_H \times \cdots V_H}_{k \text{ times}}.$$

Thus a vertex u of H^k has k *coordinates* u_i, $1 \leq i \leq k$, where each u_i is a vertex of H (note that these coordinates of u need not be distinct vertices of H). The edge set of H^k, E_{H^k}, contains an edge (u, v) in E_{H^k} if and only if, for $1 \leq i \leq k$, there is an edge (u_i, v_i) in H. For $1 \leq i \leq k$, the *projection* on the ith coordinate of H^k is the function $p_i : V_{H^k} \to V_H$ where $p_i(u) = u_i$. It is clear that each projection is a graph homomorphism.

Theorem 3. *If H is bipartite or contains a loop, then* ECOLOGICAL H-COLOURING *is in P. If H is not bipartite and contains no loops, then* ECOLOGICAL H-COLOURING *is* NP*-complete.*

Proof. The first sentence of the theorem is an easy observation which we briefly justify. If H has no edges, then G has an ecological H-colouring if and only if G has no edges. Suppose H is bipartite and contains at least one edge (x, y). If G is bipartite, then we can find an ecological H-colouring by mapping each vertex of G to either x or y. If G is not bipartite then it is clear that there is no homomorphism from G to H. If H has a loop, then any graph has an ecological H-colouring since we can map every vertex to a vertex with a loop.

We prove that the ECOLOGICAL H-COLOURING problem is NP-complete for loopless non-bipartite H by reduction from H-COLOURING which is known to be NP-complete for loopless non-bipartite H [8].

Let G be an instance of H-colouring and let n be the number of vertices in G. Let k denote the number of vertices in H_N, the neighbourhood graph of H (recall that the neighbourhood graph of H is a graph in which each vertex has a unique neighbourhood and is obtained from H by repeatedly deleting one vertex from any pair with the same neighbourhood). Let π denote a vertex in H_N^k whose k coordinates are the k distinct vertices of H_N (the order is unimportant). Let G' be a graph formed from G and n

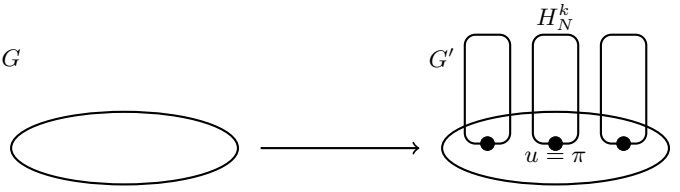


Fig. 3. The graph G' formed by attaching G to copies of H_N^k

copies of H_N^k by identifying each vertex u of G with a distinct copy of the vertex π; see Fig. 3. We can distinguish the copies of H_N^k by referring to the vertex of G to which they are *attached*.

We claim that G has an H-colouring if and only if G' has an ecological H_N-colouring which is clearly equivalent to G' having an ecological H-colouring. As it is clear that if G' has an ecological H_N-colouring, the restriction to V_G provides an H-colouring for G, all we need to prove is that when G has an H-colouring, we can find an ecological H_N-colouring for G'.

If G has an H-colouring, then clearly it also has an H_N-colouring f. We use f to find an ecological H_N-colouring g for G'. For each vertex $u \in V_G$, $f(u) = \pi_i$ for some i (this is possible because of the choice of π as a vertex that has each vertex of H_N as a coordinate). For each vertex v in the copy of H_N^k attached to u, let $g(v) = p_i(v)$. Note that $g(u) = p_i(u) = \pi_i = f(u)$ for each vertex u in V_G.

Certainly g is an H_N-colouring: the edges of E_G are mapped to edges of H_N since g is the same as f on V_G, and the edges of each copy of H_N^k are mapped to edges of H_N as g is the same as one of the projections of H_N^k on these edges.

We must show that it is ecological; that is, for each pair of vertices s and t in G', we must show that

$$g(N_{G'}(s)) = g(N_{G'}(t)) \implies g(s) = g(t). \tag{1}$$

Suppose that $g(N_{G'}(s)) = g(N_{G'}(t))$. We know that $g(s) = p_i(s) = s_i$ for some value of i. Then for each $x \in N_{H_N}(s_i)$, there is a vertex $s' \in N_{G'}(s)$ with $g(s') = x$ (since we can choose as s' a vertex in the same copy of H_N^k as s with $s'_i = x$ and s'_j being any neighbour of s_j, $1 \leq j \leq k$, $j \neq i$). Thus $g(N_{G'}(s)) \supseteq N_{H_N}(s_i)$ and so, since g is an H_N-colouring, $g(N_{G'}(s)) = N_{H_N}(s_i)$ and then, by (1), $g(N_{G'}(t)) = N_{H_N}(s_i)$. But as the neighbourhoods of vertices in H_N are distinct, we must have $g(t) = s_i = g(s)$. This completes the proof of Theorem 3. □

4 Conclusions and Open Problems

In the first part of our paper, we show that checking whether a pair (G, c) is online extendible with some finite set $C \supseteq c(V_G)$ can be done in polynomial time for fixed C. Determining the computational complexity of this problem when C is part of the input remains an open problem. We obtain a positive result when considering pairs (G, c) in which each vertex of the k-vertex graph G has a distinct colour. For such (G, c), we

can check in time polynomial in k if (G, c) is online extendible with any $C \supseteq c(V_G)$. Indeed, we find that if $|C| = k + 1$, then (G, c) is always online extendible with C, and there are infinitely many examples of (G, c) that are not online extendible with C when $|C| = k$. It would be interesting to know whether there are examples of graphs that can be extended online with an infinite number of colours but not with a finite number. We have not been able to find such examples.

In the second part of our paper we gave a complete computational complexity classification of the ECOLOGICAL H-COLOURING problem, thus answering an open problem posed in [4]. What about the computational complexity of the problems that ask whether a given graph G allows an edge-surjective or vertex-surjective ecological colouring to a fixed target graph H? If H is not a neighbourhood graph, G allows neither an edge-surjective nor a vertex-surjective ecological colouring. Hence, both problems differ from the ECOLOGICAL H-COLOURING problem and are only interesting for neighbourhood graphs H. We note, however, that determining the complexity of the corresponding problems that ask if a graph allows an edge-surjective homomorphism, or a vertex-surjective homomorphism, respectively, to a fixed graph H are notoriously difficult open problems.

References

1. Borgatti, S.P., Everett, M.G.: Graph colorings and power in experimental exchange networks. Social Networks 14, 287–308 (1992)
2. Borgatti, S.P., Everett, M.G.: Ecological and perfect colorings. Social Networks 16, 43–55 (1994)
3. Burt, R.S.: Structure Version 4.2 Reference Manual, Center for the Social Sciences. Columbia University, New York (1991)
4. Crescenzi, P., Di Ianni, M., Greco, F., Rossi, G., Vocca, P.: On the existence of Ecological Colouring. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 90–100. Springer, Heidelberg (2008)
5. Erickson, E.H.: The relational basis of attitudes. In: Social Structures: A Network Approach, pp. 99–121. Cambridge University Press, Cambridge (1988)
6. Everett, M.G., Borgatti, S.P.: Role colouring a graph. Mathematical Social Sciences 21, 183–188 (1991)
7. Gyárfás, A., Lehel, J.: On-line and first-fit colorings of graphs. Journal of Graph Theory 12, 217–227 (1988)
8. Hell, P., Nešetřil, J.: On the complexity of H-colouring. Journal of Combinatorial Theory, Series B 48, 92–110 (1990)
9. Hummon, N., Carley, K.: Social networks as normal science. Social Networks 15, 71–106 (1993)
10. Kierstead, H.A.: Coloring graphs on-line. In: Fiat, A. (ed.) Online algorithms: the state of the art. LNCS, vol. 1442, pp. 281–305. Springer, Heidelberg (1998)

Classical Simulation and Complexity of Quantum Computations
(Invited Talk)

Richard Jozsa

DAMTP, Centre for Mathematical Sciences, University of Cambridge, Wilberforce Road, Cambridge CB3 0WA, U.K.

Abstract. Quantum computing is widely regarded as being able to offer computational complexity benefits beyond the possibilities of classical computing. Yet the relationship of quantum to classical computational complexity is little understood. A fundamental approach to exploring this issue is to study the extent to which quantum computations (especially with restricted sub-universal ingredients) can be classically efficiently simulated. We will discuss a series of results relating to the classical simulation of some interesting classes of quantum computations, particularly Clifford circuits, matchgate circuits and a simple class of quantum circuits (so-called IQP circuits) comprising commuting gates. We will outline an argument that a suitably efficient classical simulation of the latter class would imply a collapse of the polynomial hierarchy.

Keywords: Quantum computation, quantum complexity, classical simulation, matchgates.

1 Introduction

Quantum computing is widely regarded as being able to offer computational complexity benefits beyond the possibilities of classical computing. The non-classical phenomenon of quantum entanglement is sometimes quoted as an essential feature (cf [1]) but we still have very little understanding of how to exploit specific quantum effects for computational advantage, and the landscape bridging classical and quantum computational complexity remains largely uncharted. A fundamental approach to exploring these issues is to study the extent to which quantum computations (especially with restricted sub-universal ingredients) can be classically efficiently simulated. In this talk we will begin by briefly recalling the basic formalism of the quantum computational model and notions of classical simulation, and then discuss a series of results relating to the classical simulation of some interesting classes of quantum computations, and their implications for quantum versus classical computational complexity.

A uniform family of (quantum) circuits is a family of circuit descriptions C_w parameterised by bit strings $w = i_1 \dots i_n$, such that the mapping $w \to C_w$ is computable in classical log space. Thus uniform circuit families are always poly-sized. The output of C_w is the result of a standard quantum measurement (i.e.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 252–258, 2010.

in the computational basis) on one or more qubit lines, after C_w is applied to a nominated input state. The description C_w is deemed to include a specification of the input state and a specification of which lines are the output lines.

There are various possible notions of classical simulation for quantum circuit families. Let P_w denote the output distribution of C_w. We say that the circuit family is *strongly simulatable* if each of the output probabilites in P_w can be computed to m digits of precision (i.e. exponential precision) in classical poly(n, m) time. A circuit family is *weakly simulatable* if given the description of C_w, its output distribution P_w can be sampled by purely classical means in poly(n) time. A circuit family is *weakly simulatable with multiplicative error* $c \geq 1$ if there is a family R_w of distributions (on the same sample spaces as P_w) such that R_w can be sampled in classical poly(n) time and for all x and w we have

$$\frac{1}{c}\ \mathrm{Prob}(P_w = x) \leq \mathrm{Prob}(R_w = x) \leq c\ \mathrm{Prob}(P_w = x).$$

A further notion of approximate weak simulation has been formulated in [3]: recall first that the Chernoff-Hoeffding bound (cf Appendix of [3]) implies the following result – if we have a quantum process implementing C_w then by running it poly-many times we can (with probability exponentially close to 1) obtain an estimate $\tilde{p}$ of any output probability p to within polynomial precision i.e for any polynomial $f(n)$ we can output $\tilde{p}$ such that $|p - \tilde{p}| < 1/f(n)$. We say that the circuit family is (classically) *weakly simulatable with additive polynomial error* if the same estimates can be obtained from the circuit descriptions C_w by purely classical means in poly(n) time (and probability exponentially close to 1).

Note that if a uniform circuit family C_w decides a language L with bounded error probability $0 \leq \epsilon < \frac{1}{2}$ then the existence of a weak simulation for C_w implies that $L \in$ BPP. Similarly the existence of a weak simulation with additive polynomial error, or with multiplicative error $1 \leq c < 2(1 - \epsilon)$, will also imply that $L \in$ BPP. The class of languages decided with bounded error probability by a uniform quantum circuit family is conventionally denoted BQP. The term "efficient" will be used to mean "polynomial time".

Perhaps the earliest explicitly recognised example of the classical simulatability of a restricted class of quantum circuits is the Gottesman-Knill theorem [2,4] which we briefly recall. The Pauli group $\mathcal{P}_1$ on a single qubit comprises the standard Pauli matrices [2] X, Y and Z together with the identity matrix I and scalar multiples by powers of i. The Pauli group $\mathcal{P}_n$ on n qubits, a subgroup of the unitary group $U(2^n)$ in 2^n dimensions, is the n-fold tensor power of $\mathcal{P}_1$. The Clifford group $\mathcal{C}_n$ is then defined to be the normaliser of $\mathcal{P}_n$ in $U(2^n)$. The Gottesman-Knill theorem (slightly modified from the original form) asserts the following: suppose we have any uniform family of quantum circuits comprising Clifford gates with input being a product state (e.g. any computational basis state) and output given by a measurement on a single specified qubit. Then the output can be strongly simulated. In fact for Clifford circuits single qubit output probabilities are always 0,1 or $\frac{1}{2}$. An explicit characterisation of Clifford gates is known [4] and it is notable that such circuits can generate entanglements between the qubits. Thus a naive classical simulation by direct linear algebra

calculation of the evolving state vector components, fails to be efficient, and the special relationship between the Clifford and Pauli groups turns out to be able to provide an alternative method that *is* efficient.

2 Matchgate Circuits

Inspired by the simulation of Clifford circuits, it is interesting to seek further instances of mathematical structural properties that can be exploited for efficient classical simulation of quantum circuits. A second example is given by the theory of matchgates, introduced by Valiant [5], which have a series of remarkable properties. For us, a matchgate [5,9] is defined to be any 2-qubit gate $G(A, B)$ of the form (in the computational basis):

$$G(A,B) = \begin{pmatrix} p & 0 & 0 & q \\ 0 & w & x & 0 \\ 0 & y & z & 0 \\ r & 0 & 0 & s \end{pmatrix} \qquad A = \begin{pmatrix} p & q \\ r & s \end{pmatrix} \qquad B = \begin{pmatrix} w & x \\ y & z \end{pmatrix} \tag{1}$$

where A and B are both in $SU(2)$ or both in $U(2)$ with the *same determinant*. Thus the action of $G(A, B)$ amounts to A acting in the even parity subspace (spanned by $|00\rangle$ and $|11\rangle$) and B acting in the odd parity subspace (spanned by $|01\rangle$ and $|10\rangle$). Such gates arise in the theory of counting perfect matchings in weighted graphs [5,6] and also in aspects of quantum physics (especially the theory of so-called non-interacting fermions [8,7]). A fundamental simulation result of Valiant (slightly modified from [5] and elaborated in [9]) is the following: consider any matchgate circuit of size N such that (i) the matchgates act on nearest neighbour (n.n.) lines only, (ii) the input is any computational basis state $|i_1 \dots i_n\rangle$, and (iii) the output is the final measurement of any designated single qubit line. Then the output probabilities may be classically calculated to m digits of precision in poly(N, m) time. In particular uniform families of n.n. matchgate circuits can be classically strongly simulated. After the announcement of Valiant's result it was noticed [8,7] that this theorem is closely related to known simulation results for systems of non-interacting fermions in quantum physics. It is also interesting to note that, as emphasised in [10], the proof of Valiant's theorem may be cast into a form parallelling that of the Gottesman-Knill theorem, with suitable alternative mathematical structures substituting for the Pauli and Clifford groups.

3 Assessing Simulation Complexity and Extensions

A further feature that emerges from the above simulation proofs is the following: if we examine more closely the actual computations involved in the classical simulations then in some cases we can characterise the exact (sub classical-poly-time) computational power of the circuit classes. In this way Aaronson and Gottesman [11] showed that the computational power of Clifford circuits coincides,

in a natural way, with the classical complexity class $\oplus$L (of languages decided by classical nondeterministic log-space Turing machines which are deemed to accept if an odd number of computational paths halt with accept). More precisely, introduce the language $\mathcal{L}_{\text{clifford}}$ of descriptions of Clifford circuits such that on input $|0\ldots0\rangle$, an output measurement on the first qubit line yields 1 with probability 1 (recalling [2] that the latter probability is always 0 or $\frac{1}{2}$ or 1 for any Clifford circuit). Then is was shown that this language is complete for $\oplus$L (relative to log-space reductions).

For the case of matchgate circuits, the classical simulation method (based on the so-called Jordan-Wigner representation, cf [9] for an exposition) proceeds by translating the action of circuits of width n (hence acting in a state space of dimension 2^n) into the consideration of products of *orthogonal* matrices of size $2n \times 2n$. The orthogonality property is especially fortuitous here since we may then view these matrices as special cases of *unitary* matrices, and hence quantum operations, acting on $\log 2n = O(\log n)$ qubit lines. In this way we can obtain an exact relationship [12] between the power of matchgate circuits and quantum space bounded computation. For example in the setting of uniform circuit families we have the following theorem [12]: the computational power of log-space uniform n.n. matchgate circuit families (which generally have polynomial size and width) coincides with universal unitary poly-time quantum computation on $O(\log n)$ qubits (i.e. within a quantum log space bound). We obtain equalities between corresponding classes of decision problems both in the bounded error and unbounded error settings. This result is interesting in that it identifies the computational power of an algebraically restricted class (i.e. matchgates) of computations as coinciding with the power of full quantum computation subject only to a resource restriction (log space).

Given any class of quantum circuits that can be classically simulated (and hence offering no supra-classical computational benefit) it is interesting to consider what extra quantum ingredients suffice to restore full universal quantum computing power. In the case of matchgates we obtain a tantalizing result [9]: although circuits of matchgates acting on n.n. qubit lines are efficiently simulatable, if we allow the 2-qubit matchgates to act also on just *next*-nearest-neighbour lines, then we efficiently regain full universal quantum computation [9]. In this sense we could attribute any extra computational power of quantum over classical computation to the mere ability of matchgates to act on lines just distance one further apart! This (and other such results) may suggest that the gap between classical and quantum computational power might be considerably more subtle than initially intuitively envisaged. In view of the above, we may even be lead to ask whether (almost) all quantum computations might after all, turn out to be classically efficiently simulatable e.g. is BPP = BQP? This question is at present unresolved.

4 IQP Circuits

Although the above results about n.n. vs. next-n.n. matchgate circuits may suggest that the distinction between BPP and BQP could be delicate, we can also

develop other arguments supporting the presence of a substantial gap in computational power. We consider an especially simple class of quantum circuits, originally introduced in [13], called temporally unstructured (or instantaneous) quantum computations, and an associated class called IQP (of families of probability distributions that can be generated in "instantaneous quantum poly-time"). Here we will describe a restricted version of the formalism of [13] that will suffice for our purposes. Let $\mathcal{S}$ denote the class of all 2-qubit unitary gates which are *diagonal* in the computational basis and with diagonal entries all being powers of $e^{i\pi/8}$. An IQP circuit is defined to be any circuit of the following form: each qubit line begins and ends with a Hadamard gate H, and in between we have a circuit of gates drawn only from $\mathcal{S}$. The input state is always $|0\rangle\,|0\rangle \ldots |0\rangle$ and the output is a quantum measurement (in the computational basis) on a designated set of lines. A uniform family of IQP circuits is a family C_w of IQP circuit descriptions where $w \to C_w$ is computable in classical poly(n) time. In contrast to our previous notion of uniform family (used for classically simulatable circuits) it is convenient here to allow full poly-time (rather than just log-space) uniformity and to standardise the input states as always being $|0\rangle \ldots |0\rangle$. Let IQP denote the class of (families of) probability distributions that arise as outputs of uniform IQP circuit families.

IQP is indeed a very simple kind of quantum computational process. The gates of $\mathcal{S}$ all commute so (apart from the initial and final Hadamard gate on each line) the output is independent of the order in which the gates are applied, hence "temporally unstructured" or "instantaneous (as in quantum physics, commuting operations may be physically implemented simultaneously on a system). If we place two Hadamard gates on each line between each occurrence of a gate from $\mathcal{S}$ (recalling that $HH = I$) then we may absorb all H gates (including the initial and final ones on each line) into conjugation actions on the gates from $\mathcal{S}$. Thus alternatively we could define an IQP circuit simply as a circuit made entirely of gates diagonal in the X-basis $\{|0\rangle \pm |1\rangle\}$, with diagonal entries, input state and output measurements remaining as before.

Our main result about IQP computations is the following [14]: suppose that outputs from any uniform family of IQP circuits can be weakly simulated to within multiplicative error c with $1 \leq c < \sqrt{2}$. Then the classical polynomial hierarchy PH collapses to its third level, PH_3. This provides strong evidence that such weak classical simulation of IQP processes should not be possible and correspondingly, that quantum poly-time computational processes, utilising only such simple IQP processes as their quantum ingredients, should be able to provide complexity benefits surpassing the possibilities of classical computation.

We refer to the forthcoming preprint [14] for a full discussion and proof of this result and here we make only some remarks. An essential technical tool in the proof is the notion of a *post-selected* computation for a decision problem. This is a (classical or quantum) probabilistic process with output lines comprising a single line (x) and a further register of so-called post-selection lines (denoted y). Then we consider only runs of the process for which the y register is seen to output some fixed value, say $00\ldots0$ i.e. instead of using Prob(x) as

the accepting and rejecting probabilities, we use the conditional probabilities $\mathrm{Prob}(x|y = 00\ldots0)$. Aaronson [15] showed that the post-selected version post-BQP of BQP coincides with the classical class PP. As a preliminary lemma for our result above we show that the class called post-IQP, of languages decided with bounded error by uniform IQP circuits with post-selection, also equals post-BQP and hence PP. Then we can argue that if IQP circuits could be weakly simulated to within a multiplicative error as above, then we would have post-IQP $\subseteq$ post-BPP, where post-BPP is the post-selected version of BPP. The latter class (also known as $\mathrm{BPP}_{\mathrm{path}}$ [16,17]) is known [16] to be contained in the third level PH_3 of the polynomial hierarchy. Then putting all these relations together, the multiplicative-error weak simulatability of IQP processes would entail PP = post-IQP $\subseteq$ post-BPP $\subseteq \mathrm{PH}_3$, and then Toda's theorem would imply that PH = PH_3.

As a final remark we point out that it may be shown [14] that uniform families of IQP circuits *can* be classically weakly simulated (even without multiplicative error i.e. with $c = 1$) *provided that* the number of output lines is suitably small *viz.* growing only as $O(\log n)$. (The methods of [3] can also be used to give a weak simulation with additive polynomial error in these cases). In view of this it is notable that our argument above leading to a collapse of PH involves a hypothetical weak simulation (with multiplicative error) of distributions resulting from $O(n)$ lines of IQP processes (as we generally need to consider post-selection registers y of width $O(n)$). Thus the assumption of non-collapse of PH implies classical hardness of simulation only for IQP processes with a suitably large number of output lines. Recalling that the class BQP is defined in terms of only *single* line measurements, it would then still be not inconsistent to have BQP and BPP being equal with PH not collapsing.

Acknowledgments. This work was supported by the UK's EPSRC-funded QIP IRC network and the EC networks QICS and QAP.

References

1. Jozsa, R., Linden, N.: On the Role of Entanglement in Quantum Computational Speedup. Proc. R. Soc. Lond. A 459, 2011–2032 (2003)
2. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
3. Van den Nest, M.: Simulating Quantum Computers With Probabilistic Methods (preprint, 2009), http://arXiv.org/abs/0911.1624
4. Gottesman, D.: Stabilizer Codes and Quantum Error Correction, PhD thesis. California Institute of Technology, Pasadena, CA (1997)
5. Valiant, L.: Quantum Circuits that can be Classically Simulated in Polynomial Time. SIAM J. Computing 31, 1229–1254 (2002)
6. Valiant, L.: Holographic algorithms. SIAM J. Computing 37, 1565–1594 (2002)
7. Knill, E.: Fermionic Linear Optics and Matchgates (preprint, 2001), http://arXiv.org/abs/quant-ph/0108033
8. Terhal, B., DiVincenzo, D.: Classical Simulation of Noninteracting-fermion Quantum Circuits. Phys. Rev. A 65, 32325 (2002)

9. Jozsa, R., Miyake, A.: Matchghates and Classical Simulation of Quantum Circuits. Proc. Roy. Soc. Lond. A 464, 3089–3106 (2008)
10. Jozsa, R.: Embedding Classical into Quantum Computation. In: Calmet, J., Geiselmann, W., Mueller-Quade, J. (eds.) Mathematical Methods in Computer Science. LNCS, vol. 5393, pp. 43–49. Springer, Heidelberg (2008)
11. Aaronson, S., Gottesman, D.: Improved Simulation of Stabiliser Circuits. Phys. Rev. A 70, 052328 (2004)
12. Jozsa, R., Kraus, B., Miyake, A., Watrous, J.: Matchgate and Space-bounded Quantum Computations are Equivalent. Proc. Roy. Soc. Lond. A 466, 809–830 (2010)
13. Shepherd, D., Bremner, M.: Temporally Unstructured Quantum Compuitation. Proc. Roy. Soc. Lond. A 465, 1413–1439 (2009)
14. Bremner, M., Jozsa, R., Shepherd, D.: Preprint in preparation (2010). A preliminary version of some of this work was presented in a poster at QIP 2010, Zürich (January 2010)
15. Aaronson, S.: Quantum Computing, Post-selection and Probabilistic Polynomial-time. Proc. Roy. Soc. Lond. A 461, 3473–3483 (2005)
16. Han, Y., Hemaspaandra, L., Thierauf, T.: Threshold Computation and Cryptographic Security. SIAM Journal on Computing 26, 59–78 (1997)
17. Kuperberg, G.: How Hard is it to Approximate the Jones Polynomial? (preprint 2009), http://arXiv.org/abs/0908.0512

Prefix-Free and Prefix-Correct Complexities with Compound Conditions

Elena Kalinina

Department of Mechanics and Mathematics,
Moscow State University, Russia
kalinina.helen@yandex.ru

Abstract. In this paper we compare two types of conditional prefix complexity assuming that conditions are sets of strings rather than strings. We show that prefix-free and prefix-correct versions of the definition are essentially different.

1 Introduction

A function f (arguments and values are binary words) is called *prefix-correct* if

$$f(x) \text{ is defined and } x \leqslant x' \Rightarrow f(x') = f(x)$$

Here $x \leqslant x'$ means that x is a prefix of x'. A *prefix-correct description mode, or method* is a computable prefix-correct function (the arguments and values are binary words). *Prefix complexity* of a word z with respect to description mode f is the length of the shortest x such that $f(x) = z$ (in this case x is called a description of z w.r.t. f):

$$KP_f(z) = \min\{l(x) \mid f(x) = z\}.$$

It is known that there exists an *optimal* prefix-correct description mode, such that for all other f' there exists a constant c, and

$$KP_f(z) \leqslant KP_{f'}(z) + c$$

for all z. We fix some optimal prefix-correct description method. In what follows we call complexity w.r.t. this optimal method *prefix complexity* $KP(z)$ of z. Similarly to the plain Kolmogorov complexity, complexities with respect to different optimal prefix-correct description methods differ at most by an additive constant. Thus $KP(z)$ is defined up to an additive constant.

The definition above was given by Levin in [3], [4] and Gács in [2]. Chaitin in [1] used a different definition based on prefix-free functions. A function is *prefix-free* if every two words from its domain are incomparable (one of them cannot be a prefix of the other). There exist optimal prefix-free description methods; we fix one of them and denote prefix-free complexity by $KP'(z)$.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 259–265, 2010.

It is known that these two variants of prefix complexity are equal up to an additive term $O(1)$. This follows from bounds

$$-\log m(z) \leqslant KP(z) + O(1) \leqslant KP'(z) + O(1) \leqslant -\log m(z),$$

where $m(z)$ is the a priori probability of z (m is the maximal enumerable semi-measure; it is unique up to a multiplicative constant).

The definitions above have their conditional versions. A function f of two arguments (both are binary words) is called *prefix-correct in the first argument* if for every fixed value of the second argument we get a prefix-correct function. If $f(x, y) = z$ then x is called a description of z conditional on y, and

$$KP_f(z|y) = \min\{l(x) \mid f(x, y) = z\}.$$

Again, the class of such description methods has an optimal one: there is f such that for all g we have $KP_f(z|y) \leqslant KP_g(z|y) + c$ for some c and all y, z. We fix one of them and denote the corresponding conditional complexity by $KP(z|y)$.

This definition can be modified: instead of prefix-correct description modes (in the first argument) we can consider prefix-free descriptions modes (in the first argument). For this class of description methods there is also an optimal description method. We denote the corresponding conditional complexity by $KP'(z|y)$.

These two definitions differ from each other by only an additive constant:

$$KP(z|y) = KP'(z|y) + O(1) = -\log m(z|y) + O(1),$$

where $m(z|y)$ is a conditional version of a priori probability.

In the paper [6], the plain conditional Kolmogorov complexity $KS(z|y)$ was generalized to the case when condition consists of many words. Let f be a computable function of two arguments, z a binary word, and Y a set of binary words. Complexity $KS_f(z\|Y)$ is defined as the minimal length of x such that $f(x, y) = z$ *for every* $y \in Y$. Again there exists an optimal f, and we denote the corresponding complexity by $KS(z\|Y)$. The goal of [6] was to define Kolmogorov complexity for a larger class of algorithmic problems than problems of the type "print the string a". In general, any set of strings Z has its (plain) Kolmogorov complexity defined as $\min\{KS(z) \mid z \in Z\}$. The conditional completity $KS(z\|Y)$ is essentially the complexity of the algorithmic problem "transform any element of Y to z".

The prefix version of complexity $KS(z\|Y)$ was introduced in [7]. For every binary z and a set of binary words Y we call *conditional prefix complexity* $KP_f(z\|Y)$ *of* z *conditional on* Y the minimal length of a word x such that $f(x, y) = z$ for all $y \in Y$. There is an f in the class of all partial computable functions that are prefix-correct in the first argument such that for every other f' in the class there exists c such that $KP_f(z\|Y) \leqslant KP_{f'}(z\|Y) + c$ for all z and all Y. We call such a function *strongly optimal*. We fix a strongly optimal function and denote the corresponding prefix complexity conditional on a set by $KP(z\|Y)$.

Similarly, there is a strongly optimal prefix-free (in the first argument) description method; we denote the corresponding complexity of z conditional on a set Y by $KP'(z\|Y)$.

These definitions are the counterparts of the usual $KP(z)$ and $KP'(z)$ (and their conditional variants $KP(z|y)$, $KP'(z|y)$) and coincide with them up to an additive constant in the case $Y = \{\text{empty string}\}$ (respectively, $Y = \{y\}$).

Let us define also a priori probability $m(z\|Y)$ as infimum of $m(z|y)$ over all $y \in Y$. It is not hard to see that for all z, Y we have

$$-\log m(z\|Y) \leqslant KP(z\|Y) + O(1) \leqslant KP'(z\|Y) + O(1)$$

Is it true that all the three complexities in these inequality coincide up to an additive constant (as it happens to $KP(z|y)$, $KP'(z|y)$, and $-\log m(z|y)$)? This question was raised in [7].

The partial positive answer to this question was known in one special case. Let n be a natural and Y_n the set of all naturals greater than n. Denote complexities conditional on this Y_n by $KP(z| \geqslant n)$, $KP'(z| \geqslant n)$ and $m(z| \geqslant n)$. In [7] it was proven that for all z the limits of the sequences (as n tends to infinity) $-\log m(z| \geqslant n)$, $KP(z| \geqslant n)$, $KP'(z| \geqslant n)$ coincide (up to an additive constant).

In this paper we show that in general case the answer to the question is negative, even for sets Y of cardinality at most 2.

2 Inequality between Two Types of Conditional Prefix Complexity

It is easy to prove the following inequality (for all z and Y):

$$KP(z\|Y) \leqslant KP'(z\|Y) + O(1)$$

Indeed, let g be a strongly optimal prefix-free (in the first argument) function. We can convert it in a strongly optimal prefix function f as follows. For a given pair (x, y) we run in parallel computations of $g(x', y)$ for all x' that are prefixes of x. Since $g(\cdot, y)$ is prefix-free, at most one of these computations converges. When we find x' such that $g(x', y)$ is defined, we set $f(x, y) = g(x', y)$.

By the construction, f is computable, prefix-correct in the first argument, and it is a continuation of g. Hence, complexity w.r.t. f is not greater than that w.r.t. g. The prefix-correct conditional complexity is not greater (up to an additive constant) than KP_f, and we are done.

Now we state the result of this paper:

Theorem 1. *For all c there are z, Y such that*

$$KP(z\|Y) \leqslant KP'(z\|Y) - c.$$

Proof. Let g be the strongly optimal prefix-free function. It suffices to show that there exists a prefix-correct in the first argument function f such that for c there are z, Y with

$$KP_f(z\|Y) \leqslant KP'_g(z\|Y) - c. \tag{1}$$

First we fix any constant c and construct a function f that satisfies the required inequality for that value of c. We will define f by exhibiting an algorithm computing f. More precisely, we will define an algorithm A that enumerates all the triples (x, y, z) with $f(x, y) = z$. The algorithm A will run the algorithm computing g on all inputs in a dovetailed fashion and, observing appearing values of g, it will define f's values on certain arguments. During this procedure, when $f(x, y)$ becomes defined, we also mentally define $f(x', y)$ for all continuations $x' \geqslant x$ of x.

The algorithm will find a string z and a finite (1- or 2-element) set Y satisfying (1). At each time the algorithm is aware only about finitely many values of g and thus has at his hand a restriction of g on a finite set of pairs (x, y). If that restriction satisfies (1) for the currently constructed g, we will say that we "are fine". If for certain z, Y the inequality (1) remains true forever, we will say that the algorithm A is successful. We will prove that this is indeed the case.

The main idea is as follows. The algorithm is allowed to define f on continuations of strings where f has been defined previously, whereas g cannot get defined on continuations of strings in the domain of g. We will use this property of g to force g "run out of space" or to satisfy (1).

The algorithm A runs in $2^{c+1} + 1$ stages. On stage i we will use descriptions of length i for f. Stage i will consists of 2^{i-1} steps. And each step will basically consist of a huge number (about $2^{2^{2^{c+1}}}$) of applications of a certain Procedure.

Stage 0. Take a large enough finite set Z of strings (the cardinality of Z depends on c, we specify it later). Choose for every $z \in Z$ a string y_z and set $f(\Lambda, y_z) = z$ for all $z \in Z$ (here Λ is the empty word). Run the algorithm computing g for all possible inputs in a dovetailed fashion. Wait until for every $z \in Z$ there appears x_z of length less than c with $g(x_z, y_z) = z$. If it never happens, we are fine, as then there exists a z such that $0 = KP_f(z\|\{y_z\}) \leqslant KP'_g(z\|\{y_z\}) - c$.

The number of words of length less than c is $2^c - 1$, so by pigeon hole principle we can find x such that $x_z = x$ for an essential fraction of z from Z. We delete from Z all other values of z and denote $\{x\} = X$. The cardinality of Z has reduced at most by a factor of 2^c.

Stage i ($1 \leqslant i \leqslant 2^{c+1}$). At the start of Stage i we have a (large enough) set Z and a set of binary words X such that

1. $|X| = 2^{i-1}$.
2. $\forall x \in X : |x| < c + i - 1$.
3. $\forall z \in Z, \forall x \in X : g(x, y_z) = z$.

The goal of the stage is to add 2^{i-1} words of length less than $c + i$ to X at the expense of reducing Z so that property 3 remain true (or to make (1) forever true for some z, Y).

Stage i consists of 2^{i-1} steps. On each step we add to X one word of length less than $c+i$ keeping property 3 true. On each step the algorithm works as follows.

Step k ($1 \leqslant k \leqslant 2^{i-1}$): As we have yet made less that 2^{i-1} steps, we have $|X| < 2^i$. We say that a $z \in Z$ is *good* if

$$\exists x : x \notin X, \quad |x| < c+i, \quad g(x, y_z) = z,$$

and *bad* otherwise. While the number of bad elements in Z is greater than $|X|$ run the following procedure.

> *Procedure*: Select a subset of bad words $Z' \subset Z$ of cardinality $|X|+1$. Pick an arbitrary v such that $f(\cdot, v)$ is not yet defined anywhere. Then, for all $z \in Z'$, choose a word u_z of length i and set $f(u_z, v) = z$. There are enough unused words u_z for this purpose, as $|Z'| = |X|+1 \leqslant 2^i$ and the number of words of length i is 2^i. So we get
>
> $$\forall z \in Z' \quad KP_f(z \| \{y_z, v\}) = i,$$
>
> since $f(\Lambda, y_z) = z$ and hence $f(u_z, y_z) = z$.
> Continue computation of g and wait until it happens that
>
> $$\forall z \in Z' \quad KP'_g(z \| \{y_z, v\}) < c+i.$$
>
> If it never happens, we are fine. Otherwise, for every $z \in Z'$ there is x_z of length less than $c+i$ with $g(x_z, y_z) = z$. As $|X| < |Z'|$, for some $z \in Z'$ we have $x_z \notin X$. Thus, we have acquired a new good word x_z.

Every application of the Procedure decreases the number of bad words. Once the number of bad words has become at most $|X|$, for at least $|Z| - |X|$ strings $z \in Z$ there is $x_z \notin X$ with $|x_z| < c+i$ and $g(x_z, y_z) = z$. By pigeonhole principle there is x such that for at least $2^{-c-i}(|Z| - |X|)$ string $z \in Z$ we have $x_z = x$. We remove from Z all other z, add this x to X and proceed to the next step. (End of Step k.)

We keep adding elements in X until $|X|$ reaches the required cardinality (to this end it suffices to make Step k for $k = 1, \ldots, 2^{i-1}$). Then we proceed to the next stage $i+1$. (End of Stage i.)

We have to show that it is impossible to complete all $2^{c+1}+1$ stages (that is, the algorithm succeeds on some stage). Indeed, all words in X are pairwise incomparable, since they are from the domain of the prefix-free function g. On Stage the set X gets 2^{i-1} new words of length at most $c+i-1$. If all $2^{c+1}+1$ stages have finished, we get a contradiction with the Kraft–McMillan inequality:

$$\sum_{i=0}^{2^{c+1}} 2^{i-1} \cdot \frac{1}{2^{c+i}} > 1.$$

So we get z, Y such that $KP_f(z \| Y) \leqslant KP'_g(z \| Y) - c$.

How to generalize this argument so that to construct f that beats all constants c? Since the initial cardinality of Z is finite and depends only on c, we can run the algorithm A in parallel for all c. We need to ensure that all sets Z_c and are pairwise disjoint. Also we need to use different descriptions for different c's. To this end we can prefix each description the algorithm $A(c)$ is about to use by a self-delimiting description of c in $O(\log c)$ bits. In this way we will construct a function f such that for all c there are z, Y with

$$KP_f(z\|Y) \leqslant KP'_g(z\|Y) - c + O(\log c).$$

As c is arbitrary, the statement of the theorem is not weakened in this way. □

The next theorem is a more refined version of our result. It provides an upper bound of z as a function of c in the previous theorem.

Theorem 2. *For all c there are z, Y such that*

$$KP(z\|Y) \leqslant KP'(z\|Y) - \log\log|z| + O(\log\log\log|z|)$$

and $|z| = 2^{2^{c+1}} + O(1)$, the set Y consists at most of two words of length $|z| + O(1)$.

Proof. To show this we have to estimate the initial cardinality of Z_c. Since there are less than 2^{c+i} words of length less than $c+i$, adding every new word in X costs reducing $|Z_c|$ by a factor of at most 2^{c+i} (the pigeon hole principle). On each stage we add 2^{i-1} words, and the number of stages is at most 2^c. Hence, in the course of 2^c stages Z_c gets reduced by a factor at most

$$((2^{c+2^c})^{2^{2^c}})^{2^c} \sim 2^{2^{2^{c+1}}}.$$

And we need Z_c to be non-empty on all stages. So, everything is OK if the initial cardinality of Z_c is $O(2^{2^{2^{c+1}}})$. Since z is chosen in Z_c, it holds $c = \log\log|z| + O(1)$.

Required set Y consists of one word y_z (if the goal has been reached on stage 0) or two words $\{y_z, v\}$ (otherwise). Obviously, $|\{y_z | z \in Z_c\}| = |Z_c|$, so we can choose as y_z strings of length $2^{2^{c+1}} + O(1)$. On each step we used at most $|Z_c|$ words called v in the construction. The number of steps is at most 2^{2^c}, so the number if needed v's is at most $2^{2^c} \cdot |Z_c| \sim O(2^{2^{2^{c+1}}})$ and we can choose strings of length $2^{2^{c+1}} + O(1)$ as v's. □

Is the lower bound of the last theorem tight? The only known upper bound for $KP'(z\|Y)$ in terms of $KP(z\|Y)$ is the following. For every word z and set Y,

$$KP'(z\|Y) \leqslant KP(z\|Y) + O(\log|z|)$$

Indeed, let $g([|x|]x, y) = f(x, y)$, where $[|x|]$ is a self-delimited representation of the length of x. With respect to g, we have

$$KP'_g(z\|Y) \leqslant KP_f(z\|Y) + O(\log|z|).$$

Thus there is an exponential gap between the known lower and upper bounds for $KP'(z\|Y) - KP(z\|Y)$.

References

1. Chaitin, G.J.: A theory of program size formally identical to information theory. Journal of the ACM 22, 329–340 (1975)
2. Gács, P.: On the symmetry of algorithmic information. Soviet Math. Dokl. 15(5) (1974)
3. Levin, L.A.: Laws of information conservation (non-growth) and aspects of the foundation of probability theory. Problems of Information Transmission 10, 206–210 (1974)
4. Levin, L.A.: The various measures of the complexity of finite objects (an axiomatic description). Soviet Math. Dokl. 17, 522–526 (1976)
5. Li, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn., p. 638. Springer, Heidelberg (1997)
6. Shen, A., Vereshchagin, N.K.: Logical operations and Kolmogorov complexity. Theoretical Computer Science 271, 125–129 (2002)
7. Uspensky, V.A., Vereshchagin, N.K., Shen, A.: Kolmogorov complexity (a textbook in preparation)

Monotone Complexity of a Pair*

Pavel Karpovich

Moscow State Lomonosov University
pkarpovich@mail.ru

Abstract. We define monotone complexity $KM(x, y)$ of a pair of binary strings x, y in a natural way and show that $KM(x, y)$ may exceed the sum of the lengths of x and y (and therefore the a priori complexity of a pair) by $\alpha \log(|x| + |y|)$ for every $\alpha < 1$ (but not for $\alpha > 1$).

We also show that decision complexity of a pair or triple of strings does not exceed the sum of its lengths.

1 Introduction

There are different versions of Kolmogorov complexity: plain complexity (C), prefix complexity (K), decision complexity (KR), monotone complexity (KM), etc. Let us recall the definitions of plain, monotone and decision complexities in a form suitable for generalizations (see [7,8]).

1.1 Plain Complexity

Kolmogorov complexity $C_F(x)$ of a binary string x with respect to a computable function F (a decompressor) is defined as

$$C_F(x) = \min\{|p| : F(p) = x\},$$

where $|p|$ stands for the length of a binary string p. There exists an optimal decompressor U such that C_U is minimal up to $O(1)$; $C_U(x)$ is then called (plain) Kolmogorov complexity of x.

Let us reformulate this definition in a way that is parallel to the definition of monotone complexity. Instead of a function F let us consider its graph. A *description mode* is an enumerable set W of pairs of binary strings that is a graph of a function, i.e.,

$$\langle p, x\rangle \in W,\ \langle p', x'\rangle \in W,\ p = p' \ \Rightarrow\ x = x',$$

If $\langle p, x\rangle \in W$, then p is called a *description for x with respect to W*. The complexity $C_W(x)$ of a binary string x is the minimal length of a description for x

* The work was performed while visiting LIF Marseille (CNRS & Univ. Aix–Marseille); the visit was made possible by the CNRS France–Russia exchange program; preparation of the final text was supported also by NAFIT ANR 008-01 grant).

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 266–275, 2010.

with respect to W. There is an optimal description mode S such that for every description mode W there exists c_W such that

$$C_S(x) \leqslant C_W(x) + c_W$$

for every binary string x. The corresponding function C_S is plain Kolmogorov complexity.

Monotone complexity. We use the definition of monotone complexity $KM(x)$ suggested by L. A. Levin. (Levin [2] gave a criterion of Martin-Löf randomness in its terms: a binary sequence ω is Martin-Löf random if and only if $|x| - KM(x) \leqslant c$ for some constant c and all prefixes x of sequence ω; here $|x|$ denotes the length of a string x. Earlier a similar criterion was proven by Schnorr who used a different version of complexity, called "process complexity".) Let us recall the definition of monotone complexity in terms of binary relations. A *monotone description mode* is an enumerable set W of pairs of binary strings such that:

- if $\langle p, x\rangle \in W$ and $p \preceq p'$, then $\langle p', x\rangle \in W$.
- if $\langle p, x\rangle \in W$ and $x' \preceq x$, then $\langle p, x'\rangle \in W$.
- if $\langle p, x\rangle \in W$ and $\langle p, x'\rangle \in W$, then $x \preceq x'$ or $x' \preceq x$.

Here $x \preceq x'$ means that x is a prefix of x' (or $x = x'$). The intuition behind this definition: a binary string u is considered as partial information about an infinite sequence that has prefix u; then $p \preceq p'$ means that p' is a refinement of p, so if p describes x, every $p' \succeq p$ should also describe x, and so on.

If $\langle p, x\rangle \in W$, then p is called a *description for x with respect to W*. The monotone complexity $KM_W(x)$ of x with respect to a monotone description mode W is (again) the minimal length of a description for x. There is an optimal monotone description mode S such that

$$KM_S(x) \leqslant KM_W(x) + c_W$$

for every monotone description mode W and binary string x. The function KM_S is called *monotone Kolmogorov complexity*. (It is indeed monotone: if x is a prefix of x', then $KM(x) \leqslant KM(x')$.)

1.2 Decision Complexity

Decision complexity was defined by D.W. Loveland [4]. As before, we reformulate the definition in terms of binary relations. (Here description is treated as an isolated binary string while described object is treated as information about an infinite sequence.)

Formally, a *decision description mode* is an enumerable set W of pairs of binary strings such that:

- if $\langle p, x\rangle \in W$ and $x' \preceq x$, then $\langle p, x'\rangle \in W$.
- if $\langle p, x\rangle \in W$ and $\langle p, x'\rangle \in W$, then $x \preceq x'$ or $x' \preceq x$.

If $\langle p, x\rangle \in W$, then p is called a *description for x with respect to W*. The decision complexity $KR_W(x)$ of x is the minimal length of a description for x with respect to W. There is an optimal decision description mode S such that

$$KR_S(x) \leqslant KR_W(x) + c_W$$

for every decision description mode W and binary string x. $KR_S(x)$ is called *decision Kolmogorov complexity*.

The notions of monotone complexity and decision complexity can be naturally generalized to tuples of strings. (Monotone complexity for tuples was considered also by H. Takahashi, cf. [6].)

2 Monotone Complexity of a Pair

A *monotone description mode for pairs* is a pair of enumerable sets W_1 and W_2; each of them is a monotone description mode (as defined earlier).

The monotone complexity $KM_{W_1,W_2}(x, y)$ of a pair of binary strings x and y is the minimal length of a string p such that $\langle p, x\rangle \in W_1$ and $\langle p, y\rangle \in W_2$ (i.e., p describes x with respect to W_1 and describes y with respect to W_2). There is an optimal monotone description mode for pairs and we can define monotone complexity of a pair, denoted by $KM(x, y)$.

Monotone complexity of pairs is a monotone function: $x \preceq x'$ and $y \preceq y'$ implies $KM(x, y) \leqslant KM(x', y')$. Monotone complexity of pairs $\langle x, x\rangle$, $\langle x, \Lambda\rangle$ and $\langle \Lambda, x\rangle$ (here Λ stands for an empty string) equals $KM(x)$ (up to $O(1)$ additive term).

Monotone complexity of a string x is bounded by its length:

$$KM(x) \leqslant |x| + c$$

(for some c and all x). It is easy to prove that monotone complexity of a pair $\langle x, y\rangle$ is bounded by sum of lengths of strings x and y with additional logarithmic term. For every $\alpha > 1$ we have

$$KM(x, y) \leqslant |x| + |y| + \alpha \log(|x| + |y|) + O(1).$$

(all the logarithms have base 2). Indeed, a pair $\langle x, y\rangle$ can be (monotonically) described by the concatenation of a self-delimited code for x (of size $|x|+\alpha \log |x|$) and string y. The following theorem shows that this bound cannot be significantly improved.

Theorem 1. *For every $\alpha < 1$ and every $c \in \mathbb{N}$ there exists a pair of binary strings $\langle x, y\rangle$ such that*

$$KM(x, y) > |x| + |y| + \alpha \log(|x| + |y|) + c.$$

Proof. We fix some universal monotone description mode W of pairs. By way of contradiction, let us suppose the inequality

$$KM(x, y) \leqslant |x| + |y| + \alpha \log(|x| + |y|) + c$$

holds for some $\alpha < 1$, some $c \in \mathbb{N}$ and for all pairs $\langle x, y\rangle$. Then every pair $\langle x, y\rangle$ has description of length $f(|x| + |y|)$ where

$$f(n) = n + \lfloor \alpha \log n \rfloor + c$$

(and $f(0) = c$). (Note that if p is a description for a string x, then every $p' \succeq p$ is also description for x).

We get the desired contradiction by counting how many objects can a description serve and how many descriptions and objects we have. First of all, note that we have about $n2^n$ pairs where sum of lengths is n but only 2^n descriptions of length n. This is not enough for us, because the same string can be a description of many pairs: if p is a description of some pair $\langle x, y\rangle$ with long x and y, it is a description of all pairs $\langle x', y'\rangle$ where $x' \preceq x$ and $y' \preceq y$, and $n + 1$ pairs among them have $|x'| + |y'| = n$. So we get the same factor n here as before. The crucial observation is that if some short p is a description of a pair $\langle x, y\rangle$ with long x and y, then all extensions of p describe the same pair and therefore we waste a lot of descriptions. To make this argument formal, we need to consider at the same time descriptions of different lengths.

It is done in the following way. Let S be a set of binary strings. We define the *gain* of the set S, denoted by $G(S)$, as follows: each pair $\langle x, y\rangle$ that has a description p in S with $|p| = f(|x| + |y|)$, adds $2^{-(|x|+|y|)}$ to the gain.

$$G(S) = \sum_{\langle x,y\rangle \text{ has a description } p \text{ in } S \text{ with } |p| = f(|x|+|y|)} 2^{-(|x|+|y|)}.$$

Let S_n be a set of all strings of length at most $f(n)$. By assumption, S_n contains descriptions of length $f(k)$ for all pairs $\langle x, y\rangle$ such that $k = |x| + |y| \leqslant n$. Therefore the gain of S_n is at least

$$\sum_{k \leqslant n} (k + 1) \simeq n^2/2$$

At the other hand, we prove the following lemma (and get the desired contradiction):

Lemma. *The gain of the set of all strings of length at most $f(n)$ does not exceed $O(n^{1+\alpha})$.*

Proof. Let p be a string of length $f(l)$ for some $l \leqslant n$. We prove the following upper bound on the gain of set $S_{p,n}$ of all binary strings of length at most $f(n)$ that have prefix p:

$$2^{f(l)} G(S_{p,n}) \leqslant (n+1)2^{f(n)-n} + \sum_{k=l}^{n-1} 2^{f(k)-k+1} + \sum_{k=\lceil (n-1)/2 \rceil}^{n-1} (2k+1-n)2^{f(k)-n} \quad (1)$$

(Note that in the last term the factor $(2k + 1 - n)$ is non-negative if and only if $k \geqslant \lceil (n - 1)/2 \rceil$.) Using the fact that $f(k)$ is less than $k + \alpha \log(k) + c$ we get an upper bound for the gain of $S_{b,n}$ when $|b| = c$ (we let $l = 0$):

$$2^c G(S_{b,n}) \leqslant 2^c(n+1)n^\alpha + 2^c \cdot 2 \cdot \sum_{k=0}^{n-1} k^\alpha + 2^c \cdot 2 \cdot \sum_{k=\lceil (n-1)/2 \rceil}^{n-1} (2k+1-n)2^{k-n}k^\alpha$$

The left-hand side is an upper bound for $G(S_n)$ since the gain is achieved on strings of size b or more, and all three terms in the right hand side (both sums and the additional term $2^c(n+1)n^\alpha$) are bounded by $O(n^{1+\alpha})$ values.

It remains to prove the inequality (1) by a backward induction on the length of string p. There are $2^{f(k)}$ different subsets $S_{p,n}$ with $|p| = f(k)$, and our bound is valid for each of them. The right side of the inequality (1) depends only on the length of p.

Induction base ($l = n$). For strings p with $|p| = f(n)$ we need to show that the following inequality holds:

$$G(S_{p,n}) \leqslant (n+1)2^{-n}$$

Indeed, the set $S_{p,n}$ consists of only one string of length $f(n)$ that can be a description for $n+1$ (or less) pairs $\langle x, y\rangle$ with $|x| + |y| = n$, and the pairs with smaller sum of lengths do not give any gain.

Induction step. Suppose the inequality (1) is valid for all sets $S_{p',n}$ with $|p'| = f(l)$ and $l > k$. We will prove the bound on the gain $G(S_{p,n})$ with $|p| = f(k)$. At first we consider a case when $f(k+1) = f(k) + 1$. (It can also happen that $f(k+1) = f(k)+2$, but we consider this case later.) The set $S_{p,n}$ consists of the root p and two subtrees $S_{p0,n}$ and $S_{p1,n}$. A simple bound for $G(S_{p,n})$ is the sum of gains of this subtrees and the root's gain, but it is not enough. We should use the fact that if the root (p) is a description for many pairs of total length k then there should be of lot of pairs (of greater total length) that have descriptions in both subtrees $S_{p0,n}$ and $S_{p1,n}$, and we should take into account descriptions for each of this pairs only once.

There is some maximal pair of binary strings $\langle x, y\rangle$ such that p is a description of $\langle x, y\rangle$ (in the following sense: if p is a description of some other pair $\langle x', y'\rangle$, then $x' \preceq x$ and $y' \preceq y$). The total length $|x| + |y|$ of this pair may be greater than k; in this case p provides gain for several pairs. Let r be a number of those pairs. (Obviously, $0 \leqslant r \leqslant k+1$). Then (by definition) the root p itself provides gain $r2^{-k}$. If $r > 1$, the root p is also a description for at least $r-1$ pairs with total length $k+1$. These pairs are already taken into account in both $G(S_{p0,n})$ and $G(S_{p1,n})$ since they have descriptions of length $f(k+1)$ in these subtrees. Therefore, we may subtract this overlap of size $(r-1)2^{-(k+1)}$ from the sum $G(S_{p0,n}) + G(S_{p1,n})$. Continuing this line of reasoning, we note that the root p is a description for $r-2$ pairs with total length $k+2$, for $r-3$ pairs with total length $k+3$ and so on. We should take into account the overlap for these pairs too. Gains and penalties should be taken only for pairs with total length at most n. Thus we get the following bound on the gain of $S_{p,n}$:

$$G(S_{p,n}) \leqslant G(S_{p0,n}) + G(S_{p1,n}) + r2^{-k} - (r-1)2^{-(k+1)} - \ldots - (r-i)2^{-(k+i)}.$$

Here i is the maximal integer such that $r - i \geqslant 1$ and $k + i \leqslant n$, i.e., $i = \min(r-1, n-k)$. Transforming the right hand side (splitting one term into two), we get

$$G(S_{p,n}) \leqslant G(S_{p0,n}) + G(S_{p1,n}) + 2^{-k} + (r-1)2^{-k} - (r-1)2^{-(k+1)} - \\ - (r-2)2^{-(k+2)} - \ldots - (r-i)2^{-(k+i)},$$

then (combining terms that contain $r-1$)

$$G(S_{p,n}) \leqslant G(S_{p0,n}) + G(S_{p1,n}) + 2^{-k} + (r-1)2^{-(k+1)} - \\ - (r-2)2^{-(k+2)} - \ldots - (r-i)2^{-(k+i)},$$

then (splitting again)

$$G(S_{p,n}) \leqslant G(S_{p0,n}) + G(S_{p1,n}) + 2^{-k} + 2^{-(k+1)} + (r-2)2^{-(k+1)} - \\ - (r-2)2^{-(k+2)} - \ldots - (r-i)2^{-(k+i)},$$

then (combining terms that contain $r-2$)

$$G(S_{p,n}) \leqslant G(S_{p0,n}) + G(S_{p1,n}) + 2^{-k} + 2^{-(k+1)} + (r-2)2^{-(k+2)} - \ldots - (r-i)2^{-(k+i)},$$

and so on until we get

$$G(S_{p,n}) \leqslant G(S_{p0,n}) + G(S_{p1,n}) + 2^{-k} + 2^{-(k+1)} + \ldots + 2^{-(k+i-1)} + (r-i)2^{-(k+i)},$$

Recall that we have two cases: $i = \min(r-1, n-k)$ and minimum can be equal to the first or the second term. If $r - 1 < n - k$, the first term matters. Then $i = r - 1$, so $r - i = 1$ and in the right hand side we have a geometric sequence that can be bounded by twice its first term, i.e.,

$$G(S_{p,n}) \leqslant G(S_{p0,n}) + G(S_{p1,n}) + 2 \cdot 2^{-k}.$$

If $r - 1 \geqslant n - k$, then $i = n - k$ and (in addition to geometric sequence) we have the last term:

$$G(S_{p,n}) \leqslant G(S_{p0,n}) + G(S_{p1,n}) + 2 \cdot 2^{-k} + (r - n + k)2^{-n}.$$

The maximal value for this last term is achieved when r is maximal, i.e., $r = k+1$. So in any case we have the following bound:

$$G(S_{p,n}) \leqslant G(S_{p0,n}) + G(S_{p1,n}) + 2 \cdot 2^{-k} + (2k + 1 - n)2^{-n}.$$

At the end we have the expression:

$$G(S_p) < 2\max(G(S_{p0}), G(S_{p1})) + 2^{-k+1} + \max(0, 2k + 1 - n)2^{-n}$$

Remember, $f(k+1) - f(k)$ is equal to 1 by assumption. Then the inequality may be rewritten as:

$$G(S_p) < 2^{f(k+1)-f(k)} \max(G(S_{p0}), G(S_{p1})) + 2^{-k+1} + \max(0, 2k+1-n)2^{-n} \quad (2)$$

Now we can multiply both sides of inequality (2) by $2^{f(k)}$ and use the induction assumption. The max-operation in the last terms restricts the sum to its non-negative terms, i.e., for $k \geqslant \lceil (n-1)/2 \rceil$.

Now it remains to consider the case $f(k+1) - f(k) = 2$. In this case we can also use the inequality (2) with term $\max(G(S_{p00}), G(S_{p10}), G(S_{p01}), G(S_{p11}))$ instead of $\max(G(S_{p0}), G(S_{p1}))$. (We have the sum of gains for four subtrees, and $2^{f(k+1)-f(k)} = 4$. Note that we do not use the overlap in full: the same pairs are served in all four subtrees, so we could subtract three times more, but it is not needed.) This is enough for our induction argument.

Lemma (and therefore Theorem 1) are proven.

3 Decision Complexity of Triples

A *decision description mode for pairs* is a pair of enumerable sets W_1 and W_2; each of them is a decision description mode. The complexity $KR_{W_1,W_2}(x, y)$ of a pair of binary strings x and y is the minimal length of a string p such that $\langle p, x\rangle \in W_1$ and $\langle p, y\rangle \in W_2$ (i.e., p describes x with respect to W_1 and p describes y with respect to W_2).

There is an optimal decision description mode for pairs and we can define decision complexity of a pair, denoted by $KR(x, y)$. We can also define decision complexity $KR(x, y, z)$ of a triple in the same way (as well as decision complexity of k-tuples for any fixed k).

It is easy to see that decision complexity of a binary string x is bounded by $|x| + O(1)$. Indeed, we may consider the set W of all pairs $\langle p, x\rangle$ where x is a prefix of p. In this case every string x is a description of itself; switching to the optimal description mode, we lose $O(1)$.

It is also easy to prove that decision complexity of a pair $\langle x, y\rangle$ is bounded by $|x| + |y| + O(1)$. Let the set W_1 from our definition be the set of all pairs $\langle p, x\rangle$ where x is a prefix of p. Let the set W_2be the set of all pairs $\langle p, y\rangle$ where y is a prefix of p^R (the reversed string p). Then any pair $\langle x, y\rangle$ has a description xy^R, and its length is $|x| + |y|$.

It turns out (quite unexpectedly) that similar statement is true for triples (though we do not know the simple explanation why it happens and the argument we use looks artificial; we do not know whether it generalizes to k-tuples for $k > 3$):

Theorem 2

$$KR(x, y, z) \leqslant |x| + |y| + |z| + c$$

for some c and all triples $\langle x, y, z\rangle$.

Proof. The statement of Theorem 2 is a consequence of the following combinatorial statement:

Lemma 1. *For every $n \in \mathbb{N}$ there is a set Z_n that contains 2^n triples of n-bit strings $\langle a, b, c\rangle$ such that for every triple $\langle x, y, z\rangle$ with $|x| + |y| + |z| = n$ there exists a triple $\langle a, b, c\rangle \in Z_n$ such that $x \preceq a$, $y \preceq b$ and $z \preceq c$.*

Note that we need at least 2^n elements in Z_n since every triple $\langle a, b, c\rangle$ serves $\binom{n+2}{2}$ triples $\langle x, y, z\rangle$ and there are $2^n\binom{n+2}{2}$ triples $\langle x, y, z\rangle$ with $|x|+|y|+|z| = n$.

Lemma 1 implies the statement of Theorem 2. Indeed, we may assume that Z_n is a computable function of n, and fix some bijection between elements of Z_n and n-bit strings. Then a binary string p of length n that corresponds to a triple $\langle a, b, c\rangle \in Z_n$ is considered as a description for every triple $\langle x, y, z\rangle$ where $x \preceq a$, $y \prec b$, $z \preceq c$. It remains to prove Lemma 1.

Lemma 1 is a simple consequence of the following algebraic statement. Consider for every n a n-dimensional vector space $\mathbb{F}_2^n$ over two-element field $\mathbb{F}_2$.

Lemma 2. *There is a family of $3n$ vectors $a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_n$ in this space such that for every non-negative q, r, t with $q + r + t = n$ the vectors $a_1, \ldots, a_q, b_1, \ldots, b_r, c_1, \ldots, c_t$ are linearly independent.*

In other terms, we have three bases (a_i), (b_i) and (c_i) in our space $\mathbb{F}_2^n$ with additional property: if we take in total n vectors from these bases, and in each basis start from the beginning, we again get a basis in $\mathbb{F}_2^n$.

Let us show how Lemma 2 implies Lemma 1 (and therefore the statement of Theorem 2). There are 2^n different linear functions on $\mathbb{F}_2^n \to \mathbb{F}_2$. For each linear function f we construct a triple of binary strings $\langle a, b, c\rangle$ (and these triples form Z_n):

$$a = f(a_1)\ldots f(a_n), \quad b = f(b_1)\ldots f(b_n), \quad c = f(c_1)\ldots f(c_n).$$

So we get 2^n triples. For any triple of binary strings $\langle x, y, z\rangle$ such that $|x| + |y| + |z| = n$, consider $q = |x|$, $r = |y|$, and $t = |z|$. Since $a_1, \ldots, a_q, b_1, \ldots, b_r, c_1, \ldots, c_t$ are independent, there exists a linear function that has values $x_1, \ldots, x_q$ on $a_1, \ldots, a_1$, has values $y_1, \ldots, y_r$ on $b_1, \ldots, b_r$ and $z_1, \ldots, z_t$ on $c_1, \ldots, c_t$. It remains to prove Lemma 2.

Proof of lemma 2. We will construct the required family by induction over n. The induction step moves us from n to $n + 3$, so we need to consider the cases $n = 1, 2, 3$ for a base.

Induction base. For $n = 1$, the space has dimension 1 and three vectors are a, a, a where a is the only nonzero vector in the space.

For $n = 2$ consider the basis e, f in our (2-dimensional) space; six vectors could be, for example

$$\begin{array}{cc} e & f \\ f & e \\ e+f & e \end{array}$$

(the first row is a_1, a_2, the second is b_1, b_2, the third is c_1, c_2). Each row is evidently a basis, and if we take any two vectors from the first column, we also get a basis.

Finally, for $n = 3$ we take a basis e, f, g in three-dimensional space and consider vectors

$$\begin{array}{ccc} e & f & g \\ g & f & e \\ f+e+g & f & e. \end{array}$$

Each row is evidently a basis; first column is a basis. If we take two first vectors of any row and complement them by a first element of some other row, we again get a basis. (Note that we can check either the linear independence or the fact that chosen vectors span the entire space.)

Induction step. By induction assumption, we have $3k$ vectors

$$\begin{array}{l} a_1\ a_2\ \ldots\ a_k \\ b_1\ b_2\ \ldots\ b_k \\ c_1\ c_2\ \ldots\ c_k \end{array}$$

in a k-dimensional space. Now we add three new dimensions and corresponding vectors a, b, c (that complement any basis in k-dimensional space giving a basis in $(k+3)$-dimensional space). We need to construct $3k+3$ vectors in this extended space. It can be done as follows:

$$\begin{array}{l} a\ a_1[+c]\ a_2[+c]\ \ldots\ a_k[+c]\ b+c\ c \\ b\ b_1[+a]\ b_2[+a]\ \ldots\ b_k[+a]\ c+a\ a \\ c\ c_1[+b]\ c_2[+b]\ \ldots\ c_k[+b]\ a+b\ b \end{array}$$

(square brackets mean that we either add the term in brackets or not, the choice will be made later).

We need to check the every family of $k+3$ vectors (in each row we choose some vectors starting from the left) is independent. Let us start with simple cases where this can be checked independently of the terms in brackets.

Each row forms a basis: first vector and two last vectors generate all three vectors a, b, c, after that the square brackets terms do not matter and we use the induction assumption.

If selection involves all three rows, then vectors a, b, and c are there, and the rest of the selection is k vectors taken from old family, so we get a basis (induction assumption).

It remains to consider the case when selection involves exactly two rows, say, two first rows. Then it includes vectors a and b. Therefore, the terms $[+a]$ in the second row do not matter (since we can add b without changing linear independence). There are several possibilities starting from

$$a, b, b_1, b_2, \ldots, b_k, a+c$$

(all the terms except the first one are taken from the second row) and ending with

$$a, a_1[+c], a_2[+c], \ldots, a_k[+c], b+c, b$$

(all the terms, except the last one, are taken from the first row). These two extreme cases are easy (we have a, b, c and vectors from the old basis), but intermediate cases require more attention. Let us start with selection

$$a, a_1[+c], b, b_1, b_2, \ldots, b_k$$

(two vectors from the first row and the rest from the second row). We have here $b_1, \ldots, b_k$ that form a basis in the old space; vector a_1 is a combination of them, so if we add c, we get a basis in the new space (all three new vectors a, b, c are now accessible). Then we move to the selection

$$a, a_1 + c, a_2[+c], b, b_1, \ldots, b_{k-1}.$$

Here (by induction) the vectors $a_1, b_1, \ldots, b_{k-1}$ form a basis, therefore a_2 is a combination of them. Using $a_1 + c$ instead of a_1 in this combination, we may get $a_2 + c$ instead of a_2. If this is the case, we do not add c to a_2 and get a basis in the new space; if we still get a_2, not $a_2 + c$ (this happens if a_1 was not involved in the expression for a_2), we use $a_2 + c$. Then we consider the next selection

$$a, a_1 + c, a_2[+c], a_3[+c], b, b_1, b_2, \ldots, b_{k-2},$$

recall that $a_1, a_2, b_1, \ldots, b_{k-2}$ form a basis, take an expression for a_3 in this basis, look whether c appears if we use $a_1 + c$ and $a_2[+c]$ instead of a_1 and a_2, and so on.

The case when selection does not involve first or second row is similar (circular permutation of rows).

Lemma 2 is proven.

Question: Is an algebraic statement similar to Lemma 2 true for quadruples (or k-tuples) instead of triples? If not, is the combinatorial statement similar to Lemma 1 true? If not, is the decision complexity bounded by the sum of lengths?

Remark: If it is not the case for k-tuple for some fixed k, then get a new proof of Theorem 1 in a weak form saying that $KM(x, y)$ is not bounded by $|x|+|y|+O(1)$. indeed, it is easy to see that if such a bound were true, this would imply similar bound for k-tuples for any k, and this would imply Theorem 2 for any k.

References

1. Gács, P.: On the relation between descriptional complexity and algorithmic probability. In: FOCS 1981. Journal version: Theoretical Computer Science, vol. 22, pp. 71–93 (1983)
2. Levin, L.A.: On the notion of a random sequence. Soviet Math. Dokl. 14, 1413–1416 (1973)
3. Lim, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer, Heidelberg (1997)
4. Loveland, D.W.: A Variant of the Kolmogorov Concept of Complexity. Information and Control 15, 510–526 (1969)
5. Shen, A.: Algorithmic Information Theory and Kolmogorov Complexity. Lecture Notes of An Introductory Course. Uppsala University Technical Report 2000-034 (2000)
6. Takahashi, H.: On a definition of random sequences with respect to conditional probability. Information and Computation 206, 1375–1382 (2008)
7. Shen, A.: Algorithmic variants of the notion of entropy. Soviet Math. Dokl. 29(3), 569–573 (1984)
8. Uspensky, V.A., Shen, A.: Relation between varieties of Kolmogorov complexities. Math. Systems Theory 29(3), 271–292 (1996)
9. Zvonkin, A.K., Levin, L.A.: The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. Russian Math. Surveys 25(6), 83–124 (1970)

Symbolic Models for Single-Conclusion Proof Logics*

Vladimir N. Krupski

Faculty of Mechanics and Mathematics, Lomonosov Moscow State University,
Moscow 119992, Russia

Abstract. Symbolic semantics for logics of proofs based on Mkrtychev models covers the the case of multi-conclusion proof logics. We propose symbolic models for single-conclusion proof logics (*FLP* and its extensions). The corresponding soundness and completeness theorems are proven. The developed symbolic model technique is used to establish the consistency of contexts required for proof internalization. In particular, we construct an extension of *FLP* that enjoys the strong proof internalization property with empty context.

1 Introduction

Propositional logics of proofs (see surveys [6,7]) are justification logics ([8]) with justifications denoting formal proofs. They formalize the invariant properties of arithmetical proof predicates together with computable operations on proofs induced by admissible inference rules. The propositional language is extended by new formulas of the form $t:F$ (is read as "t proves F") where F is a formula and t is a proof term reflecting the proof constructed by means of a fixed set of basic operations on proofs. Typical examples of basic operations are binary "$\cdot$" (application of *modus ponens* rule) and unary "!" (proof checking).

Historically the first one was Logic of Proofs *LP* [2,4]. It corresponds to strictly multi-valued proof predicates. In the sense of *LP* justifications mean finite sets of arithmetical derivations with additional basic operation "+" that implements the union of such sets. But "standard" proof predicates traditionally used in formal arithmetic have the form

"x is (a code of) a derivation and y is (a code of) its last formula"

and are single-valued, i.e. a derivation can justify at most one formula. This case is covered by single-conclusion (functional) proof logic *FLP* and its extensions [10,11,12,13] where the uniqueness of such formula is formalized via Unification axioms (see below, section 3). The approach based on Unification axioms was proposed for the first time in [1] but was applied there to more primitive proof logic language without operations on proofs.

* The research described in this paper was partially supported by grants RFBR 08-01-00399, 06-01-72554 and INTAS 05-1000008-8144.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 276–287, 2010.

The consistency of LP and other justification logics that do not involve unification can be easily established by forgetful translation that replaces all subformulas of the form $t{:}F$ by F. It maps theorems of LP into classical propositional tautologies. Moreover, this translation is the main method to study the consistency of theories that are axiomatized over LP by formulas of the form $c : A$ (declares c to be a sufficient justification for A).

The forgetful translation does not work for FLP and its extensions. It maps Unification axioms into formulas with arbitrary truth tables. The consistency of FLP can be proven by another translation that replaces all subformulas of the form $t{:}F$ by $\bot$. But this method fails to establish the consistency of extensions capable to prove positive facts of the form $t : F$. More thorough model theory should be used for this purpose.

Arithmetical provability interpretations for single-conclusion proof logics were considered in [11,13] where arithmetical soundness, completeness and decidability theorems for these logics were proven. In this paper we propose the symbolic model technique for single-conclusion proof logics. In most cases it is easier to construct a symbolic model than arithmetical one, so it is natural to use the developed technique in semantical studies of single-conclusion proof logics.

The idea to interpret formulas of the proof logic language as statements about some abstract referential data structure comes from [3]. In [15] it was formalized via the definition of symbolic model for LP (M-model, Mkrtychev model) and was used to prove the decidability of this logic. Later the definition was adapted to many other multi-conclusion proof logics in [14,18,19,17] and was generalized for the case of epistemic modal logics with justifications (Kripke-Fitting models) in [9,8,16]. The decidability results and complexity bounds for these logics are based on symbolic semantics. Kripke-Fitting models provide the main semantics for epistemic justification logics. But for single-conclusion proof logics the adequate classes of symbolic models were not found. The straightforward adaptation of definitions from [15] results in the class of single-valued M-models which is too poor. The logic FLP is incomplete with respect to it.

We fill this gap by developing the symbolic model technique for single-conclusion proof logics. The new idea we propose is to distinguish between the proof logic language and the language of a model. The corresponding translation is a substitution that replaces variables of the first language by appropriate expressions of the second one. It is an additional parameter in the definition of a single-valued symbolic interpretation. We prove the completeness of FLP with respect to interpretations of this sort.

The logics under consideration are FLP and its extensions by reference constructions of the form

$$f(t) = \text{“such } \xi \text{ that } t \text{ proves } P(\xi, \overline{\eta})\text{”}.$$

Here $P(\xi, \overline{\eta})$ is a substitutional instance of some pattern formula $P(x, \overline{y})$ and t is a proof term. For given t the list of objects (proofs or sentences) denoted by $\xi, \overline{\eta}$ is unique provided t really proves $P(\xi, \overline{\eta})$ and the proof predicate is single-valued (see [12,13] for restrictions on pattern formulas and further details).

Reference constructions are used as Skolem functions for some limited form of quantification. The propositional logic of proofs with reference constructions FLP_{ref} is still decidable [13], whereas the logics of proofs with quantifiers are not recursively enumerable [5].

For convenience, we extend the symbolic semantics to the language with two particular reference constructions:

$$goal(t) = \text{“the formula that is proven by } t\text{”},$$

$$refl(t) = \text{“such } s \text{ that } t \text{ proves that } s \text{ proves something”}.$$

Here the pattern formulas are p and $x{:}p$ where p is a sentence variable and x is a proof variable. The corresponding soundness theorem is established.

As an example, we consider an application of symbolic model technique to consistency proofs that concern proof internalization. Logics of proofs are able to argue about their own derivations encoded via proof terms. In the case of single-conclusion proof logics the sound usage of this encoding requires additional context (constant specification) that declares parameters of the proof term to denote the (trivial) proofs of axioms involved in the derivation. We establish the consistency of these contexts. Moreover, we construct a decidable, consistent and axiomatically appropriate constant specification. It is a joined context that is sufficient for sound internalization of any derivation.

2 The Language $L(FLP)$

The language $L(FLP)$ of single-conclusion proof logic FLP [11] has two sorts: *proof terms* (Tm) and *formulas* (Fm),

$$Tm ::= x_i \mid (Tm \cdot Tm) \mid (!\,Tm)\,,$$
$$Fm ::= \bot \mid p_i \mid (Fm \to Fm) \mid (Tm : Fm),$$

where $x_0, x_1, \ldots$ and $p_0, p_1, \ldots$ are proof and propositional variables respectively. The following priority ordering is supposed: $!, \cdot, \to, :$. We also use boolean connectives $\neg, \wedge, \vee, \leftrightarrow$, but treat them as shortenings which denote the classical representations of these connectives in the basis $\{\bot, \to\}$.

The members of $Expr = Tm \cup Fm$ will be considered as terms in the signature $\Omega = \{\bot, \to, :, !, \cdot\}$ and will be called *expressions*. In this context *a substitution* is a sort preserving homomorphism of free term algebras of signature Ω, i.e. a function on $Expr$ that maps proof terms into proof terms, formulas into formulas and commutes with symbols from Ω. A substitution σ is *finite*, if the set $Dom(\sigma) = \{v \mid v \text{ is a variable and } v\sigma \neq v\}$ is finite. Otherwise it is called *infinite*. Let

$$Var(\sigma) = \bigcup_{v \in Dom(\sigma)} Var(v\sigma) \cup Dom(\sigma)$$

where $Var(e)$ is the set of all variables that occur in $e \in Expr$.

For $e, e' \in Expr$ we shall write $e' < e$ when e' is a proper subexpression (subterm or subformula) of e and $e' \equiv e$ when expressions e and e' are syntactically equal.

Definition 1. A *conditional unification problem* is a finite set of conditional equalities

$$A_i = B_i \Rightarrow C_i = D_i, \qquad A_i, B_i, C_i, D_i \in Expr, \;\; i = 1, \ldots, n. \tag{1}$$

Definition 2. Let $\sim$ be an equivalence relation on *Expr*. It is called *valid* when the following conditions hold:

(i) $f(e_1, \ldots, e_n) \sim f(e'_1, \ldots, e'_n) \Leftrightarrow e_1 \sim e'_1 \wedge \ldots \wedge e_n \sim e'_n$ where $f \in \Omega$ is a function symbol of arity $n > 0$ and $e_1, \ldots, e_n, e'_1, \ldots, e'_n \in Expr$.
(ii) $f(e_1, \ldots, e_n) \not\sim g(e'_1, \ldots, e'_m)$ for $f \neq g$, $f, g \in \Omega$.
(iii) The subexpression relation induces a strict partial ordering $\prec$ of the quotient set $Expr/_\sim$ where $[e]_\sim \prec [e']_\sim$ iff

$$e \equiv e_0 \sim e'_0 < e_1 \sim e'_1 < \ldots < e_n \sim e'_n \equiv e'$$

for some finite sequence of expressions $e_0, e'_0, e_1, e'_1, \ldots, e_n, e'_n$ and $n > 0$. The partial order $(Expr/_\sim, \prec)$ remains well-founded.

Lemma 1. *Let Expr, Expr′ be free term algebras of signature Ω. Any substitution σ : Expr → Expr′ defines a valid equivalence relation $\sim_\sigma$ on Expr:*

$$e \sim_\sigma e' \Leftrightarrow e\sigma \equiv e'\sigma. \tag{2}$$

Proof. Clearly, $\sim_\sigma$ is an equivalence relation and satisfies (i), (ii). The statement (iii) follows from the fact that $[e]_{\sim_\sigma} \prec [e']_{\sim_\sigma}$ implies $e\sigma < e'\sigma$. □

Definition 3. A valid equivalence relation $\sim$ is called *consistent* with the conditional unification problem (1) iff $A_i \sim B_i$ implies $C_i \sim D_i$ for $i = 1, \ldots, n$. The conditional unification problem is called *unifiable* when it has a consistent equivalence relation. A substitution σ is a *unifier* of (1) if the equivalence relation $\sim_\sigma$ is consistent with (1).

The classical (unconditional) unification is a special case of our definitions when $A_i \equiv B_i$ for all i. In the conditional case the main results of the classical unification theory are also valid. In particular, the following statements are proven in [11,13].

- The unifiability property for conditional unification problems of the form (1) is decidable.
- Any unifiable problem of the form (1) has the least consistent equivalence relation. The latter has the form $\sim_{\sigma_0}$ where σ_0 is a finite substitution that is idempotent ($\sigma_0^2 = \sigma_0$) and conservative (all variables from $Var(\sigma_0)$ occur in (1)). The substitution σ_0 can be computed effectively given A_i, B_i, C_i, D_i, $i = 1, \ldots, n$.
- The substitution σ_0 is the most general unifier of (1) in the following *weak sense*: any substitution σ that unifies (1) has the form $\sigma = \sigma_0\lambda$ for some substitution λ. (Note that not every substitution of the form $\sigma_0\lambda$ must unify (1).)

Definition 4. Let S be a conditional unification problem (1) and $A, B \in Expr$. We shall write $A = B \, mod \, S$ when $A \sim B$ for every valid equivalence relation $\sim$ that is consistent with S.

Lemma 2 ([11]). *The relation $A = B \, mod \, S$ is decidable.*

Proof. The unifiability property of S is decidable. If S is not unifiable then $A = B \, mod \, S$ holds for every $A, B \in Expr$. For unifiable S one should restore the most general unifier σ_0 of S and test the equality $A\sigma_0 \equiv B\sigma_0$. □

3 The Single-Conclusion Proof Logic *FLP*

With a formula of the form $G = \bigwedge_{i=1}^{n} t_i : F_i$ we associate a conditional unification problem:

$$t_i = t_j \Rightarrow F_i = F_j, \quad i, j = 1, \ldots, n. \tag{3}$$

We shall write $A = B \, mod \, G$ when $A = B \, mod \, S$ and S is the conditional unification problem (3).

The single-conclusion (or functional) proof logic *FLP* is defined by the following calculus (see [11]):

(A0) Axioms of the classical propositional logic.
(A1) $t : F \to F$.
(A2) $t : (F \to G) \to (s : F \to t \cdot s : G)$.
(A3) $t : F \to !t : t : F$.
(A4) Unification axioms: $\bigwedge\limits_{i=1}^{n} t_i : F_i \to (A \leftrightarrow B)$ if $A = B \, mod \bigwedge\limits_{i=1}^{n} t_i : F_i$.

Inference rule: (MP) $F \to G$, $F \vdash G$

4 Single-Valued M-Models

Definition 5. Let $S^c = \{S_0, S_1, \ldots\}$ and $P^c = \{a_0, a_1, \ldots\}$ be two disjoint sets of constants, propositional and proof constants respectively. The language $L(S^c, P^c)$ is the same as $L(FLP)$ but with all variables replaced by constants of corresponding sorts:

$$Tm^c ::= a_i \mid (Tm^c \cdot Tm^c) \mid (!\, Tm^c)\,,$$
$$Fm^c ::= \bot \mid S_i \mid (Fm^c \to Fm^c) \mid (Tm^c : Fm^c).$$

Definition 6. An *M-model* (Mkrtychev model, [15]) is a triplet $\langle L(S^c, P^c), \mathcal{E}, v \rangle$. Here $L(S^c, P^c)$ is the language of the model, $v : S^c \to \{0, 1\}$ is a truth assignment for propositional constants and $\mathcal{E} : Tm^c \to 2^{Fm^c}$ is an evidence function. In the model a term $t \in Tm^c$ is accepted as a valid evidence for every formula $F \in \mathcal{E}(t)$. The following closure conditions are supposed:

– *if $F \to G \in \mathcal{E}(t)$ and $F \in \mathcal{E}(s)$ then $G \in \mathcal{E}(t \cdot s)$;*

– *if* $F \in \mathcal{E}(t)$ *then* $t{:}F \in \mathcal{E}(!t)$.

An M-model is called single-valued *if for every term $t \in Tm^c$ the set $\mathcal{E}(t)$ is a singleton or empty.*

For an M-model $M = \langle L(S^c, P^c), \mathcal{E}, v\rangle$ the validity relation $\models$ is defined as follows:

$$\begin{array}{ll} M \not\models \bot; & \\ M \models S_i & \Leftrightarrow v(S_i) = 1; \\ M \models F \to G & \Leftrightarrow M \not\models F \text{ or } M \models G; \\ M \models t{:}F & \Leftrightarrow F \in \mathcal{E}(t) \text{ and } M \models F. \end{array}$$

Definition 7. A *symbolic model* $\langle \sigma, M\rangle$ for the language $L(FLP)$ is a single-valued M-model $M = \langle L(S^c, P^c), \mathcal{E}, v\rangle$ supplied with a translation σ from $L(FLP)$ into $L(S^c, P^c)$. The translation is a substitution $\sigma\colon L(FLP) \to L(S^c, P^c)$ that replaces all variables of the language $L(FLP)$ by expressions of the language $L(S^c, P^c)$. (Note that σ preserves sorts, i.e. $x_i\sigma \in Tm^c$ and $p_i\sigma \in Fm^c$, but we do not require it to be a finite substitution.)

The validity relation for formulas of the language $L(FLP)$ is defined as follows:

$$\langle \sigma, M\rangle \models F \Leftrightarrow M \models F\sigma.$$

Comment. M-models can be considered as symbolic models with trivial translation $\sigma = id$. They provide an adequate semantics for multi-conclusion proof logics [15,14]. Our definition can be rewritten in this form too: the validity relation for a symbolic model $\langle \sigma, M\rangle$ coincides with one for M-model $\langle L(FLP), \mathcal{E}', v'\rangle$ with $\mathcal{E}'(t) = \{F \mid F\sigma \in \mathcal{E}(t\sigma)\}$, $v'(p_i) = v(p_i\sigma)$. In this sense the semantics given by Definition 7 is a special case of the standard semantics for logics of proofs based on Mkrtychev models. We decompose the standard definition in order to separate the models that satisfy Unification axioms.

Theorem 1. *If $FLP \vdash F$ then F is valid in every symbolic model of the language $L(FLP)$.*

Proof. Axioms (A0)–(A3) and the rule (MP) are sound with respect to the class of all M-models (see [15]). Let us prove the soundness of (A4).

Let $\langle \sigma, M\rangle$ be a symbolic model for $L(FLP)$ and $\langle \sigma, M\rangle \models \bigwedge_{i=1}^{n} t_i{:}F_i$. Consider the equivalence relation (2) induced by σ. It is valid by Lemma 1. If $t_i \sim_\sigma t_j$ then both formulas $F_i\sigma$, $F_j\sigma$ belong to $\mathcal{E}(t_i\sigma)$. But $\mathcal{E}(t_i\sigma)$ is a singleton, so $F_i \sim_\sigma F_j$. Thus, $\sim_\sigma$ is consistent with (3) and $A \sim_\sigma B$. The latter implies $\langle \sigma, M\rangle \models A \leftrightarrow B$. □

5 Language Extension

The more expressive but still propositional proof logic language is proposed in [12,13]. It extends the language $L(FLP)$ by reference constructions which are formalized via second order function variables of types $Tm \to Fm$ and $Tm \to Tm$. For example, the operation that extracts a provable formula from its proof

is expressed by a function variable $goal : Tm \to Fm$ and axiomatized (over Unification axioms) by the axiom schema

$$t{:}F \to t{:}goal(t). \tag{4}$$

Another example is the reflection operation $refl : Tm \to Tm$ that restores a proof term for F given a proof term for $t{:}F$. The corresponding axiom schema is

$$t_1{:}t{:}F \to t_1{:}refl(t_1){:}F. \tag{5}$$

Comment. The reason why function variables (not constants) are used comes from Unification axioms. One has to allow the expressions like $goal(t)$ or $refl(t)$ to be unifiable with expressions of the form $f(\overline{u})$ where $f \in \Omega$. Otherwise, when for example $F \equiv (F_1 \to F_2)$ and $goal$ is treated as a constant, the formula

$$t{:}F \wedge t{:}goal(t) \to (\top \leftrightarrow \bot)$$

should be qualified as Unification axiom because F and $goal(t)$ are not unifiable. So (4) implies $\neg(t{:}F)$ for any t and any F with $\to$ as the main connective.

In a single-valued M-model $M = \langle L(S^c, P^c), \mathcal{E}, v\rangle$ function variables $\alpha{:}\ Tm \to Fm$ and $\beta{:}\ Tm \to Tm$ denote functions α^* and β^* that map any term $t \in Tm^c$ of the language $L(S^c, P^c)$ into a formula $\alpha^*(t) \in Fm^c$ and a term $\beta^*(t) \in Tm^c$ respectively. The axioms like (4), (5) may restrict the choice of these functions (when present). The formal definition is as follows.

Definition 8. Let $L(FLP_2)$ be the extension of the language $L(FLP)$ by function variables α_i, β_j where $i \in I$, $j \in J$. Terms (Tm_2) and formulas (Fm_2) of the language are defined by the following grammar:

$$\begin{aligned} Tm_2 &::= x_i \mid (Tm_2 \cdot Tm_2) \mid (!Tm_2) \mid \beta_j(Tm_2)\,, \\ Fm_2 &::= \bot \mid p_i \mid (Fm_2 \to Fm_2) \mid (Tm_2 : Fm_2) \mid \alpha_i(Tm_2). \end{aligned}$$

A *symbolic model* for the language $L(FLP_2)$ is a pair $\langle \sigma, \overline{M}\rangle$ where

$$\overline{M} = \langle M, \{\alpha_i^*\}_{i\in I}, \{\beta_j^*\}_{j\in J}\rangle$$

is a single-valued M-model $M = \langle L(S^c, P^c), \mathcal{E}, v\rangle$ supplied with the values $\{\alpha_i^*\}_{i\in I}, \{\beta_j^*\}_{j\in J}$ for function variables and σ is an interpretation of the language $L(FLP)$ in M in the sense of Definition 7 extended to the language $L(FLP_2)$ by the equalities

$$\alpha_i(t)\,\sigma = \alpha_i^*(t\sigma), \qquad \beta_i(t)\,\sigma = \beta_i^*(t\sigma), \qquad t \in Tm_2. \tag{6}$$

A formula $F \in Fm_2$ is valid in a symbolic model $\langle \sigma, \overline{M}\rangle$ iff $M \models F\sigma$.

Comment. For function variables α_i, β_j the expressions of the form $\alpha_i(t)$ and $\beta_j(t)$ are ordinary first order variables. We simply add these new variables to the language and consider them as new atomic elements of the term algebra

of signature Ω. The extended interpretation σ from Definition 8 is an infinite substitution that replaces the new variables too. Equalities (6) define the expressions that should be substituted for them. The following property of the extended interpretation is the direct consequence of (6):

$$t\sigma \equiv t'\sigma \;\Rightarrow\; \alpha_i(t)\,\sigma \equiv \alpha_i(t')\,\sigma \wedge \beta_j(t)\,\sigma \equiv \beta_j(t')\,\sigma. \tag{7}$$

Definition 9. Let $Expr_2 = Tm_2 \cup Fm_2$ be the set of all expressions of the language $L(FLP_2)$ considered as terms in signature Ω. In particular, the expressions $\alpha_i(t)$, $\beta_j(t)$ are treated as atomic and have no proper subexpressions. An equivalence relation $\sim$ on the set $Expr_2$ is called *valid* if it satisfies the conditions (i),(ii), (iii) from Definition 2 and the following one:

(iv) If $t \sim t'$ then $\alpha_i(t) \sim \alpha_i(t')$ and $\beta_j(t) \sim \beta_j(t')$, $i \in I$, $j \in J$.

The relation $A = B \,mod\, G$ and the set of Unification axioms for the language $L(FLP_2)$ are defined similarly but involve valid equivalence relations in the sense of Definition 9. Both of them are also decidable (see [13]).

Definition 10. The logic FLP_2 in the language $L(FLP_2)$ is defined by the same axiom schemes (A0)-(A4) with methavariables denoting corresponding expressions of the language $L(FLP_2)$. The inference rule is (MP).

Theorem 2. *If $FLP_2 \vdash F$ then F is valid in every symbolic model of the language $L(FLP_2)$.*

Proof. We follow the proof of Theorem 1. What should be updated is the proof of the fact that the equivalence relation $\sim_\sigma$ on $Expr_2$ (see (2)) induced by the extended interpretation σ of a symbolic model $\langle \sigma, \overline{M} \rangle$ for the language $L(FLP_2)$ is valid. The extended interpretation is a substitution, so Lemma 1 can be applied. It proves the properties (i), (ii) and (iii), i.e. the validity of $\sim_\sigma$ in the sense of Definition 2. But (iv) follows from (7), so the relation $\sim_\sigma$ is valid in the sense of Definition 9 too. □

Definition 11. The logic FLP_2^+ in the language $L(FLP_2)$ with two function variables $goal : Tm_2 \to Fm_2$ and $refl : Tm_2 \to Tm_2$ is the extension of FLP_2 by axiom schemes (4) and (5).

Definition 12. An *FLP_2^+-model* is a symbolic model that satisfies (4) and (5). Any symbolic model $\langle \sigma, M \rangle$ for the language $L(FLP)$ can be extended to an FLP_2^+-model by appropriate choice of the functions $goal^*$ and $refl^*$. It is sufficient to define $goal^*(t)$ as F if $M \models t{:}F$ and $refl^*(t)$ as t' if $M \models t{:}t'{:}F$.

Corollary 1. *The logic FLP_2^+ is sound with respect to the class of all FLP_2^+-models.*

6 Completeness

Single-conclusion proof logics are complete with respect to corresponding classes of symbolic models. It follows from the saturation technique used in arithmetical completeness proofs for these logics (see [11,13]). We prove the completeness theorem for *FLP*.

Comment. In the case of multi-conclusion proof logics the consistency proofs are usually based on the standard construction of maximal consistent sets of formulas. This method does not work for single-conclusion proof logics. The problem is to restore the interpretation σ via unification of an infinite set S of conditional equalities encoded by a consistent set of formulas. The consistency implies that any finite subset of S is unifiable which does not guarantee the unifiability of S. Saturation procedures from [11,13] provide the unifiability.

Theorem 3. *Let $F \in Fm$ and $FLP \nvdash F$. There exists a symbolic model $\langle \sigma, M\rangle$ for the language $L(FLP)$ such that $\langle \sigma, M\rangle \not\models F$.*

Proof. The saturation procedure for the logic *FLP* is described in [11]. It transforms an unprovable formula $F \in Fm$ into a saturated triplet $\langle \Gamma, \Delta, \theta\rangle$ where $\Gamma, \Delta \subset Fm$, $\Gamma \cap \Delta = \emptyset$, and $\theta\colon Expr \to Expr$ is a finite idempotent substitution. The following saturation properties hold:

- If $X \to Y \in \Gamma$ then $X \in \Delta$ or $Y \in \Gamma$. If $X \to Y \in \Delta$ then $X \in \Gamma$ and $Y \in \Delta$. If $t{:}X \in \Gamma$ then $X \in \Gamma$.
- If $t : (X \to Y) \in \Gamma$ and $s : X \in \Gamma$ then $t \cdot s : Y \in \Gamma$. If $t : X \in \Gamma$ then $!t{:}t{:}X \in \Gamma$.
- $\Gamma\theta = \Gamma$, $\Delta\theta = \Delta$, $\{\bot, F\theta\} \subseteq \Delta$.
- If $t{:}X \in \Gamma$ and $t{:}Y \in \Gamma$ then $X \equiv Y$.

Let $\langle \Gamma, \Delta, \theta\rangle$ be a saturated triplet for F. With every variable $\xi \notin Dom(\theta)$ we associate a new constant $\overline{\xi}$. For an expression $e \in Expr$ let $\overline{e} = e\lambda$ where λ is a substitution that replaces all variables $\xi \notin Dom(\theta)$ by $\overline{\xi}$. Consider the language $L(S^c, P^c)$ with

$$S^c = \{\overline{p}_i \mid p_i\theta = p_i,\ i \in \omega\}, \qquad P^c = \{\overline{x}_i \mid x_i\theta = x_i,\ i \in \omega\},$$

and an infinite substitution $\sigma = \theta\lambda$. Note that $e\sigma \in Tm^c \cup Fm^c$ for every $e \in Expr$ because θ is idempotent. Let $\mathcal{E}(\overline{t}) = \{\overline{X} \mid t{:}X \in \Gamma\}$ for $\overline{t} \in Tm^c$ and $v(\overline{p}_i) = 1$ iff $p_i \in \Gamma$. It follows from saturation properties that $\mathcal{E}$ is an evidence function in the sense of Definition 6 and $\mathcal{E}(\overline{t})$ contains at most one formula for every $\overline{t} \in Tm^c$. Thus, $\langle \sigma, M\rangle$ where $M = \langle L(S^c, P^c), \mathcal{E}, v\rangle$ is a symbolic model for the language $L(FLP)$.

By induction on $G \in Fm$ we prove that $G \in \Gamma$ implies $M \models \overline{G}$ and $G \in \Delta$ implies $M \not\models \overline{G}$. For example, consider the case $G \equiv t : X$. If $t : X \in \Gamma$ then $X \in \Gamma$ (saturation property), so $M \models \overline{X}$ by induction hypothesis. We also have $\overline{X} \in \mathcal{E}(\overline{t})$ by the definition of $\mathcal{E}$. Thus, $M \models \overline{t{:}X}$. If $t{:}X \in \Delta$ then $t{:}X \notin \Gamma$ and $\overline{X} \notin \mathcal{E}(\overline{t})$ which implies $M \not\models \overline{t{:}X}$. Other cases are treated similarly.

As a consequence we have $M \not\models F\sigma$ because $F\theta \in \Delta$ and $F\sigma \equiv \overline{F\theta}$. □

7 Example: Consistency of Contexts

Logics of proofs are able to argue about their own derivations encoded via proof terms. The possibility to encode derivations by proof terms known as *proof internalization property* (see [6,7]) is an essential property of all proof logic languages. For single-conclusion proof logics FLP, FLP_2 and FLP_2^+ the encoding is is based on the following Lemma 3 in which $\vdash$ denotes the derivability in one of these logics.

Lemma 3. *If* $\vdash F$ *then it is possible to construct a term* $t(c_1,\dots,c_n)$ *depending on fresh variables* $c_1,\dots,c_n$ *such that*

$$c_1{:}A_1,\dots,c_n{:}A_n \vdash t(c_1,\dots,c_n){:}F \tag{8}$$

where $A_1,\dots,A_n$ *is the list of all axioms involved in the derivation of* F*.*

Proof. Consider a derivation $F_1,\dots,F_m$ of F with axioms $A_1,\dots,A_n$. Let $c_1,\dots,c_n$ be distinct variables that do not occur in the derivation. For $k = 1,\dots,m$ define the term t_k as follows. If F_k is an axiom A_i set $t_k = c_i$. Otherwise, the formula F_k is obtained by the rule (MP) from some formulas F_i and F_j with $i,j < k$. Set $t_k = t_i \cdot t_j$. By induction on k using (A2) one can prove that $c_1{:}A_1,\dots,c_n{:}A_n \vdash t_k{:}F_k$, so the term $t = t_m$ satisfies (8). □

Any derivation can be represented by corresponding term $t(c_1,\dots,c_n)$. For sound usage of this representation one should prove that the set of hypotheses in (8) is consistent. It can be done by symbolic model technique (see Proposition 1).

For logics FLP, FLP_2, FLP_2^+ the term *satisfiability* means satisfiability in corresponding classes of symbolic models (models for $L(FLP)$, models for $L(FLP_2)$ or FLP_2^+-models respectively).

Proposition 1. *For logics* FLP, FLP_2, FLP_2^+ *the following holds: if a set of formulas* Γ *is satisfiable,* $F \in \Gamma$ *and* x_i *is a fresh variable that does not occur in* Γ *then the set* $\Gamma \cup \{x_i{:}F\}$ *is also satisfiable.*

Proof. Case FLP. Let $\langle \sigma, M\rangle$ with $M = \langle L(S^c,P^c),\mathcal{E},v\rangle$ be a symbolic model for the language $L(FLP)$ and $M \models G\sigma$ for all $G \in \Gamma$. We extend the language $L(S^c,P^c)$ by new proof constant d.

Let $\mathcal{E}'$ be the minimal evidence function that extends $\mathcal{E}$ to all terms of the language $L(S^c,P^c\cup\{d\})$ and $\mathcal{E}'(d) = \{F\sigma\}$. It can be easily checked that $M' = \langle L(S^c,P^c\cup\{d\}),\mathcal{E}',v\rangle$ is a single-valued M-model. Moreover, $M' \models G\sigma$ for all $G \in \Gamma$ and $M' \models d{:}F\sigma$.

Consider a substitution σ' such that $x_i\sigma' = d$ and $z\sigma' = z\sigma$ for all variables $z \neq x_i$ of the language $L(FLP)$. It is a translation of the proof logic language into the language of the model M' which coincides with σ on formulas from Γ and $(x_i{:}F)\sigma' = d{:}F\sigma$. So, $\langle \sigma', M'\rangle$ is a model for $\Gamma \cup \{x_i{:}F\}$.

Cases FLP_2 and FLP_2^+. In addition to previous steps one has to define the values $\alpha_i^*(t)$, $\beta_j^*(t)$ for terms t of the language $L(S^c,P^c\cup\{d\})$ that contain d. The validity of formulas from $\Gamma \cup \{x_i{:}F\}$ does not depend on these values, so it is

sufficient to satisfy the axioms. In the case of FLP_2 there is no restrictions, so the values $\alpha_i^*(t)$, $\beta_j^*(t)$ for such t's may be any formula and any term, respectively. For FLP_2^+ the values should be chosen in accordance with axioms (4) and (5) (see Definition 12). □

Corollary 2. *In conditions of Lemma 3 the set of hypotheses* $\{c_1 : A_1, \ldots, c_n : A_n\}$ *is consistent.*

Proof. By soundness results and Proposition 1, it is sufficient to prove the satisfiability of the set $\{A_1, \ldots, A_n\}$. But formulas $A_1, \ldots, A_n$ are axioms, so it remains to prove that the classes of symbolic models for the logics FLP, FLP_2 and FLP_2^+ are nonempty.

The class of symbolic models for FLP includes trivial symbolic models with $L(S^c, P^c) = L(FLP)$, $\sigma = id$ and $\mathcal{E}(t) = \emptyset$ for all $t \in Tm$. Any symbolic model for FLP can be extended to an FLP_2^+-model which is also a model for FLP_2. □

Definition 13. Let one of the logics FLP, FLP_2 or FLP_2^+ be fixed. Any set of formulas of the form $c{:}A$ where c is a variable and A is an axiom will be called a *context*. A context Γ is *axiomatically appropriate* (see [9]) if $c{:}A \in \Gamma$ for every axiom A of the logic and some variable c.

Comment. In the case of LP contexts are called *constant specifications*. For single-conclusion proof logics the term "context" is preferable because the languages of these logics have no proof constants.

Let Γ be an axiomatically appropriate context. Consider a theory that extends the logic by all formulas $c{:}A \in \Gamma$ as new axioms. For this theory the internalization property (Lemma 3) holds in a strong form without the hypotheses in (8). But the theory may happen to be inconsistent.

Note that the logic LP has the same form. It is an extension of its logical kernel LP_0 by all formulas of the form $c{:}A$ where A is an axiom of LP_0 and c is a proof constant. The consistency of LP is quite evident and can be proven by forgetful projection. The consistency of theories over single-conclusion proof logics requires more thorough analysis (see Introduction).

Theorem 4. *The logics FLP,* FLP_2 *and* FLP_2^+ *have consistent decidable axiomatically appropriate contexts.*

Proof. Consider one of these logics. Let c_j^i be the variable $x_{2^i 3^j}$, $i, j \in \omega$, and $A_0^i, A_1^i, \ldots$ be the lexicographic ordering of all axioms that do not contain variables c_j^k for $k \geqslant i$. The context $\Gamma = \{c_j^i : A_j^i \mid i, j \in \omega\}$ is decidable and axiomatically appropriate. It follows from Proposition 1 that all finite contexts of the form $\Gamma_k = \{c_j^i : A_j^i \mid i, j < k\}$ for $k \in \omega$ are satisfiable. Thus, all Γ_k are consistent which implies the consistency of $\Gamma = \bigcup_k \Gamma_k$. □

Corollary 3. *The logics FLP,* FLP_2, FLP_2^+ *have consistent recursively axiomatizable extensions which admit proof internalization in the following strong form: if* $\vdash F$ *then* $\vdash t{:}F$ *for some proof term* t.

Proof. Let T be a theory that extends the logic by the context Γ constructed in Theorem 4. T is recursively axiomatizable and consistent. As in Lemma 3, the strong proof internalization property for T can be established by induction on the derivation of F. If F is a logical axiom then $t = c$ where $c{:}F \in \Gamma$. If F is an axiom $c{:}A \in \Gamma$ then $t =!c$. The rule (MP) should be treated as in the proof of Lemma 3. □

References

1. Artemov, S., Straßen, T.: Functionality in the basic logic of proofs. Technical Report IAM 92-004, University of Bern (1993)
2. Artemov, S.: Operational modal logic. Technical Report MSI 95–29, Cornell University (1995)
3. Artemov, S., Krupski, V.: Data storage interpretation of labeled modal logic. Annals of Pure and Applied Logic 78(1-3), 57–71 (1996)
4. Artemov, S.: Explicit provability and constructive semantics. Bulletin of Symbolic Logic 7(1), 1–36 (2001)
5. Artemov, S., Yavorskaya (Sidon), T.: On first order logic of proofs. Moscow Mathemetical Journal 1(4), 475–490 (2001)
6. Artemov, S.N.: Kolmogorov and Gödel's approach to intuitionistic logic: current developements. Russian Mathematical Surveys 59(2), 203–229 (2004)
7. Artemov, S., Beklemishev, L.: Provability logic. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, 2nd edn., vol. 13, pp. 189–360. Springer, Heidelberg (2005)
8. Artemov, S., Nogina, E.: Introducing justification into epistemic logic. Journal of Logic and Computation 15(6), 1059–1073 (2005)
9. Fitting, M.: The logic of proofs, semantically. Annals of Pure and Applied Logic 132(1), 1–25 (2005)
10. Krupski, V.: Operational logic of proofs with functionality condition on proof predicate. In: Adian, S., Nerode, A. (eds.) LFCS 1997. LNCS, vol. 1234, pp. 167–177. Springer, Heidelberg (1997)
11. Krupski, V.: The single-conclusion proof logic and inference rules specification. Annals of Pure and Applied Logic 113(1-3), 181–206 (2001)
12. Krupski, V.: Reference constructions in the single-conclusion proof logic. Journal of Logic and Computation 16(5), 645–661 (2006)
13. Krupski, V.: Referential logic of proofs. Theoretical Computer Science 357(1-3), 143–166 (2006)
14. Kuznets, R.: On the complexity of explicit modal logics. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 371–383. Springer, Heidelberg (2000)
15. Mkrtychev, A.: Models for the Logic of Proofs. In: Adian, S., Nerode, A. (eds.) LFCS 1997. LNCS, vol. 1234, pp. 266–275. Springer, Heidelberg (1997)
16. Rubtsova, N.: On realization of S5-modality by evidence terms. Journal of Logic and Computation 16(5), 671–684 (2006)
17. Rubtsova, N.M.: Logic of Proofs with substitution. Mathematical Notes 82(5-6), 816–826 (2007)
18. Yavorskaya (Sidon), T.: Negative operations on proofs and labels. Journal of Logic and Computation 15(4), 517–537 (2005)
19. Yavorskaya (Sidon), T., Rubtsova, N.: Operations on proofs and labels. Journal of Applied Non-Classical Logics 17(3), 283–316 (2007)

Complexity of Problems Concerning Carefully Synchronizing Words for PFA and Directing Words for NFA

P.V. Martyugin

Ural State University,
620083 Ekaterinburg, Russia
martugin@mail.ru

Abstract. We show that the problem of checking careful synchronizability of partial finite automata and the problem of finding the shortest carefully synchronizing word are PSPACE-complete. We show that the problem of checking D_1, D_2 and D_3-directability of nondeterministic finite automata and the problem of finding the shortest D_1, D_2 and D_3-directing word are PSPACE-complete. The restrictions of these problems to 2-letter automata remain PSPACE-complete.

Keywords: Synchronization, Automata, Directing Words, Computational Complexity.

1 Introduction

A *deterministic finite automaton (DFA)* is a triple $\mathscr{A} = (Q, \Sigma, \delta)$, where Q is a finite set of states, Σ is a finite alphabet and δ is a totally defined transition function. Denote by 2^Q the set of all subsets of the set Q, and by Σ^* the free Σ-generated monoid with the empty word λ. The function δ extends in a natural way to the action $Q \times \Sigma^* \to Q$. This extension is also denoted by δ. A DFA $\mathscr{A} = (Q, \Sigma, \delta)$ is called *synchronizing* if there exists a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves the automaton in one particular state no matter at which state in Q it started: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$. Any word w with this property is said to be a *reset* or *synchronizing* word for the automaton $\mathscr{A}$.

A conjecture proposed by Černý in [1] states that every synchronizing automaton with n states can be synchronized by a word of length at most $(n-1)^2$. There have been many attempts to prove it, but they all have failed so far. The conjecture has been proved only for some special cases of automata. The best known upper bound is $(n^3 - n)/6$ (see [2]). The corresponding lower bound has been proved by Černý [1]. Surveys of results concerning synchronizing words can be found in [4].

It is natural to consider the computational complexity of various problems arising from the study of automata synchronization. The main natural questions are as follows: given an automaton is it synchronizing or not, and what is the

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 288–302, 2010.

length of the shortest synchronizing word for a given automaton? In [3], Eppstein presented an algorithm which checks whether a given DFA $\mathscr{A} = (Q, \Sigma, \delta)$ is synchronizing. This algorithm runs in $O(|\Sigma| \cdot |Q|^2) + |Q|^3$ time. Moreover, for a synchronizing automaton this algorithm finds some synchronizing word although not necessarily the shortest. In [3], it is proved that the following problem is NP-complete. Given a DFA $\mathscr{A}$ and an integer L, is there a word of length $\leq L$ resetting the automaton $\mathscr{A}$? This problem remains NP-complete even if the input automaton has a 2-letter alphabet.

The notion of a synchronizing word can be generalized to the case of DFA with a partial transition function (PFA) and to nondeterministic finite automata (NFA). *A partial finite automaton (PFA)* is a triple $\mathscr{A} = (Q, \Sigma, \delta)$, where Q is a finite set of states, Σ is a finite alphabet and δ is a partial function from $Q \times \Sigma$ to Q (the function δ can be undefined on some pairs from the set $Q \times \Sigma$). The function δ can be naturally extended to $2^Q \times \Sigma^*$ as follows. We put $\delta(q, \lambda) = q$ for every $q \in Q$. Let $q \in Q$, $a \in \Sigma$, $w \in \Sigma^*$. If both states $p = \delta(q, w)$ and $\delta(p, a)$ are defined, then we put $\delta(q, wa) = \delta(p, a)$. If $S \subseteq Q$, $w \in \Sigma^*$ and the values $\delta(q, w)$ are defined for all states $q \in S$, then we put $\delta(S, w) = \{\delta(q, w) | q \in S\}$.

A PFA $\mathscr{A} = (Q, \Sigma, \delta)$is called *carefully synchronizing*, if there is a word $w \in \Sigma^*$ such that the value $\delta(Q, w)$ is defined and $|\delta(Q, w)| = 1$. We say that such a word w is a *carefully synchronizing word (c.s.w.)* for the automaton $\mathscr{A}$. Clearly DFA is a partial case of PFA and in this case any c.s.w. is also synchronizing. Therefore careful synchronization of PFA is a natural generalization of the synchronization of DFA. The notion of careful synchronization was introduced in [6]. This paper is devoted to a generalization of the Černý problem to the case of PFA. Let $c(n)$ be the maximal length of the shortest c.s.w. for carefully synchronizing PFA $\mathscr{A}$ with n states. It follows from [6] and [7] that

$$\Omega(3^{n/3}) \leq c(n) \leq O(n^2 \cdot 4^{n/3}).$$

A *nondeterministic finite automaton (NFA)* is a triple $\mathscr{A} = (Q, \Sigma, \delta)$ such that Q is a finite set of states, Σ is a finite alphabet, and δ is a function from $Q \times \Sigma$ to 2^Q. The function δ can be naturally extended to the set $2^Q \times \Sigma^*$. Let $S \subseteq Q$, $a \in \Sigma$, then we put $\delta(S, a) = \bigcup_{q \in S} \delta(q, a)$. We also put $\delta(S, \lambda) = S$. Let $S \subseteq Q$, $w \in \Sigma^*$, $w = ua$ and the set $\delta(S, u)$ is defined, then we put $\delta(S, w) = \delta(\delta(S, u), a)$. Note, that PFA is a partial case of NFA, but the definitions of the functions $\delta : 2^Q \times \Sigma^* \to 2^Q$ for PFA and NFA are sufficiently different, because these function have different co-domain.

Let $\mathscr{A} = (Q, \Sigma, \delta)$ be an NFA and $w \in \Sigma^*$. The word w is D_1-*directing* if $\delta(q, w) \neq \emptyset$ for all $q \in Q$ and $|\delta(Q, w)| = 1$. The word w is D_2-*directing* if $\delta(q, w) = \delta(Q, w)$ for all $q \in Q$. The word w is D_3-*directing* if $\bigcap_{q \in Q} \delta(q, w) \neq \emptyset$. The NFA $\mathscr{A}$ is called D_1, D_2 or D_3-*directing* if there is a D_1, D_2 or D_3-directing word for it. The D_1 and D_3-directability is a generalization of careful synchronization in the case when an NFA is a PFA and the D_1, D_2 and D_3-directability is a generalization of the synchronization in the case that an NFA is a DFA. The D_1, D_2 and D_3-directing words were studied in [5,7]. Let $d_1(n)$, $d_2(n)$ and $d_3(n)$

be the maximal lengths of the shortest D_1, D_2 or D_3-directing words for NFA with n states respectively. The following lower and upper bounds of the length of the shortest directing words were obtained:

$$2^n - n - 1 \leq d_1(n), d_2(n) \leq O(2^n), \quad \Omega(3^{n/3}) \leq d_3(n) \leq O(n^2 \cdot 4^{n/3}).$$

There are two natural questions. How can we check whether a given PFA (NFA) is carefully synchronizing (D_1, D_2, D_3-directable)? How can we find the shortest carefully synchronizing (D_1, D_2, D_3-directing) word for a given PFA (NFA)?

These problems can be solved using simple algorithms. Let us consider an algorithm of finding a c.s.w. for a PFA (the problems of finding D_1, D_2, D_3-words can be solved using similar algorithms). The algorithm uses the *power automaton* $2^{\mathscr{A}} = (2^Q, \Sigma, \delta)$, where 2^Q is a set of all subsets of Q, and $\delta : 2^Q \times \Sigma \to 2^Q$ is an extension of the function δ. If there is a path from the set Q to some one-element set in the graph of the automaton $2^{\mathscr{A}}$, then there is a c.s.w. for $\mathscr{A}$. This word can be read along the path in the graph. The path in the graph of the automaton $2^{\mathscr{A}}$ can be found using breadth-first search. The breadth-first search finds the shortest c.s.w. Such algorithm uses an exponential time and an exponential memory in the size of PFA $\mathscr{A}$.

The natural questions are as follow: can we solve the problems described before using the memory of polynomial size? What is the complexity of checking careful syncronizability and D_1, D_2, D_3-directability? What is the complexity of finding carefully synchronizing (D_1, D_2, D_3-directing) words? In this paper we prove that all these problems are PSPACE-complete. These problems stated for automata over a k-letter alphabet for $k > 1$ (for instance for 2-letter alphabet) are PSPACE-complete too.

Let us consider one more natural subclass of PFA. A PFA $\mathscr{A} = (Q, \Sigma, \delta)$ is called a *PFA with a zero* if there is a state $z \in Q$ such that for any letter $a \in \Sigma$ it follows $\delta(z, a) = z$. We will show that checking careful syncronizability for PFA with a zero is PSPACE-complete even for 2-letter PFA.

2 Problems

Let us give formal definitions of the problems we have discussed.

Problem: CARSYN (ZERO CARSYN)
Input: A PFA (a PFA with a zero) $\mathscr{A} = (Q, \Sigma, \delta)$.
Question: Is the automaton $\mathscr{A}$ carefully synchronizing?

Problem: D1DIR (D2DIR, D3DIR)
Input: A NFA $\mathscr{A} = (Q, \Sigma, \delta)$.
Question: Is the automaton $\mathscr{A}$ D_1-directing (D_2-directing, D_3-directing)?

Problem: SHORT CARSYN (ZERO CARSYN) WORD
Input: A PFA (a PFA with a zero) $\mathscr{A} = (Q, \Sigma, \delta)$.
Task: To check whether the automaton $\mathscr{A}$ is carefully syncronizing and to find the shortest c.s.w. for the automaton $\mathscr{A}$ if $\mathscr{A}$ is carefully synchronizing.

Problem: SHORT D1DIR (D2DIR, D3DIR) WORD
Input: A NFA $\mathscr{A} = (Q, \Sigma, \delta)$.
Task: To check whether the automaton $\mathscr{A}$ is D_1-directing (D_2-directing, D_3-directing) and to find the shortest D_1-directing (D_2-directing, D_3-directing) word for automaton $\mathscr{A}$ if $\mathscr{A}$ is D_1-directing (D_2-directing, D_3-directing).

If the input of some PROBLEM contains only automata over an alphabet of size $\leq k$ for some fixed k, then we call such a problem k-PROBLEM (for example, k-CARSYN or k-SHORT D1DIR WORD).

We will prove that all the problems defined above are PSPACE-complete. We use polynomial-space reducibility of problems. Let the problem A be polynomial-space reducible to the problem A. We denote this $A \leq_p B$. This means that for any instance α of the problem A there is an instance β of the problem B such that the answer on α is true iff the answer on β is true and the size of β is polynomial in the size of α. The problem B does not have to be a decision problem, but can also be a search problem (like SHORT CARSYN WORD). To find a solution of such a problem includes the problem of determining the existence of the solution. Therefore, the relation $A \leq_p B$ is defined for all problems B defined in this section. The length of fining words can be not polynomial in the size of input automata. Therefore, then we calculate the computational space complexity of some search problem we do not take take in account the size of output. The size of search problem is the size of used memory without output tape. The next proposition contain some trivial relations among the problems.

Proposition 1. *Let PROBLEM* $\in$ {*CARSYN, ZERO CARSYN, D1DIR, D2DIR, D3DIR*}*, and let* $k \geq 2$ *be an integer, then*

1. *ZERO CARSYN* $\leq_p$ *CARSYN, D2DIR*
2. *k-ZERO CARSYN* $\leq_p$ *k-CARSYN, k-D2DIR*
3. *SHORT ZERO CARSYN WORD* $\leq_p$ *SHORT CARSYN (D2DIR) WORD;*
4. *k-SHORT ZERO CARSYN WORD* $\leq_p$ *k-SHORT CARSYN (D2DIR) WORD.*
5. *CARSYN* $\leq_p$ *D1DIR, D3DIR;*
6. *k-CARSYN* $\leq_p$ *k-D1DIR, k-D3DIR;*
7. *SHORT CARSYN WORD* $\leq_p$ *SHORT D1DIR (D3DIR) WORD;*
8. *k-SHORT CARSYN WORD* $\leq_p$ *k-SHORT D1DIR (D3DIR) WORD.*
9. *PROBLEM* $\leq_p$ *SHORT PROBLEM WORD;*
10. *k-PROBLEM* $\leq_p$ *k-SHORT PROBLEM WORD;*
11. *k-PROBLEM* $\leq_p$ $k+1$*-PROBLEM;*
12. *k-SHORT PROBLEM WORD* $\leq_p$ $k+1$*-SHORT PROBLEM WORD;*
13. *k-PROBLEM* $\leq_p$ *PROBLEM;*
14. *k-SHORT PROBLEM WORD* $\leq_p$ *SHORT PROBLEM WORD;*

Proof. See Appendix.

Now, let us formulate the main theorem.

Theorem 1. *For any integer* $k \geq 2$

1. *The problems CARSYN, SHORT CARSYN WORD, k-CARSYN and k-SHORT CARSYN WORD are PSPACE-complete;*
2. *The problems ZERO CARSYN, ZERO SHORT CARSYN WORD, k- ZERO CARSYN and k-SHORT ZERO CARSYN WORD are PSPACE-complete;*
3. *The problems D1DIR, SHORT D1DIR WORD, k-D1DIR and k-SHORT D1DIR WORD are PSPACE-complete;*
4. *The problems D2DIR, SHORT D2DIR WORD, k-D2DIR and k-SHORT D2DIR WORD are PSPACE-complete;*
5. *The problems D3DIR, SHORT D3DIR WORD, k-D3DIR and k-SHORT D3DIR WORD are PSPACE-complete.*

Proof. Our proof will consist of three steps. Firstly we reduce the problem FINITE AUTOMATA INTERSECTION to the problem ZERO CARSYN and prove that the problem CARSYN is PSPACE-hard (Proposition 2). Secondly we reduce the problem ZERO CARSYN to the problem 2-ZERO CARSYN with the help of automata obtained in Proposition 3. This proves that the problem 2-ZERO CARSYN is PSPACE-hard (Proposition 3). Thirdly we show that the problems SHORT D1, D2 and D3DIR WORD belong to the class PSPACE (Proposition 4). Then Proposition 1 gives us PSPACE-completeness of all considered problems.

Now we give some auxiliary definitions and notations. Let $w \in \Sigma^*$. Denote by We let $|w|$ denote the length of the word w. Let $i, j \in \{1, \dots, |w|\}$. We let $w[i]$ denote the i-th letter of the word w. We let $w[i, j]$ denote the word $w[i]w[i+1]\cdots w[j]$. Let n be a natural number. We denote by Σ^n the set of all words of length n over the alphabet Σ.

3 The Problems Are PSPACE-Hard

Let us prove that the problem ZERO CARSYN is PSPACE-hard. We use the classical PSPACE-complete problem FINITE AUTOMATA INTERSECTION (see [8]). We consider deterministic finite automata of the form $\mathscr{A} = (Q, \Sigma, \delta, s, F)$ as *recognizers*, where Q is a set of states, Σ is an alphabet, δ is a totally-defined transition function, $s \in Q$ is an initial state and $F \subseteq Q$ is a set of final states. Let w be a word Σ^*. The automaton $\mathscr{A}$ *accepts* the word w if and only if $\delta(s, w) \in F$.

Problem: FINITE AUTOMATA INTERSECTION
Input: The recognizers $\mathscr{A}_1 = (Q_1, \Sigma, \delta_1, s_1, F_1), \dots, \mathscr{A}_k = (Q_k, \Sigma, \delta_k, s_k, F_k)$, for $k \geq 2$.
Question: Is there a word $w \in \Sigma^*$ such that
$\delta_1(s_1, w) \in F_1, \dots, \delta_k(s_k, w) \in F_k$?

Proposition 2. *The problem ZERO CARSYN is PSPACE-hard.*

Proof. We reduce the problem FINITE AUTOMATA INTERSECTION to the problem ZERO CARSYN. Let recognizers $\mathscr{A}_1 = (Q_1, \Sigma', \delta_1, s_1, F_1), \dots$
$\dots, \mathscr{A}_k = (Q_k, \Sigma', \delta_k, s_k, F_k)$, $k \geq 2$ be an input of the problem FINITE AUTOMATA INTERSECTION. We construct a PFA $\mathscr{B} = (Q, \Sigma, \delta)$ such that there

is a c.s.w. for the automaton $\mathscr{B}$ if an only if the recognizers $\mathscr{A}_1, \ldots, \mathscr{A}_k$ have a common accepting word.

Let $\mathscr{B} = (Q, \Sigma, \delta)$ be a PFA with $Q = Q_1 \cup \ldots \cup Q_k \cup \{beg, end\}$, $\Sigma = \Sigma' \cup \{x, y\}$ and the following transition function. Let $c \in \Sigma'$ and $q \in Q_i$ for some $i \in \{1, \ldots, k\}$. We put

$$\delta(q, c) = \delta_i(q, c),\ \delta(q, x) = s_i,\ \delta(q, y) = \begin{cases} end, & q \in F_i \\ \text{undefined}, & \text{otherwise} \end{cases}$$

$$\delta(end, x) = \delta(end, y) = \delta(end, c) = end.$$

$$\delta(beg, x) = s_1, \delta(beg, y), \delta(beg, c) \text{ are undefined.}$$

The automaton $\mathscr{B}$ is represented by Figure 2 which can be found in Appendix. The state *end* is a zero in the PFA $\mathscr{B}$.

Let $w \in \Sigma'^*$ be a a word accepted by all the recognizers $\mathscr{A}_1, \ldots, \mathscr{A}_k$. Let us prove that the word xwy is a c.s.w. for the PFA $\mathscr{B}$. Indeed, $\delta(Q, x)$ is defined and $\delta(Q, x) = \{s_1, \ldots, s_k, end\}$. We have $\delta_1(s_1, w) \in F_1, \ldots, \delta_k(s_k, w) \in F_k$. Therefore the function $\delta(\bullet, w)$ is defined on the set $\{s_1, \ldots, s_k\}$ and $\delta(\{s_1, \ldots, s_k\}, w) \subseteq F_1 \cup \cdots \cup F_k$. Hence the letter y is defined on the set $\delta(Q, xw) = \delta(\{s_1, \ldots, s_k\} \cup \{end\}, w)$ and $\delta(Q, xwy) = \{end\}$. Thus the word xwy is a c.s.w. for the PFA $\mathscr{B}$.

Lemma 1. *Each shortest c.s.w. $u \in \Sigma^*$ for the automaton $\mathscr{B}$ is of the form $u = xwy$ for some $w \in \Sigma'^*$.*

Proof. See Appendix.

Let $u \in \Sigma^*$ be some shortest c.s.w. for the automaton $\mathscr{B}$. By Lemma 1 we get $u = xwy$ for some $w \in \Sigma'^*$. The letter y is defined on the set $\delta(Q, xw)$. Therefore $\delta(Q, xw) \subseteq F_1 \cup \cdots \cup F_k \cup \{end\}$. This means that $\delta_1(s_1, w) \in F_1, \ldots, \delta_k(s_k, w) \in F_k$ and the word w is a common accepting word for the recognizers $\mathscr{A}_1, \ldots, \mathscr{A}_k$.

We have reduced the problem FINITE AUTOMATA INTERSECTION to the problem ZERO CARSYN. The size of the PFA $\mathscr{B}$ is polynomial in the sum of the sizes of automata $\mathscr{A}_1, \ldots, \mathscr{A}_k$. Thus the problem ZERO CARSYN is PSPACE-hard. The proposition is proved.

A PFA $\mathscr{B}$ is called *simple* if it can be constructed from some recognizers using the procedure from the proof of Proposition 2. Note that the problem ZERO CARSYN is PSPACE-hard not only for the class of arbitrary PFA with a zero but for the class of simple PFA. We denote such problem by SIMPLE CARSYN. It will be used in the proof of the next proposition. Proposition 3 states the PSPACE-hardness for the problem ZERO CARSYN for two-letter automata.

Proposition 3. *The problem 2-ZERO CARSYN is PSPACE-hard.*

Proof. We reduce the problem SIMPLE CARSYN to the problem 2-ZERO CARSYN. Let a PFA $\mathscr{B}$ be an input of the problem SIMPLE CARSYN. This means that $\mathscr{B} = (Q, \Sigma, \delta)$, $Q = \{q_1, \ldots, q_n\}$, $\Sigma = \{c_1, \ldots, c_m, x, y\}$ and there

is a state $end \in Q$ such that every shortest c.s.w. u for PFA $\mathscr{B}$ is equal to xwy for some $w \in \{c_1, \ldots, c_m\}^*$, $\delta(Q, xwy) = end$ and the action of letters $y, c_1, \ldots, c_m$ is undefined on the set Q. We also suppose $end = q_n$. We construct a PFA $\mathscr{C} = (P, \{a, b\}, \gamma)$ with a zero state z such that there is a c.s.w. for the automaton $\mathscr{C}$ if an only if there is a c.s.w. for the automaton $\mathscr{B}$.

Let $\mathscr{C} = (P, \{a, b\}, \gamma)$ be a PFA, $P = \{p_{i,k} | i \in \{1, ..., n-1\}, k \in \{0, ..., m+1\}\} \cup \{z\}$. Denote $c_0 = y$, $c_{m+1} = x$. We also denote $p_{n,1} = \ldots = p_{n,m+1} = z$ Let $i \in \{1, ..., n\}$, $k \in \{0, ..., m+1\}$. Define the function γ.

$$\text{for } j \le m, \gamma(p_{i,k}, a) = p_{i,k+1},\ \gamma(p_{i,m+1}, a) = p_{i,m+1},$$

$$\gamma(p_{i,k}, b) = \begin{cases} p_{t,0}, & \text{if } \delta(q_i, c_k) = q_t \\ \text{undefined}, & \text{if } \delta(q_i, c_k) \text{ is undefined.} \end{cases}$$

The state $z = p_{n,1} = \ldots = p_{n,m+1}$ is a zero state in the automaton $\mathscr{C}$.

Figure 1 represents an example of reduction according to the proofs of Propositions 2 and 3. At first, the set of recognizers $\{\mathscr{A}_1, \mathscr{A}_2\}$ reduces to the corresponding

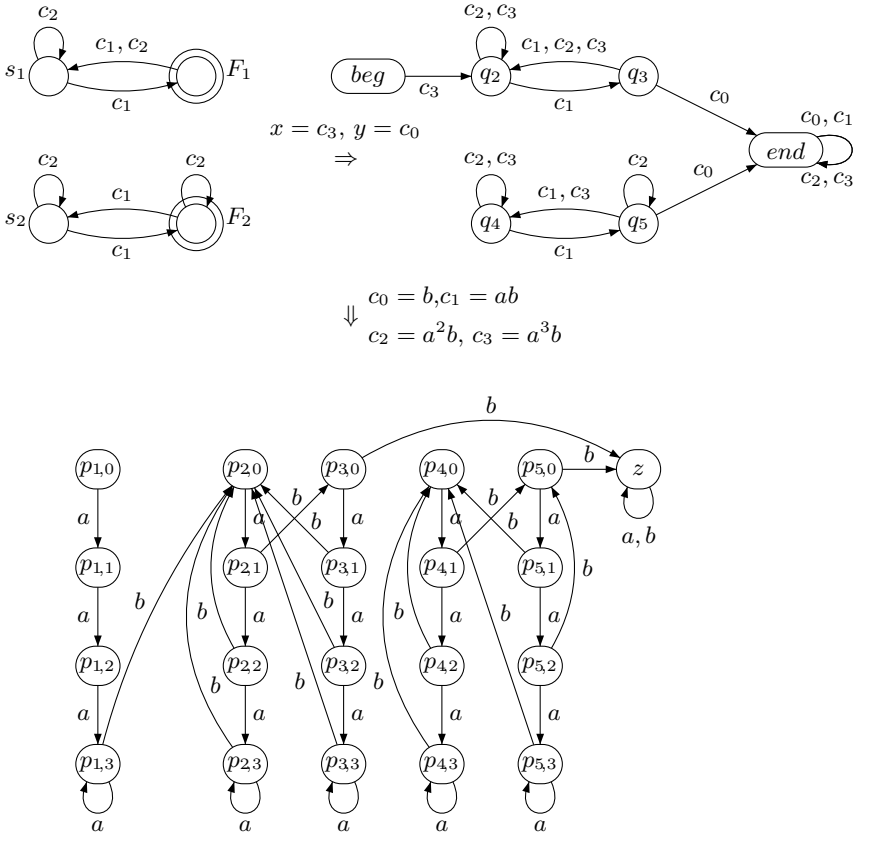


Fig. 1. The example of automata $\mathscr{A}_1, \mathscr{A}_2$, $\mathscr{B}$ and $\mathscr{C}$

PFA $\mathscr{B}$ using the proof of Proposition 2. Then the PFA $\mathscr{B}$ reduces to the corresponding PFA $\mathscr{C}$.

Let $i \in \{1, \ldots, n\}$. We call the set $Q_i = \{p_{i,0}, \ldots, p_{i,m+1}\}$ the *i-th column* of the set P. Let $k \in \{0, \ldots, m+1\}$. We call the set $R_k = \{p_{1,k}, \ldots, p_{n,k}\}$ the *k-th row* of the set P. Note, that n-th column is one state z, and on the over hand, the state z belongs to any row. The following statements are evident corollaries of the definition of the function γ.

Lemma 2. *1.* $\gamma(Q_i, a) \subseteq Q_i$ *for* $i \in \{1, \ldots, n\}$;
2. $\gamma(R_k, a) = R_{k+1}$ *for* $k \in \{0, \ldots, m\}$, $\gamma(R_{m+1}, a) = R_{m+1}$;
3. $\gamma(P, b) \subseteq R_0$.

Let us formulate more complicated properties about the function γ.

Lemma 3. *Let* $v \in \{a, b\}^* \backslash \{a\}^*$ *and the value* $\gamma(P, v)$ *is defined then*

1. for any $i \in \{1, \ldots, n\}$, *we have* $|Q_i \cap \gamma(P, v)| \leq 1$;
2. $|\gamma(P, va)| = |\gamma(P, v)|$;
3. $\gamma(P, vb) \subseteq R_0$.

Proof. See Appendix

Let us consider a morphism $\phi : \{a, b\}^* b \to \Sigma^*$, where $\Sigma = \{c_0, \ldots, c_{m+1}\}$ and the set $\{a, b\}^* b$ is the set of all words over $\{a, b\}$ ended by b. By definition put $\phi(a^k b) = c_k$ for $k \in \{0, \ldots, m+1\}$ and $\phi(a^k b) = c_{m+1}$ for $k \geq m+2$. Every word from the set $\{a, b\}^* b$ is a product of words from the set $\{a^k b | k = 0, 1, 2, ...\}$. Therefore ϕ can be extended to the set $\{a, b\}^* b$. We also consider $\phi^{-1} : \Sigma^* \to \{a, b\}^* b$. Put $\phi^{-1}(c_k) = a^k b$. Of course, the morphism ϕ^{-1} is not a inverse relation to the function ϕ, but ϕ^{-1} is a convenient notation. The morphism ϕ^{-1} also can be extended to Σ^*. Note that for any word $u \in \Sigma^*$ we have $\phi(\phi^{-1}(u)) = u$. We are going to prove that the word u is a c.s.w. for PFA $\mathscr{B}$ if and only if the word $\phi^{-1}(u)$ is a c.s.w. for PFA $\mathscr{C}$.

Let $v \in \{a, b\}^* b$ and the value $\gamma(P, v)$ is defined. By Lemma 2 we get $\gamma(P, b) \subseteq R_0$. Therefore $\gamma(P, v) \subseteq R_0$. We define the value $I(v) = \{i | p_{i,0} \in \gamma(P, v)\}$. Let $u \in \Sigma^*$ and the value $\delta(Q, u)$ is defined. Denote $J(u) = \{i | q_i \in \delta(Q, u)\}$.

Lemma 4. *If* $v \in \{a, b\}^* b$ *and the value* $\gamma(P, v)$ *is defined, then* $I(v) = J(\phi(v))$. *If* $u \in \Sigma^*$ *and the value* $\delta(Q, u)$ *is defined, then* $J(u) = I(\phi^{-1}(u))$.

Proof. Let $v \in \{a, b\}^* b$, then $v = a^{h_1} b a^{h_1} b \ldots a^{h_r} b$ for some integer $h_1, \ldots, h_r$. Let $v_k = a^{h_1} b \ldots a^{h_k} b$ for some $k \in \{1, \ldots, r\}$. The value $\gamma(P, v)$ is defined. This means that the value $\gamma(P, a^{h_1} b) = \gamma(P, v_1)$ is defined too. It is possible only in the case of $h_1 \geq m+1$, because if $h_1 < m+1$ then $p_{m,n} \in \gamma(P, a^{h_1})$ and the letter b is undefined on the set $\gamma(P, a^{h_1})$. Therefore $\phi(v_1) = \phi(a^{h_1} b) = c_{m+1} = x$. It can be easily verified that $I(v_1) = I(a^{h_1} b) = J(c_{m+1})$.

Let $k > 1$. Let $I(v_{k-1}) = J(\phi(v_{k-1}))$. We prove that $I(v_k) = J(\phi(v_k))$. Let $i \in I(v_{k-1}) = J(\phi(v_{k-1}))$. This means that $p_{i,0} \in \gamma(P, v_{k-1})$ and $q_i \in \delta(Q, \phi(v_{k-1}))$. The value $\gamma(p_{i,0}, a^k b)$ is defined. This means that $\gamma(p_{i,0}, a^k b) = p_{m,0}$ for some

$m \in I(v_k)$. Therefore, from the definition of the function γ we obtain that the value $\delta(q_i, \phi(a^{h_k}b))$ is defined and $\delta(q_i, \phi(a^{h_k}b)) = q_m$. Hence $m \in J(\phi(v_k))$ and $I(v_k) \subseteq J(\phi(v_k))$. The proof of the inclusion $I(v_k) \supseteq J(\phi(v_k))$ is analogous. Therefore $I(v_k) = J(\phi(v_k))$. This means that $I(v) = J(\phi(v))$.

The second statement of the lemma can be proved similarly using an induction and the definition of the function γ. The lemma is proved.

Let u be some shortest c.s.w. for PFA $\mathscr{B}$. By Lemma 1 we get $u = xwy$ for some $w \in \Sigma'^*$. Therefore $u = c_{m+1}wc_0 = \phi(a^{m+1}b)w\phi(b)$. Hence for any $k \in \{1, \ldots, |u|\}$ the last letter of the word $\phi^{-1}(u[1,k])$ is b and $\phi^{-1}(u[1,k]) \in \{a,b\}^*b$. By Lemma 4 we obtain $J(u[1,k]) = I(\phi^{-1}(u[1,k]))$. In particular $J(u) = I(\phi^{-1}(u))$. Hence $|I(\phi^{-1}(u))| = |J(u)| = 1$. Therefore the word $\phi^{-1}(u)$ is a c.s.w. for the automaton C.

Let v be some shortest c.s.w. for PFA $\mathscr{C}$. It is easy to see that any word form the set a^* is not carefully synchronizing. Therefore $v \in \{a,b\}^* \backslash \{a\}^*$. By Lemma 3 we obtain $|\gamma(P, v[1, |v|-1]a)| = |\gamma(P, v[1, |v|-1])|$. Therefore $v[|v|] = b$ and $v \in \{a,b\}^*b$. Whence $I(v) = J(\phi(v))$. The word v is a c.s.w. for the automaton $\mathscr{C}$ and hence $|I(v)| = 1$. Therefore $|J(\phi(v))| = 1$. This means that the word $\phi(v)$ is a c.s.w. for the automaton $\mathscr{B}$. The proposition is proved.

4 The Problems Are in PSPACE

Proposition 4. *The problems SHORT D1, D2 and D3DIR WORD belong to the complexity class PSPACE.*

Proof. At first, recall the following property.

Lemma 5. *There is a constant C such that for $D \in \{D_1, D_2, D_3\}$ and for every D-directing NFA with n states the length of the shortest D-directing word is less than $C \cdot 2^n$.*

Proof. The statement follows from [7].

Using Lemma 5 and the Savitch's theorem (which states that PSPACE= NPSPACE) it can be proved that the problems CARSYN, D1DIR, D2DIR and D3DIR are belong to PSPACE, because any carefully synchronizing or directing word can be nondeterministically applied to a given PFA or NFA using $O(n)$ bits of memory. Similar but more complicated ideas can be used to prove that the corresponding search problems are also belong to PSPACE. We do not discuss here such approach because there can be some conceptual problems with a using of Savitch's theorem for a search problem and for a model with out taking in account of the size of output.

Instead of using nondeterministic calculations we introduce here an deterministic searching algorithm. Let $\mathscr{A} = (Q, \Sigma, \delta)$ be an NFA with $Q = \{q_1, \ldots, q_n\}$ and $\Sigma = \{a_1, \ldots, a_k\}$. The automaton $\mathscr{A}$ can be stored using $O(nk)$ bits of memory. We are going to describe an algorithm for searching the shortest D_1, D_2 or D_3-directing words for the automaton $\mathscr{A}$. Our purpose is to construct

an algorithm that uses space that is a polynomial in $O(nk)$. We do not take in account the size of output word. Our algorithm examines different word and finds the first appropriate one. A simple corollary of Lemma 5 is that the length of the shortest directing word can be kept using $O(n)$ bits of memory. The same fact is true for all numbers used in our algorithm.

Our algorithm is a procedure $FindWord(D, \mathscr{A})$ which takes some $D \in \{D_1, D_2, D_3\}$ and some NFA $\mathscr{A}$ as input parameters and returns one of the shortest D_1, D_2 or D_3-directing word letter by letter. The procedure *FindWord* uses the function $GetLetter(L, N)$ where N and L are integers and $N \leq L$. The parameters D and $\mathscr{A}$ are global variables for the function *GetLetter*. This function returns either the N-th letter of a D-directing word of length L for the NFA $\mathscr{A}$ or the value *no* if there is no D-directing word of length L for $\mathscr{A}$. The function calls $GetLetter(L, N_1)$ and $GetLetter(L, N_2)$ return different letters of one word. Let us describe the procedure $FindWord(D, \mathscr{A})$. If the NFA $\mathscr{A}$ is D-directing then the procedure finds the length of the shortest D-directing word and writes this word letter by letter.

```
Procedure FindWord(D, A)
    L = 1
    While L ≤ C · 2^n and GetLetter(L, 1) = no Do
        L = L + 1
    If (L > C · 2^n) Then
        Return "There is no D-directing word"
    Else
        For N = 1 To L Do
            Write GetLetter(L, N)
End Procedure
```

Now we describe the function $GetLetter(L, N)$. Let $\mathscr{A} = (Q, \Sigma, \delta)$ and $Q = \{q_1, \ldots, q_n\}$. A *position* of the NFA $\mathscr{A}$ is an array $P = (P^1, \ldots, P^n)$ where $P^1, \ldots, P^n \subseteq Q$. Let $Pos(\mathscr{A})$ be the set of all positions of the NFA $\mathscr{A}$. The function δ extends to the function acting from $Pos(\mathscr{A}) \times \Sigma$ to $P(\mathscr{A})$. Let $P \in Pos(\mathscr{A})$ and $a \in \Sigma$. We put $\delta(P, a) = (\delta(P^1, a), \ldots, \delta(P^n, a))$. The *starting position* of the NFA $\mathscr{A}$ is the array $S_0 = (\{q_1\}, \ldots, \{q_n\})$. Let $w \in \Sigma^*$. The array $P = (P^1, \ldots, P^n)$ is a D_1-*position* of $\mathscr{A}$ if $P^1 = \cdots = P^n = \{q\}$ for some $q \in Q$. The array P is a D_2-*position* of $\mathscr{A}$ if $P^1 = \cdots = P^n$. The array P is a D_3-*position* of $\mathscr{A}$ if $P^1 \cap \cdots \cap P^n \neq \emptyset$. The position $\delta(S_0, w)$ describes the images of all states of $\mathscr{A}$ under the action of w. The following lemma is a trivial consequence of the definitions of D_1, D_2, D_3-directing words and D_1, D_2, D_3-positions.

Lemma 6. *Let $D \in \{D_1, D_2, D_3\}$. The word $w \in \Sigma^*$ is D-directing for the NFA $\mathscr{A}$ iff $\delta(S_0, w)$ is a D-position.*

Recall $\mathscr{A} = (Q, \Sigma, \delta)$, $|Q| = n$, $|\Sigma| = k$. Let us calculate without proof how much memory is sufficient to keep some objects and to do some operations which we need for the algorithm.

Lemma 7. *1. Every subset $T \subseteq Q$ can be kept using n bits of memory.*

2. *Every position $P \in Pos(\mathscr{A})$ can be kept using n^2 bits of memory.*
3. *Let $P, R \in Pos(\mathscr{A})$ and we need to check whether there is a letter $a \in \Sigma$ such that $\delta(P, a) = R$. It can be done using $O(n)$ of memory.*
4. *For $D \in \{D_1, D_2, D_3\}$ every position $P \in Pos(\mathscr{A})$ can be checked to be a D-position using $O(n)$ of memory.*

Let us describe the function $GetLetter(L, N)$. We denote $S[0] = S_0$.

```
Function GetLetter(L, N)
    m = ⌊log(L)⌋
    P[m + 1] = S[0]
    Return InPath(m, 2^m, L, N)
End Function
```

The function $GetLetter(L, N)$ uses the recursive function $InPath$. The function $InPath$ can return yes, no or some letter $a \in \Sigma$. Define the relation $\leq$ on the set $\Sigma \cup \{yes, no\}$. Put $no < yes < a$ for any $a \in \Sigma$. The sense of every returned letter shall be described later.

Let $D \in \{D_1, D_2, D_3\}$. By Lemma 5 we have the length of the shortest D-directing word for the NFA $\mathscr{A}$ is less than $C \cdot 2^n$ for some constant C. Let $\bar{m} = \lfloor \log(C \cdot 2^n) \rfloor + 1$. Our algorithm uses an additional memory of size at most $\bar{m} \cdot n^2 + O(n^2) = O(n^3)$. We keep at most $\bar{m} + 1$ positions $P[0], \ldots, P[\bar{m}]$ of the automaton $\mathscr{A}$. The NFA $\mathscr{A}$, the directing type $D \in \{D_1, D_2, D_3\}$ and the positions $P[0], \ldots, P[\bar{m}]$ are global variables for the function $InPath$.

```
Function InPath(t, M, L, N)
    res = no
    For Each P[t] ∈ Pos(A)
        If (M ≠ L) or (M = L and P[t] is a D-position)
            If t > 0 Then
                res_1 = InPath(t − 1, M − 2^(m−1), N)
                res_2 = InPath(t − 1, M + 2^(m−1), N)
            If t = 0 Then
                r =the number of the last nonzero bit in the number M − 1
                l =the number of the last nonzero bit in the number M + 1
                If M − 1 ≥ L Then
                    res_1 = yes
                Else
                    If there is a ∈ Σ such that δ(P[r], a) = P[0] Then
                        If M = N Then res_1 = a
                        Else res_1 = yes
                    If there is b ∈ Σ such that δ(P[0], b) = P[l] Then
                        If M = N − 1 Then res_2 = b
                        Else res_1 = yes
            If res_1 > no and res_2 > no and res ≤ yes Then
                        res = max(res, res_1, res_2)
    Return res
End Function
```

Due to space limitations we do not give a detailed proof of the algorithm correctness, but we attempt to describe how the algorithm works in Appendix.

The algorithm $GetLetter(L, N)$ returns the N-th letter of the first found D-directing word or *no* if there is no such word. It follows from Lemma 7 that all operations of the algorithm use polynomial space. It can be calculated that the algorithm uses $O(n^3)$ of additional memory i.e., additional memory of polynomial size. The proposition is proved.

By Proposition 3 we have that the problem 2-ZERO CURSYN is PSPACE-hard. From Proposition 4 we have that the problems SHORT D1, D2 and D3DIR WORD belong to PSPACE. Let $k \geq 2$. From Proposition 1 we have that all problems from the statement of Theorem 1 are PSPACE-complete. Theorem 1 is proved.

Acknowledgement. The author is grateful to Dr. D. Ananichev and Prof. M. Volkov for valuable help. He also acknowledges support from the Federal Education Agency of Russia, project 2.1.1/3537, and from the Russian Foundation for Basic Research, grant 09-01-12142.

References

1. Černý, J.: Poznámka k homogénnym eksperimentom s konecnými avtomatami. Mat.-Fyz. Cas. Slovensk. Akad. Vied. 14, 208–216 (1964) (in Slovak)
2. Pin, J.-E.: On two combinatorial problems arising from automata theory. Ann. Discrete Math. 17, 535–548 (1983)
3. Eppstein, D.: Reset sequences for monotonic automata. SIAM J. Comput. 19, 500–510 (1990)
4. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)
5. Ito, M.: Algebraic Theory of Automata and Languages. World Scientific, Singapore (2004)
6. Martyugin, P.V.: A Lower Bound for the Length of the Shortest Carefully Synchronizing Words. Russian Mathematics (Iz. VUZ) 54(1), 46–54 (2010)
7. Gazdag, Z., Ivan, S., Nagy-Gyorgy, J.: Improved upper bounds on synchronizing non-deterministic automata. Information Processing Letters 109(17), 986–990 (2009)
8. Kozen, D.: Lower bounds for natural proof systems. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pp. 254–266 (1977)

Appendix

The Proof of Proposition 1
1,2,3,4. The class of PFA with a zero is a subclass of the class of all PFA. Therefore ZERO CARSYN $\leq_p$ CARSYN, k-ZERO CARSYN $\leq_p$ k-CARSYN e.t.c. Let $\mathscr{A} = (Q, \Sigma, \delta)$ and the state $z \in Q$ be a zero in $\mathscr{A}$. For any word $w \in \Sigma$ we have $\delta(z, w) = z$ Therefore, the PFA $\mathscr{A}$ can be carefully synchronized only to the zero state z. We can consider the PFA $\mathscr{A}$ as a NFA. If the word $w \in \Sigma^*$ is D_2-directing for the NFA $\mathscr{A}$, then for any state $q \in Q$, we have $\delta(q, w) = \delta(z, w) = z \neq \emptyset$. Hence every D_2-directing word is carefully synchronizing for $\mathscr{A}$. Therefore ZERO CARSYN $\leq_p$ D2DIR, k-ZERO CARSYN $\leq_p$ k-D2DIR e.t.c.

5,6,7,8. Note that if some NFA is a PFA then any D_1 and D_3-directing word is a carefully synchronizing word. Therefore the problem CARSYN is a partial case of the problems D1DIR, D3DIR and the problem SHORT CARSYN WORD is a partial case of the problems SHORT D1 and D3DIR WORD e.t.c.

9,10. Checking the careful synchronizability (careful synchronizability for PFA with zero, D_1, D_2 or D_3 -directability) is a part of the problem SHORT CARSYN (ZERO CARSYN, D1DIR, D2DIR, D3DIR) WORD. Therefore the relations are evident.

11, 12. For any $k > 0$ it follows that $k+1$-PROBLEM is more complicated then k-PROBLEM. To prove this fact we can add one letter with identical action to any k-letter PFA (NFA) and obtain a $k+1$-letter automaton with the same properties of the synchronization.

13, 14. k-PROBLEM is a partial case of PROBLEM for any integer $k > 0$. Therefore, PROBLEM is more complicated then k-PROBLEM for any fixed k and k-PROBLEM $\leq_p$ PROBLEM. The proposition is proved.

The Proof of Lemma 1
Only the letter y can merge some states from Q_1 and Q_2. Therefore $u[|u|] = y$. The word u is defined on the state beg. On the other hand letters from Σ and the letter y are undefined on the state beg. Therefore $u[1] = x$. Hence $u = xwy$.

The image of the letter y contains only one element. Hence if $u[m] = y$ for some $m > 0$, then the word $w[1, m]$ is also a c.s.w. for PFA $\mathscr{B}$ and the word w is not the shortest. On the other hand only the letter y can merge some states from Q_1 and Q_2. Therefore the shortest c.s.w. u for automaton $\mathscr{B}$ contains only once the letter y and $u[|u|] = y$.

We have $u[1] = x$. Hence for any $i \in 1...k$, we have $|\delta(Q, u[1]) \cap Q_i| = 1$. All the letters from Σ and the letter x are defined on the set Q_i and map Q_i to Q_i. Therefore for any $m \in 1...|u| - 1$ we have $|\delta(Q, u[1, m]) \cap Q_i| = 1$. Hence if $u[m] = x$ then $\delta(Q, u[1, m]) = \{s_1, \ldots, s_k\} = \delta(Q, u[1])$ and the word $u[1]u[m+1, |u|]$ is also a c.s.w. This means that there is the only one letter x in the shortest c.s.w. u and $u[1] = x$. Thus $u = xwy$ for some $w \in \Sigma'^*$. The lemma is proved.

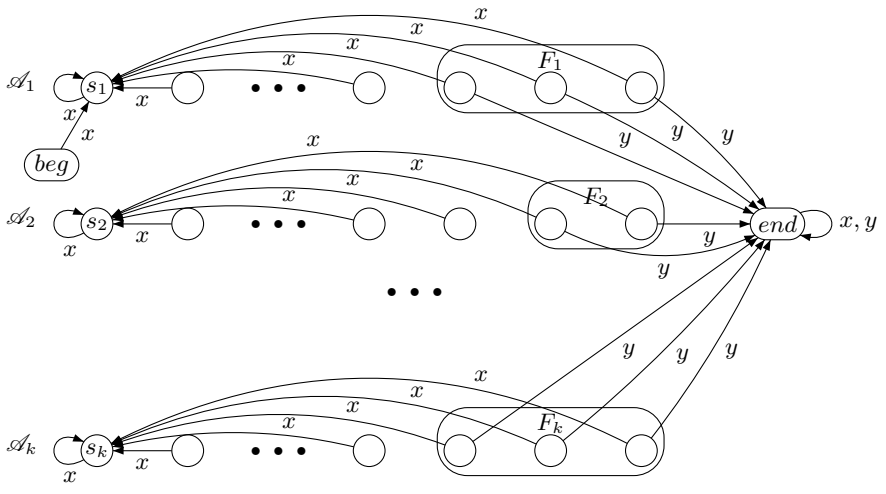


Fig. 2. Automaton $\mathscr{B}$

The Proof of Lemma 3

1. We have $v \in \{a,b\}^*\backslash\{a\}^*$. Therefore a letter b occurs in the word v. Let $v[h_1]$ be the first occurrence of the letter b in the word v. By Lemma 2 obtain $\gamma(P,b) \subseteq R_0$. Therefore $\gamma(P, v[1,h_1]) \subseteq R_0$. Therefore $|Q_i \cap \gamma(P, v[1,h_1])| \leq 1$. Let for some $j \in \{1,\ldots,|v|-1\}$ it holds $|Q_i \cap \gamma(P, v[1,j])| \leq 1$. If $v[j+1] = a$ then from Lemma 2 we have for any $i \in \{1,\ldots,n\}$ $\gamma(Q_i, a) \subseteq Q_i$. Therefore $|\gamma(P, v[1,j]a) \cap Q_i| \subseteq |\gamma(P, v[1,j]) \cap Q_i| \leq 1$. If $v[j+1] = b$ then $Q_i \cap \gamma(P, v[j+1]) \subseteq R_0$. This means $|Q_i \cap \gamma(P, v[j+1])| \leq 1$. Whence $|Q_i \cap \gamma(P,v)| \leq 1$.

2. Let $i \in \{1,\ldots,n\}$. We have $|Q_i \cap \gamma(P,v)| \leq 1$. By Lemma 2 we obtain $\gamma(Q_i, a) \subseteq Q_i$ for $i \in \{1,\ldots,n\}$. The letter a is defined on every state from the set P. Therefore if $|Q_i \cap \gamma(P,v)| = 1$, then $|Q_i \cap \gamma(P,va)| = 1$. Therefore

$$|\gamma(P,v)| = \sum_{i=1}^{n} |\gamma(P,v) \cap Q_i| = \sum_{i=1}^{n} |\gamma(P,va) \cap Q_i| = |\gamma(P,va)|.$$

3. By Lemma 2 we obtain $\gamma(P,b) \subseteq R_0$. Therefore $\gamma(P,vb) \subseteq \gamma(P,b) \subseteq R_0$. The lemma is proved.

The Description of the Function $InPath$**.**

Our goal is to find the set of positions $\{S[0],\ldots,S[L]\}$ such that there is a word $w \in \Sigma^*$ such that for $i \in \{1,\ldots,L\}$, we have $\delta(S[i-1], w[i]) = S[i]$ and $S[L]$ is a D-position. In such case the word w is a D-directing word for the NFA $\mathscr{A}$. Every position from the set $\{S[0],\ldots,S[L]\}$ will appear as a value of the one of the variables $P[0],\ldots,P[m+1]$. The position $P[m+1]$ is always equal to $S[0]$. The position $S[2^m]$ appears as a value of the variable $P[m]$. The positions $S[2^m - 2^{m-1}]$ and $S[2^m + 2^{m-1}]$ can appear as a values of $P[m-1]$.

The positions $S[2^m - 2^{m-1} - 2^{m-2}]$, $S[2^m - 2^{m-1} + 2^{m-2}]$, $S[2^m + 2^{m-1} - 2^{m-2}]$ and $S[2^m + 2^{m-1} + 2^{m-2}]$ can appear as values of $P[m-2]$ and so on. If there are exactly t zero bits at the end of the binary record of some number $M \in \{0, \ldots, L\}$, then the position $S[M]$ can appear as a value of $P[t]$. The length of the shortest D-directing word for $\mathscr{A}$ is L, therefore all the found sets $S[1], \ldots, S[L]$ are different.

When we call $InPath(s, M, L, N)$ we search a position $S[M]$. If $M = L$ then algorithm searches only D-positions because the last position should be a D-position. The number M has exactly t zero bits at the end of its binary record. We look over all the positions from $Pos(\mathscr{A})$ and put each of them into the variable $P[t]$. Let M be even. This means $t > 0$. Firstly, for any set $P[t]$ (which can be $S[M]$) we want to find a position $P[t-1]$ which can be $S[M - 2^{m-1}]$. If $P[t] = S[M]$ and $P[t-1] = S[M - 2^{m-1}]$ then there is a word v of length 2^{m-1} such that $\delta(P[t-1], v) = P[t]$. Secondly, we want to find a position $P[t-1]$ which can be $S[M + 2^{m-1}]$. If $P[t] = S[M]$ and $P[t-1] = S[M + 2^{m-1}]$ then there is a word v of length 2^{m-1} such that $\delta(P[t], v) = P[t-1]$. The function $InPath$ calls itself recursively to find the positions $S[M - 2^{m-1}]$ and $S[M + 2^{m-1}]$. If $L > M + 2^{m-1}$ then the position $S[M + 2^{m-1}]$ shell not be searched, but the algorithm shell search sets $S[M], \ldots, S[L]$ anyway.

If the sets $S[M - 2^{m-1}]$ and $S[M + 2^{m-1}]$ (or $S[L]$) are not found then the function $InPath(s, M, L, N)$ returns *no*. If the sets are found, then if $M + 2^{m-1} \geq N > M - 2^{m-1}$, then the function $InPath(s, M, L, N)$ returns the $N - M + 2^{m-1}$-th letter of the first found word v such that $\delta(S[M - 2^{m-1}], v) = S[M + 2^{m-1}]$ (or $\delta(S[M - 2^{m-1}], v) = S[L]$), if not $N \leq M - 2^{m-1}$ or $N > M + 2^{m-1}$ then the function $InPath(s, M, L, N)$ returns *yes*.

If M is odd, then $t = 0$. The variables r and l are the numbers of the last nonzero bit in the numbers $M - 1$ and $M + 1$. In this case the position $P[r]$ is suspected to be the position $S[M-1]$ and the position $P[l]$ is suspected to be the position $S[M+1]$. The variables $P[r]$ and $P[l]$ already contain some positions. The algorithm looks over all positions, puts each of them to a variable $P[0]$ and checks whether there are letters $a, b \in \Sigma$ such that $\delta(P[r], a) = P[0]$ and $\delta(P[0], a) = P[l]$. If the letters are found then $P[0] = S[M]$ and the function $InPath(s, M, L, N)$ returns *yes*, or the letter a if $M = N$, or the letter b if $M = N - 1$.

Advancing Matrix Computations with Randomized Preprocessing

Victor Y. Pan[1,2], Guoliang Qian[2], and Ai-Long Zheng[2]

[1] Department of Mathematics and Computer Science
Lehman College of the City University of New York
Bronx, NY 10468 USA
victor.pan@lehman.cuny.edu
http://comet.lehman.cuny.edu/vpan/
[2] Ph.D. Programs in Mathematics and Computer Science
The Graduate Center of the City University of New York
New York, NY 10036 USA
gqian@gc.cuny.edu, azheng-1999@yahoo.com

Abstract. The known algorithms for linear systems of equations perform significantly slower where the input matrix is ill conditioned, that is lies near a matrix of a smaller rank. The known methods counter this problem only for some important but special input classes, but our novel randomized augmentation techniques serve as a remedy for a typical ill conditioned input and similarly facilitates computations with rank deficient input matrices. The resulting acceleration is dramatic, both in terms of the proved bit-operation cost bounds and the actual CPU time observed in our tests. Our methods can be effectively applied to various other fundamental matrix and polynomial computations as well.

1 Introduction

1.1 Background

The condition number $\operatorname{cond} A = ||A||/\sigma_\rho(A)$ of a matrix A of a rank $\rho > 0$ is the ratio of its largest and smallest positive singular values [8] and is a fundamental concept of matrix computations. (Here and hereafter $||M||$ denotes the 2-norm of a matrix M.) The value $\sigma_k(A)$ is the distance from the matrix A to a nearest matrix of rank $k-1$ for $k = 1, 2, \dots, \rho$. $\sigma_\rho(A) = 1/||A^{-1}||$ for a nonsingular matrix A.

In the inversion of the matrix A and the solution of a nonsingular linear system of equations $A\mathbf{y} = \mathbf{b}$ the condition number is roughly the coefficient of the magnification of input errors, $\operatorname{cond} A \approx \frac{||\text{INPUT ERROR}||}{||\text{OUTPUT ERROR}||}$ [8]. To support the output precision of p_t bits, one needs the input precision of $p_s \geq p_t + \log_2 \operatorname{cond} A$ bits, and so computations with *ill conditioned* matrices (that is the ones having large condition numbers) require a higher input precision p_s. (Here the concepts "large", "ill" and "well conditioned" are quantified in the context of the computational task and computer environment.) The condition number

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 303–314, 2010.

is also largely responsible for convergence of some popular iterative algorithms such as iterative refinement for solving linear systems of equations [8], which only works where the precision p of the computation exceeds $\log_2 \operatorname{cond} A$.

It is known that random matrices tend to be *well conditioned* [4], [6], [14], that is to have not too large condition numbers, but in computational practice ill conditioned inputs A appear routinely, requiring costly computations with an extended precision p_s. There is a huge literature on preconditioning (cf., e.g., [2]), that is on mappings $A \to C$ such that $\operatorname{cond} C \ll \operatorname{cond} A$ and the solution $\mathbf{y}$ to a linear system $A\mathbf{y} = \mathbf{b}$ is readily expressed via the matrix C. Then the precision p_s is used only in a small fraction of all operations. Generally, however, computing such a mapping is as costly as approximating the inverse, and this limits the power of the known preconditioning methods to the important but special matrix classes.

1.2 Our Improvement

Our alternative *randomized preconditioning* enables desired maps $A \to C$ for a typical ill conditioned input A. To specify our progress assume a nonsingular linear system $A\mathbf{y} = \mathbf{b}$ of n equations. Our solution uses $O^*(I(n)p + n^2 p_t)$ bit-operation for p of the order of $\lceil \log_2 \operatorname{cond} C \rceil$, say $p = 2 \log_2 \operatorname{cond} C$, where $O^*(f)$ means $O((\log n) \log\log n)$, and $I(n)$ has the order of n^3 or $n^{2.376}$. For an ill conditioned input, this means a dramatic decrease from the order of $p_s I(n)$ required in the customary computations, where $p_s = p_t + \log_2 \operatorname{cond} A$ and $p_t \ll \log_2 \operatorname{cond} A$.

Our randomization also enables us to avoid (with a probability near one) running into auxiliary singular linear systems of equations, which is a well known hurdle in Gaussian elimination, particularly poisonous for the important task of solving homogeneous linear systems. The known remedies rely on pivoting and orthogonalization, which take their toll. Orthogonalization techniques such as QR factorization and SVD are more costly, but even pivoting "usually degrades the performance" [8, page 119] by interrupting the stream of arithmetic operations with the foreign operations of comparisons, readily destroys matrix structure and sparseness, and threatens or undermines application of block matrix algorithms. Our techniques handle rank deficient (e.g., square singular) inputs as by-product. This power is no wonder because matrices of full rank are ill conditioned if and only if they lie near rank deficient matrices.

Let us sketch some of our techniques. Assume the challenging problem of computing the null space $N(A)$ of an $m \times n$ matrix A of a rank ρ, $\rho < m \leq n$, that is computing the set of its null vectors $\mathbf{x}$, satisfying the homogeneous linear system $A\mathbf{x} = \mathbf{0}$. In Section 5 we show simple extension to the solution of nonsingular nonhomogeneous linear systems $A\mathbf{y} = \mathbf{b}$. (If $\tilde{\mathbf{y}}$ is a specific solution and if such a system is singular, then every solution can be represented as $\tilde{\mathbf{y}} + \mathbf{x}$ for some $\mathbf{x} \in N(A)$.)

Now write $r = n - \rho$ and apply *northern augmentation* $A \to C = \begin{pmatrix} V \\ A \end{pmatrix}$ for a random or even random structured (e.g., Toeplitz) $r \times n$ matrix V. Then the

matrix C has full rank with a probablity one or close to one. Let $C^{(I)}$ be a left inverse, such that $C^{(I)}C = I$ is the identity matrix, $C^{(I)} = C^{-1}$ for $m = n$. Let B denote the submatrix formed by the first r rows of the matrix $C^{(I)}$. Then one can easily prove [12] that the columns of the matrix B form a basis for the null space of the matrix A.

The above map $A \rightarrow C$ tends neither to decrease nor increase the condition number dramatically, but we achieve preconditioning via *western augmentation* $A \rightarrow C = (B_q, A)$, that is via appending q random and properly scaled columns to the matrix A for a sufficiently large integer q. By combining tools from paper [14], linear algebra, and probability theory, we prove that such an augmentation tends to define a well conditioned matrix C, and this is achieved already for $q = 1$ for a typical ill conditioned input, excluding just the inputs on and near an algebraic variety of a lower dimension. Then we can combine western and northern augmentations to compute the null space of a rank deficient input by working with well conditioned matrices of full rank.

For a smaller integer q the western augmentations always map a structured (e.g., Toeplitz or Hankel) input matrix into a matrix with the same but slightly deteriorated structure. (Namely the displacement rank grows by at most q, cf. [10].) Then our formal support for the power of our techniques can be automatically extended. Practically for a structured input, even where the integer q is small, one should apply *randomized structured augmentation*, allowing fewer random parameters but completely preserving the input structure. In this case our formal support is still valid except for the proof from [4], [6], [14] that random matrices tend to be well conditioned. Its extension to Toeplitz, Hankel and other structured matrices is an open problem (see [12] on some initial progress). According to our tests the extension holds in the Toeplitz and Hankel cases (see Tables 1 and 2), but even without assuming this extension we informally argue that our randomized structured augmentation tends to be preconditioning (see Remarks 1 and 5) and support this claim empirically in [12]. We also apply our techniques to the solution of singular Toeplitz linear systems of n equations with n unknowns and accelerate the customary algorithms by the factor $a(n)$ (in terms of the CPU time) where $a(512) > 20$, $a(1024) > 90$, and $a(2048) > 300$ (see Table 3).

1.3 Extensions

Our tests (the contribution of the second and the third authors) give empirical evidence of the power of our approach, but its present exposition shows just the tip of an iceberg. E.g., Conjugate Gradient algorithms, iterative refinement, and Newton's iteration for matrix inversion are highly effective for well conditioned inputs but lose their power as the condition number grows. Our methods naturally extend this power to ill conditioned inputs. For another example, recall the Rayleigh quotient popular iteration for matrix eigen-solving. Its every step is reduced essentially to the solution of ill conditioned linear system of equations and therefore can be accelerated by our methods.

Other natural applications include computation of the numerical rank of a matrix, approximation of a nearly rank deficient matrix with a matrix of a smaller rank, approximation of a matrix by a nearby Toeplitz-like or Hankel-like matrix, and root-finding for polynomial equations. The first author has advanced in all these directions, but due to the space limitation he leaves the respective results to his Technical Reports and journal papers. Due to the space limitation we also compress our exposition and skip many details, proofs, test results, natural variations, extensions etc., available in [12] and [13].

2 Definitions and Basic Facts

General, Toeplitz and Hankel matrices, nmbs and annihilators
We use and extend the customary definitions of matrix computations in the real and complex fields $\mathbb{R}$ and $\mathbb{C}$ (cf. [8]). I_n or just I denote the $n \times n$ identity matrix. $\mathbf{e}_i$ is its ith column vector, $i = 0, 1, \dots, n-1$. $O_{m,n}$ or just O is the $m \times n$ matrix filled with zeros. A matrix U is unitary or orthonormal if $U^H U = I$. $(B_1, \dots, B_k) = (B_j)_{j=1}^k$ is a $1 \times k$ block matrix with blocks $B_1, \dots, B_k$. $\mathrm{diag}(B_1, \dots, B_k) = \mathrm{diag}(B_j)_{j=1}^k$ is a $k \times k$ block diagonal matrix with diagonal blocks $B_1, \dots, B_k$. A^H is the Hermitian transpose of a matrix $A \in \mathbb{C}^{m \times n}$. range$(A)$ is its range (column span), $\mathrm{nul}\, A = n - \mathrm{rank}\, A$ its nullity, $N(A) = \{\mathbf{y} : A\mathbf{y} = \mathbf{0}\}$ its null space made up of its null vectors $\mathbf{y}$. A matrix H is its *complete annihilator* if $\mathrm{range}\, H = N(A)$ and is a *null matrix basis* if in addition it has full column rank. We use the abbreviations nmb, nmb(A), and ca(A). Hereafter we seek cas and nmbs for matrices $A \in \mathbb{C}^{m \times n}$ where $m \geq n$. For $m > n$ we would recall that $N(A^H A) = N(A)$ and $N(A) = \cap_{i=1}^h N(B_i)$ where $A = \sum_{i=1}^h A_i$, $A_i = (O, B_i^T, O)^T$, $B_i \in \mathbb{C}^{k_i \times n}$, $i = 1, \dots, h$, $\sum_{i=1}^h k_i \geq m$ (also cf. [8, Theorem 12.4.1]). Given a ca(A), we can compute a nmb(A) based on LUP or QR factorization of the matrix A or on the following simple fact.

Fact 1. *Suppose H is a* ca(A). *Then H is a* nmb(A) *if and only if* $\mathrm{nul}\, H = 0$ *and HY is a* nmb(A) *if X is a* ca(H) *and if (X, Y) is a nonsingular matrix.*

$J = J_n = (j_{i,k})_{i,k=0}^{n-1}$ is the reflection matrix, $j_{i,n-1-i} = 1$ for all i, $j_{i,k} = 0$ for $i + k \neq n - 1$. ($J^2 = I$.) Toeplitz matrix $T = (t_{i-j})_{i=0,j=0}^{m-1,n-1}$ (resp. Hankel matrix $H = (h_{i+j})_{i=0,j=0}^{m-1,n-1}$) is defined by its $m + n - 1$ entries, e.g., by its first row and first (resp. last) column. (TJ and JT are Hankel matrices for a Toeplitz matrix T, whereas HJ and JH are Toeplitz matrices for a Hankel matrix H.)

Hereafter "ops" stand for "arithmetic operations". $M(n)$ and $I(n)$ ops suffice to multiply an $n \times n$ matrix by a vector and (if possible) to invert it, respectively. $M(n) = 2n^2 - n$, $I(n) \leq (2/3)n^3 + O(n^2)$, $I(n) \leq c_{\mathrm{huge}} n^{2.376}$ for general matrices; $M(n) < 20n \log n$, $I(n) \leq c_{\mathrm{large}} n \log^2 n$ for Toeplitz and Hankel matrices where c_{huge} and c_{large} are immense and large constants, respectively (cf. [1], [3]).

Matrix norm, SVD, inverses, and condition number
$A = S_A \Sigma_A T_A^H$ is *SVD* or *full SVD* of an $m \times n$ matrix A of a rank ρ if $S_A S_A^H = S_A^H S_A = I_m$, $T_A T_A^H = T_A^H T_A = I_n$, $\Sigma_A = \mathrm{diag}(\widehat{\Sigma}_A, O_{m-\rho,n-\rho})$, and $\widehat{\Sigma}_A =$

$\mathrm{diag}(\sigma_j)_{j=1}^{\rho}$. Here $\sigma_j = \sigma_j(A) = \sigma_j(A^H)$ is the jth largest singular value of a matrix A, $j = 1, \ldots, \rho$. $\sigma_1(A) = ||A|| = ||A^H||$ is the 2-norm of a matrix $A = (a_{i,j})_{i,j=1}^{m,n}$, $\frac{1}{\sqrt{mn}}||A|| \le \max_{i,j=1}^{m,n}|a_{i,j}| \le ||A||$. $\mathrm{cond}\, A = ||A||/\sigma_\rho(A)$.

A matrix $X = A^{(I)}$ is a left (resp. right) inverse of a matrix A if $XA = I$ (resp. $AX = I$). $A^{(I)}$=A^{-1} for a nonsingular matrix A.

Theorem 1. *[13]. Let $A \in \mathbb{C}^{m\times r}$ and $B \in \mathbb{C}^{r\times n}$ and write $r_A = \mathrm{rank}\, A$, $r_B = \mathrm{rank}\, B$, $r_- = \min\{r_A, r_B\}$ and $r_+ = \max\{r_A, r_B\}$. Let $r_+ = r$. (In particular this holds if at least one of the matrices A and B is nonsingular.) Then $\mathrm{rank}(AB) = r_-$, $\sigma_{r_-}(AB) \ge \sigma_{r_A}(A)\sigma_{r_B}(B)$ and $\mathrm{cond}(AB) \le (\mathrm{cond}\, A)\,\mathrm{cond}\, B$.*

Random sampling, random matrices, and conditioning

$|\Delta|$ is the cardinality of a set Δ. *Random sampling* of elements from a set Δ is their selection from this set at random, independently of each other, and under the uniform probability distribution on Δ. A matrix is *random* if its entries are randomly sampled (from a fixed set Δ). The (μ, σ) *Gaussian random variable* is the Gaussian random variable with the mean μ and variance σ^2. $(\mu, \sigma) = (0, 1)$ for the *standard Gaussian random variable*. (μ, σ) (resp. standard) *Gaussian random matrix* or vector is the matrix or vector filled with independent (μ, σ) (resp. standard) Gaussian random variables.

Lemma 1. *[5]. For a set Δ of cardinality $|\Delta|$ (in a fixed ring), suppose a polynomial in m variables has total degree d and is not identically zero on the set Δ^m. Let the values of its variables be randomly sampled from the set Δ. Then the polynomial vanishes with a probability of at most $d/|\Delta|$.*

Corollary 1. *An $m \times n$ matrix with entries sampled at random from a set Δ has full rank with a probability of at least $1 - r/|\Delta|$ for $r = \min\{m, n\}$.*

$F_X(y) = \mathrm{Probability}\{X \le y\}$ for a real random variable X is the *cumulative distribution function (CDF)* of X evaluated at y. $F_A(y) = F_{\sigma_l(A)}(y)$ for an $m \times n$ matrix A and an integer $l = \min\{m, n\}$. $F_{\mu,\sigma}(y) = \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{y}\exp(-\frac{(x-\mu)^2}{2\sigma^2})dx$ is the CDF for a Gaussian random variable with a mean μ and a variance σ^2. $\Phi_{\mu,\sigma}(y) = F_{\mu,\sigma}(\mu + y) - F_{\mu,\sigma}(\mu - y)$ for $y \ge 0$.

Standard Gaussian random matrices are well conditioned with a high probability [4], [6], and even perturbations by such matrices A is expected to make well conditioned any matrix having the norm of the order $||A||$ [14].

Theorem 2. *(See [14, Theorems 3.1, 3.3].) Suppose $M \in \mathbb{R}^{m\times n}$, $\bar{U} \in \mathbb{R}^{m\times m}$, and $\bar{V} \in \mathbb{R}^{n\times n}$, $\bar{U}$ and $\bar{V}$ are orthonormal matrices, $A \in \mathbb{R}^{m\times n}$ is the $(0, \sigma)$ Gaussian random matrix independent of the matrix M, $W = \bar{U}(A + M)\bar{V}$, $l = \min\{m, n\}$, and $y \ge 0$. Then $F_W(y) \le 2.35y\sqrt{l}/\sigma$. If in addition $||M|| \le \sqrt{l}$, then $F_{\mathrm{cond}\, W}(y) \ge 1 - (14.1 + 4.7\sqrt{(2\ln y)/n})n/(y\sigma)$ for all $y \ge 1$.*

Theorem 3. *[13]. Suppose $G \in \mathbb{R}^{q\times m}$, $r_G = \mathrm{rank}\, G$, $H \in \mathbb{R}^{n\times r}$, $r_H = \mathrm{rank}\, H$, and a random matrix $W \in \mathbb{R}^{m\times n}$ has full rank with probability one. Then $F_{GW}(y) \le F_W(y/\sigma_{r_G}(G))$ if $r_G = m$; $F_{WH}(y) \le F_W(y/\sigma_{r_G}(H))$ if $r_H = n$.*

cond(AB) can be arbitrarily large even for $m \times r$ unitary matrices A and B^H if $m > r$. So we cannot merely drop the above assumptions that $r_G = m$ and $r_H = n$, but the next theorem circumvents this obstacle.

Theorem 4. *[13]. Suppose* $G \in \mathbb{R}^{r_G \times m}$, $H \in \mathbb{R}^{n \times r_H}$, $W \in \mathbb{R}^{m \times n}$, $\operatorname{rank} G = r_G < m$, $\operatorname{rank} H = r_H < n$, *and the assumptions of Theorem 2 hold. Then we have (a)* $F_{GW}(y) \le cy\sqrt{l}/(\sigma_{r_G}(G)\sigma)$ *and (b)* $F_{WH}(y) \le cy\sqrt{l}/(\sigma_{r_H}(H)\sigma)$.

3 A nmb of a Matrix via Northern Augmentation

We compute a nmb B of a rank deficient matrix A as a block of a left inverse $C^{(I)}$ where $\operatorname{cond} C$ is expected to be of the order of $\operatorname{cond} A$.

Algorithm 1. Given four integers m, n, ρ and $r = n - \rho$, a matrix $A \in \mathbb{C}^{m \times n}$ of a rank ρ, and a set $\Delta \in \mathbb{C}$, $|\Delta| \gg n \ge m > \rho > 0$, generate a random matrix $V \in \Delta^{r \times n}$ such that $||V|| \approx ||A||$. Output FAILURE and stop if the matrix $C = \begin{pmatrix} V \\ A \end{pmatrix}$ is rank deficient. Otherwise compute a left inverse $C^{(I)}$. Output the submatrix B of its first r rows. ($B = \operatorname{nmb}(A)$.)

Correctness proof. Let $Y = \operatorname{nmb}(A)$. Then $CY = \begin{pmatrix} VY \\ O \end{pmatrix}$, $Y = C^{(I)} \begin{pmatrix} VY \\ O \end{pmatrix}$. So $N(A) = \operatorname{range} Y \subseteq \operatorname{range} B$ and $\operatorname{range} B = N(A)$ for $\operatorname{rank} B = \operatorname{rank} Y = r$.

Lemma 1 implies that Probability$(\operatorname{rank} C < n) \le r/|\Delta|$. Our next theorem bounds $\operatorname{cond} C$.

Theorem 5. *Let the matrix* C *in Algorithm 1 be scaled so that* $||C|| \le 1$. *Let* $\operatorname{rank} C = n$, $\operatorname{rank} A = \rho$, *and so* $\operatorname{rank} V = r$. *Let* $A = S_A \Sigma_A T_A^H$ *(SVD),* $\Sigma_A = \operatorname{diag}(\widehat{\Sigma}_A, O)$, $\widehat{\Sigma}_A = \operatorname{diag}(\sigma_j)_{j=1}^{\rho}$, $\operatorname{diag}(I_r, S_A^H) C T_A = \begin{pmatrix} M \\ O \end{pmatrix}$, $M = \begin{pmatrix} V_0 & V_1 \\ \widehat{\Sigma}_A & O \end{pmatrix}$. *Then* $\operatorname{cond} C \le \frac{3}{\sigma_\rho(A)\sigma_r(V_1)}$.

Now we bound $\sigma_r(V_1) = 1/||V^{-1}||$ provided V is a Gaussian random matrix.

Theorem 6. *Let the assumptions of Theorem 4 hold for* $W = V$, $H = T_A \begin{pmatrix} O \\ I_r \end{pmatrix}$, $l = r_H = r$. *Then* $F_{V_1}(y) \le cy\sqrt{r}/\sigma$.

Corollary 2. *Let the assumptions of Theorems 5 and 6 hold. Then* $F_C(y) \le cy\sqrt{r}/(\sigma_\rho(A)\sigma)$.

Remark 1. In the SVD-free computation of a nmb(A) in Algorithm 1 the matrix V is defined independently of the matrix T_A of the singular vectors. This suggests that the matrix V_1 behaves as random (and thus tends to be well conditioned) even where V is a *weakly random* (e.g., structured) matrix defined by $O(r)$ random parameters.

Remark 2. We can scale the matrices A and V to maximize their norms subject to the bound $||C|| \le 1$. E.g., we can let $||V|| \approx ||A|| \approx 1/\sqrt{2}$ and deduce from Corollary 2 that $\operatorname{cond} C$ has the expected order $\operatorname{cond} A = \sigma_1(A)/\sigma_\rho(A)$.

Remark 3. In Algorithm 1 the rank ρ is given to us, but instead we can search for r as the integer for which $\operatorname{rank} C = n$ and $AC^{(I)}\begin{pmatrix} I_r \\ O \end{pmatrix} = 0$ or as the smallest integer for which $\operatorname{rank} C = n$.

Remark 4. Suppose instead of the matrix A, Algorithm 1 is applied to its approximation $\tilde{A}$ of full rank $m > \rho$ where $||\tilde{A} - A|| \ll \sigma_\rho(A)$, and so $\operatorname{cond} A \ll \operatorname{cond} \tilde{A}$. Now under the assumptions of Corollary 2, suppose $\tilde{C} = \begin{pmatrix} V \\ \tilde{A} \end{pmatrix}$, V is Gaussian random matrix scaled so that $||V|| \approx ||\tilde{A}||$. Then we can expect that $\operatorname{rank} \tilde{C} = n$, the block of the first r rows of a matrix $\tilde{C}^{(I)}$ approximates a $\operatorname{nmb}(A)$, and $\operatorname{cond} \tilde{C} \approx \operatorname{cond} C$ has the order of $\operatorname{cond} A$.

4 Northwestern Augmentation

Algorithm 1 maps a rank deficient input A into a full rank matrix C, with $\operatorname{cond} C$ at about the level of $\operatorname{cond} A$. If this level is too high, we can first apply preconditioning via western augmentation.

Algorithm 2. Assume two integers m and n, $0 < m \le n$, an ill conditioned matrix $A \in \mathbb{C}^{m \times n}$, and a finite set $\Delta \in \mathbb{C}$ of a large cardinality $|\Delta|$.

(a) Recursively generate $m \times k$ matrices $B_k = (\mathbf{b}_i)_{i=1}^k$ for $k = 1, \dots, q$ by sampling their entries at random from the set Δ until you obtain a sufficiently well conditioned matrix (B_q, A) of full rank.

(b) Apply Algorithm 1 to this matrix to compute matrix $\bar{Z} = \begin{pmatrix} Z_0 \\ Z_1 \end{pmatrix} = \operatorname{nmb}((B_q, A))$ where $Z_0 \in \mathbb{C}^{q \times s}$, $Z_1 \in \mathbb{C}^{n \times s}$, and $q \le s \le q + r$, $r = \operatorname{nul} A$.

(c) Reapply Algorithm 2 to compute the nullity $r = \operatorname{nul} Z_0$ and an $s \times r$ matrix $X = \operatorname{nmb}(Z_0)$.

(d) Compute the $n \times r$ matrix $Y = Z_1 X = \operatorname{ca}(A)$ and then a $\operatorname{nmb}(A)$ (cf. Fact 1).

At Stage (a) we can apply a known condition estimator (cf. [8, Section 3.5.4]) or just Algorithm 1. Algorithm 2 combines northern and western augmentations and can be also viewed as an aggregation process [12].

Correctness proof. By the definition of the matrices $\bar{Z}$ and X, we have $B_q Z_0 + AZ_1 = 0$ and $B_q Z_0 X = 0$. Therefore $AY = AZ_1 X = 0$. Conversely, if $A\tilde{\mathbf{y}} = \mathbf{0}$, then $(B_q, A)\begin{pmatrix} \mathbf{0} \\ \tilde{\mathbf{y}} \end{pmatrix} = \mathbf{0}$. It follows that $\bar{Z}\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{0} \\ \tilde{\mathbf{y}} \end{pmatrix}$ for some vector $\tilde{\mathbf{x}}$ because $\bar{Z} = \operatorname{nmb}((B_q, A))$. Therefore $Z_0\tilde{\mathbf{x}} = \mathbf{0}$ and $Z_1\tilde{\mathbf{x}} = \tilde{\mathbf{y}}$. (The proof applies to any $q \ge r$ such that matrix $\operatorname{rank}(B_q, A) = m$, but it is desired to decrease q to yield a matrix $\bar{Z}$ of a smaller size.)

Our next theorem implies that $\text{Probability}(q = r) \ge 1 - r/|\Delta|$, and then we estimate $\operatorname{cond}(B_q, A)$ to show the preconditioning power of Algorithm 2.

Theorem 7. *Let $m \le n$ and $r = m - \operatorname{rank} A$. Then (a) for $k < r$ the matrices (B_k, A) in Algorithm 2 are rank deficient, whereas (b) for $k \ge r$ the matrices B_k and (B_k, A) are rank deficient with a probability of at most $r/|\Delta|$.*

Proof. $\operatorname{rank}(B_k, A) \le \operatorname{rank} B_k + \operatorname{rank} A \le k + \operatorname{rank} A$, implying part (a). Clearly the matrices B_k and (B_k, A) have full rank if $k \ge m - \operatorname{rank} A$ and the entries of the matrix B_k are indeterminates. Now Lemma 1 implies part (b).

Theorem 8. *Suppose the matrix $C = (B_q, A)$ in Algorithm 2 has been scaled so that $||C|| \le 1$. Let $\operatorname{rank} C = m$ and $\operatorname{rank} A = m - q$, so that $\operatorname{rank} B_q = q$. Let $A = S_A \Sigma_A T_A^H$ be a full SVD of the matrix A (where $\Sigma_A = \operatorname{diag}(\widehat{\Sigma}_A, O)$ and the matrix $\widehat{\Sigma}_A$ is nonsingular). Then $\operatorname{cond} C \le \frac{3}{\sigma_{m-q}(A)\sigma_q(\bar{B}_1)}$ provided that*

$$S_A^H C \operatorname{diag}(I_q, T_A) = \begin{pmatrix} \bar{B}_0 & \widehat{\Sigma}_A & O \\ \bar{B}_1 & O & O \end{pmatrix}.$$

The theorem shows that $\operatorname{cond} A$ decreases by the factor $\frac{\sigma_{m-q}(A)}{\sigma_m(A)\sigma_q(\bar{B}_1)}$ in the map $A \to C$. Let us bound the value $\sigma_q(\bar{B}_1)$ provided B_q is Gaussian random matrix.

Theorem 9. *Suppose $\operatorname{rank} A = m - q$ and the assumptions of Theorem 7a hold for $W = B_q$, $G = (O, I_q) S_A^H$ and $r_G = l = q \le m$. Then $F_{\bar{B}_1}(y) \le cy\sqrt{q}/\sigma$.*

Corollary 3. *Let the assumptions of Theorems 8 and 9 hold. Then $F_C(y) \le cy\sqrt{q}/(\sigma_{m-q}(A)\sigma)$.*

Remark 5. We can readily extend Remarks 1–4 to the case of Algorithm 2. In particular to extend Remark 1, recall that in the SVD-free computation of a $\operatorname{nmb}(A)$ in Algorithm 2 the matrix B_q is defined independently of the matrix S_A of the left singular vectors of the matrix A. This suggests that the matrix $\bar{B}_1$ behaves as random (and thus tends to be well conditioned) even where B_q is a *weakly random* (e.g., random structured) matrix defined by $O(r)$ random parameters. Likewise, we can extend Remark 2, that is decrease the above upper bounds on the values $1/\sigma_m(C)$ and $\operatorname{cond} C$ by scaling the matrices A and B_q to maximize their norms subject to the bound $||C|| \le 1$. For $||B_q|| \approx ||A|| \approx 1/\sqrt{2}$, we can combine Theorems 8 and 9 and Corollary 3 and deduce that $\operatorname{cond} C$ is expected to be of the order $\sigma_1(A)/\sigma_{m-q}(A)$.

5 The Solution of a Nonsingular Linear System

We can reduce the solution of a nonsingular linear system $A\mathbf{y} = \mathbf{b}$ to computing a null vector $\mathbf{z} = \begin{pmatrix} 1/\eta \\ \mathbf{y} \end{pmatrix}$ for the matrix $M = (-\eta\mathbf{b}, A)$ and a nonzero scalar η.

Claim. Suppose $C = (-\mathbf{b}, A)$, $A = S_A \Sigma_A T_A^H$ is a full SVD of an $n \times n$ matrix A, $S_A^H S_A = S_A S_A^H = T_A^H T_A = T_A T_A^H = I_n$, $\Sigma = \operatorname{diag}(\sigma_i)_{i=1}^n$, $\sigma_i = \sigma_i(A)$ for all i, $\mathbf{g} = (g_i)_{i=1}^n = -S^H\mathbf{b}$, $||A|| = ||\mathbf{b}|| = ||\mathbf{g}|| = 1$, $\sigma_{n-1} - 2cn\sqrt{n}\sigma_n > 0$ and $1/|g_n| \le cn$ for a constant c. (The latter bound is expected to hold where $\mathbf{b}$ is a scaled Gaussian random vector independent of the matrix S.) Then $\operatorname{cond} C = \sigma_n(C) \le 2cn\sqrt{n}/(\sigma_{n-1} - 2cn\sqrt{n}\sigma_n)$.

To compute a null vector of the matrix $(-\eta\mathbf{b}, A)$ we can apply the algorithms in the previous sections to the latter matrix playing the role of the matrix A.

Remark 6. If the ratio σ_1/σ_n is large, but $\operatorname{cond} C$ is not large for the matrix $C = (-\eta\mathbf{b}, A)$, then we must require high accuracy in the computations with the matrix C to avoid large output errors, but we can facilitate this task by applying the extended iterative refinement in [11].

6 Estimates for the Bit-Operation Complexity

To ensure an output precision p_t we must represent every entry of an $n \times n$ input matrix C with the bit-precision $p_s \geq p_t + \log_2 \operatorname{cond} C$, so that overall one must process $p_s n^2 \geq (p_t + \log_2 \operatorname{cond} C)n^2$ bits of the input information and perform at least $(p_t + \log_2 \operatorname{cond} C)n^2/2$ bit operations.

Recall the bounds $M(n)$ and $I(n)$ in Section 2 and let $\mu(p)$ bit operations suffice for an op performed modulo $q \leq 2^p$. ($\mu(p) = O((p \log p) \log\log p)$ and in actual numerical computing $\mu(p)$ is optimized for the IEEE standard double precision $p = p_{\text{double}}$.) Then $I(n)\mu(p_t + \log_2 \operatorname{cond} C)$ bit operations are involved in the known algorithms for a linear system $C\mathbf{y} = \mathbf{f}$ of n equations. By combining our randomized preconditioning with the extended iterative refinement in [11] we expect to yield the bound $O(I(n)\mu(p) + M(n)\mu(p)h)$ where $h = p_t/(p - \lceil \log_2 \operatorname{cond} C \rceil)$ and p, the precision of our computations, exceeds $\lceil \log_2 \operatorname{cond} C \rceil$. The term $I(n)\mu(p)$ covers the bit operations for computing an approximate inverse $X \approx C^{-1}$ with the p-bit precision. The term $M(n)\mu(p)h$ covers the bit operations in the extended iterative refinement, whose ith step amounts essentially to multiplication of the two matrices C and X by two vectors and produces a vector $\mathbf{y}_i$ with the precision p. The solution vector $\mathbf{y} = \sum_{i=1}^{h} \mathbf{y}_i$ is represented with the precision p_t by the vectors $\mathbf{y}_1, \ldots, \mathbf{y}_h$. If $p > \lceil \log_2 \operatorname{cond} C \rceil$ and $p_t + \log_2 \operatorname{cond} C \gg (1 + h/n)p$, then our approach is expected to decreases the overall bit complexity of the solution by roughly the factor $I(n)/M(n)$.

7 The Case of Hankel and Toeplitz Inputs

Hankel and Toeplitz linear systems can be solved much faster than general ones, and our improvement factor $I(n)/M(n)$ decreases to $0.05c_{\text{large}} \log n$, although the factor $0.05c_{\text{large}}$ is still quite large for c_{large} from Section 2. For numerical solution of these linear systems most popular are the algorithms running in n^2 ops, such as the one in [7], based on the *displacement transformation method* of [9] (also see [10, Sections 1.7 and 5.6] on this method). Thus the expected improvement with our techniques is still significant.

In the augmentations $A \to (B_q, A)$ for small integers q, the input structure deteriorates little (the displacement rank can grow by at most q, cf. [10]), but for any q one can and should preserve the structure intact, to enable faster computations. All our estimates can be readily extended to such randomized stuctured augmentation, except for Theorem 2. Its extension is a well known research challenge. (See [13] on our initial progress.) Remarks 1 and 5 suggest, however, that the estimates in Corollaries 2 and 3 can be extended even independently of the extension of Theorem 2. This conjecture is supported by the data on the CPU

Table 1. Condition numbers $\mathrm{cond}\, M$ of random $n \times n$ matrices M

dimension n	**type**	**min**	**max**	**mean**	**std**
32	real	2.4×10^1	1.8×10^3	2.4×10^2	3.3×10^2
32	complex	2.7×10^1	8.7×10^2	1.1×10^2	1.1×10^2
64	real	4.6×10^1	1.1×10^4	5.0×10^2	1.1×10^3
64	complex	5.2×10^1	4.2×10^3	2.7×10^2	4.6×10^2
128	real	1.0×10^2	2.7×10^4	1.1×10^3	3.0×10^3
128	complex	1.3×10^2	2.5×10^3	3.9×10^2	3.3×10^2
256	real	2.4×10^2	8.4×10^4	3.7×10^3	9.7×10^3
256	complex	2.5×10^2	1.4×10^4	1.0×10^3	1.5×10^3
512	real	3.9×10^2	7.4×10^5	1.8×10^4	8.5×10^4
512	complex	5.7×10^2	3.2×10^4	2.3×10^3	3.5×10^3
1024	real	8.8×10^2	2.3×10^5	8.8×10^3	2.4×10^4
1024	complex	7.2×10^2	1.3×10^5	5.4×10^3	1.4×10^4
2048	real	2.1×10^3	2.0×10^5	1.8×10^4	3.2×10^4
2048	complex	2.3×10^3	5.7×10^4	6.7×10^3	7.2×10^3

time that we observed in our tests implementing our algorithms; furthermore our tests show reasonably mild growth of $\mathrm{cond}\, A$ for random Toeplitz matrices $A \in \mathbb{C}^{n \times n}$ as n grows (see Tables 1–3).

8 Numerical Tests

We performed a series of numerical experiments in the Graduate Center of the City University of New York. We conducted them on a Dell server with a dual core 1.86 GHz Xeon processor and 2G memory running Windows Server 2003 R2. The test Fortran code was compiled with the GNU gfortran compiler within the Cygwin environment. Random numbers were generated with the random_number intrinsic Fortran function assuming the uniform probability distribution over the range $\{x:\ 0 \le x < 1\}$. To shift to the range $\{y:\ b \le y \le a + b\}$ for fixed real a and b, we applied the linear transform $x \to y = ax + b$. To obtain random complex numbers, we randomly generated their real and imaginary parts. We computed QR factorizations and SVDs by applying the LAPACK procedures DGEQRF and DGESVD, respectively.

Tables 1 and 2 display the average (**mean**), minimum (**min**), maximum (**max**), and standard deviation (**std**) of the condition numbers for $n \times n$ random general and Toeplitz matrices that we observed in 100 runs for each n.

Table 3 shows data on the CPU time for computing the null vectors of random singular circulant matrices. The data are average over 100 runs for each algorithm and each input size. Circulant matrices form an important subclass of Toeplitz matrices; our tests with random Toeplitz matrices gave similar results [12].

We measured the CPU time in terms of the CPU cycles. To convert into seconds divide them by CLOCKS_PER_SEC, which is 1000 on our platform. Algorithm 2 reduced the task to solving nonsingular well conditioned Toeplitz

Table 2. Condition numbers $\mathrm{cond}_1 T$ of random real $n \times n$ Toeplitz matrices T

dimension n	**min**	**mean**	**max**	**std**
256	9.1×10^2	9.2×10^3	1.3×10^5	1.8×10^4
512	2.3×10^3	3.0×10^4	2.4×10^5	4.9×10^4
1024	5.6×10^3	7.0×10^4	1.8×10^6	2.0×10^5
2048	1.7×10^4	1.8×10^5	4.2×10^6	5.4×10^5
4096	4.3×10^4	2.7×10^5	1.9×10^6	3.4×10^5
8192	8.8×10^4	1.2×10^6	1.3×10^7	2.2×10^6

Table 3. CPU time (in cycles) for computing null vectors of circulant matrices

size	**Algorithm 2**	**QR**	**SVD**	**QR/Algorithm 2**	**SVD/Algorithm 2**
256	3.0	18.8	261.5	6.3	87.2
512	7.3	147.9	4220.9	20.3	578.2
1024	16.1	1538.3	70452.5	97.1	4445.8
2048	35.5	11748.3	–	342.1	–
4096	78.7	–	–	–	–
8192	170.4	–	–	–	–

linear systems of equations, for which we applied the code from [15] (cf. [16]). For comparison we also obtained the solution based on computing the QR factorization and SVD of the input matrices. The table entries are marked by a "-" where the tests required too long runtime and were not completed. Otherwise in all our tests we computed approximate null vectors $\mathbf{y}$ within the relative residual norms bound $\frac{||A\mathbf{y}||}{||A||\ ||\mathbf{y}||}$ of the order of 10^{-17}. The reader can download our codes for these tests from http://comet.lehman.cuny.edu/vpan/. For other tests supporting the presented algorithms and for detailed comments, see [12] and [13].

Acknowledgement. We thank the referees and the Program Committee Chair Ernst W. Mayr for helpful comments.

References

1. Bunch, J.R.: Stability of Methods for Solving Toeplitz Systems of Equations. SIAM J. Sci. Stat. Comput. 6(2), 349–364 (1985)
2. Benzi, M.: Preconditioning Techniques for Large Linear Systems: a Survey. J. of Computational Physics 182, 418–477 (2002)
3. Coppersmith, D., Winograd, S.: Matrix Multiplicaton via Arithmetic Progressions. J. of Symbolic Computation 9(3), 251–280 (1990)
4. Demmel, J.: The Probability That a Numerical Analysis Problem Is Difficult. Math. of Computation 50, 449–480 (1988)
5. Demillo, R.A., Lipton, R.J.: A Probabilistic Remark on Algebraic Program Testing. Information Processing Letters 7(4), 193–195 (1978)

6. Edelman, A.: Eigenvalues and Condition Numbers of Random Matrices, PhD Thesis (106 pages), Math Dept., MIT (1989); SIAM J. on Matrix Analysis and Applications, 9(4), 543–560 (1988)
7. Gu, M.: Stable and Efficient Algorithms for Structured Systems of Linear Equations. SIAM J. on Matrix Analysis and Applications 19, 279–306 (1998)
8. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. The Johns Hopkins University Press, Baltimore (1996)
9. Pan, V.Y.: On Computations with Dense Structured Matrices. Math. of Computation 55(191), 179–190 (1990)
10. Pan, V.Y.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Birkhäuser/Springer, Boston/New York (2001)
11. Pan, V.Y., Grady, D., Murphy, B., Qian, G., Rosholt, R.E., Ruslanov, A.: Schur Aggregation for Linear Systems and Determinants. Theoretical Computer Science 409(2), 255–268 (2008)
12. Pan, V.Y., Qian, G.: On Solving Linear System with Randomized Augmentation. Tech. Report TR 2009009, Ph.D. Program in Computer Science, Graduate Center, the City University of New York (2009), http://www.cs.gc.cuny.edu/tr/techreport.php?id=352
13. Pan, V.Y., Qian, G., Zheng, A.: Randomized Preprocessing versus Pivoting. Tech. Report TR 2009010, Ph.D. Program in Computer Science, Graduate Center, the City University of New York (2009), http://www.cs.gc.cuny.edu/tr/techreport.php?id=352
14. Sankar, A., Spielman, D., Teng, S.-H.: Smoothed Analysis of the Condition Numbers and Growth Factors of Matrices. SIAM Journal on Matrix Analysis 28(2), 446–476 (2006)
15. Van Barel, M.: A Supefast Toeplitz Solver (1999), http://www.cs.kuleuven.be/~marc/software/index.html
16. Van Barel, M., Heinig, G., Kravanja, P.: A Stabilized Superfast Solver for Nonsymmetric Toeplitz Systems. SIAM Journal on Matrix Analysis and Applications 23(2), 494–510 (2001)

Transfinite Sequences of Constructive Predicate Logics

Valery Plisko*

Faculty of Mechanics and Mathematics, Moscow State University,
Moscow 119991, Russia

Abstract. The predicate logic of recursive realizability was carefully studied by the author in the 1970s. In particular, it was proved that the predicate realizability logic changes if the basic language of formal arithmetic is replaced by its extension with the truth predicate. In the paper this result is generalized by considering similar extensions of the arithmetical language by transfinite induction up to an arbitrary constructive ordinal. Namely, for every constructive ordinal α, a transfinite sequence of extensions LA_β ($\beta \leq \alpha$) of the first-order language of formal arithmetic is considered. Constructive semantics for these languages is defined in terms of recursive realizability. Variants of LA_β-realizability for the predicate formulas are considered and corresponding predicate logics are studied.

1 Introduction

Constructive semantics of formal languages are widely used in intuitionistic proof theory. Such semantics are also interesting because of their applications in computer science, especially in extracting algorithms from constructive proofs. Historically, the first precise constructive semantics of the language of formal arithmetic LA was recursive realizability introduced by S. C. Kleene [1].

For any semantics it is of interest to study the corresponding logic as the set of predicate formulas being schemata of propositions true in the given semantics. A predicate formula $\mathcal{A}$ is *constructively valid* if there is an algorithm for establishing the constructive truth for every proposition obtained by substituting concrete predicates for the predicate variables in $\mathcal{A}$. In constructive mathematics, a predicate is commonly understood as a parametric sentence in an appropriate language. Thus the concept of a predicate can be made more precise (and unavoidably narrowed) by choosing a language for formulating the predicates. If a first-order language L with a constructive semantics is fixed, then an m-place predicate is defined as a formula in L with m free variables. A closed pedicate formula $\mathcal{A}$ is *constructively L-valid* if there is an algorithm for establishing the constructive truth for L-instances of $\mathcal{A}$.

If the language LA is taken as L in the above discussion and recursive realizability is considered as a constructive semantics of this language, then we obtain

* Partially supported by RFBR grant 08-01-00399.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 315–326, 2010.

the notion of an effectively realizable predicate formula as a refinement of the concept of a constructively valid formula. It was proved by the author [2], [3] that the class of effectively realizable predicate formulas is not arithmetical.

Consider another formalized language denoted by $LA + T$. It is obtained by adding to LA a one-place predicate symbol T and the following formation rule: if t is a term, then $T(t)$ is an atomic formula. The semantics of $LA + T$ is defined under the assumption that the realizability semantics of LA is known. Namely we consider $T(n)$ as stating that n is the Gödel number of a realizable arithmetical proposition. The semantics of $LA + T$ can be described in terms of realizability: a natural number e realizes a closed formula $T(t)$ iff the value of t is the Gödel number of a realizable arithmetical proposition and e realizes that proposition.

The following results were proved by the author [4]:

1) the class of constructively $(LA+T)$-valid predicate formulas is not definable in the language $LA + T$;

2) there are constructively LA-valid predicate formulas which are not constructively $(LA + T)$-valid.

Thus the concept of a realizable predicate formula based on the arithmetical language is rather accidental from the point of view of constructive logic, because it can not be considered as an adequate definition of a constructively valid formula. In this sense the notion of an $(LA + T)$-valid predicate formula deserves more attention. We show in this paper that similar extensions can be continued by transfinite induction up an arbitrary constructive ordinal. Namely a transfinite sequence of languages with constructive semantics is defined and the results analogous to 1) and 2) proved.

The subject of the paper is interersting in a comparison with a set-theoretical approach to the constructive predicate logic proposed by the author in [5] where the notion of an absolutely realizable predicate formula is introduced. From the classical point of view, absolutely realizable predicate formulas are exactly the constructively valid formulas. The problem is to describe this semantics by constructive methods. The results of this paper show that a natural approach based on considering transfinite sequences of the languages with constructive semantics does not lead to any final concept of a constructively valid predicate formula.

2 A Transfinite Sequence of Languages

The language LA will play an essential role in further considerations. Arithmetical terms are constructed in the usual way from the individual variables, the constant 0, and the function symbols $', +, \cdot$. The terms $0, 0', 0'', 0''', \ldots$ will be denoted by $\overline{0}, \overline{1}, \overline{2}, \overline{3}, \ldots$. Arithmetical formulas are constructed from the atomic formulas of the form $t_1 = t_2$, where t_1, t_2 are terms, by using the logical symbols $\neg, \&, \vee, \supset, \forall, \exists$ and parentheses. We write $\forall x_1, \ldots, x_n$ instead of $\forall x_1 \ldots \forall x_n$ and $\exists x_1, \ldots, x_n$ instead of $\exists x_1 \ldots \exists x_n$.

Σ-formulas are arithmetical formulas of the form $\exists x_1, \ldots, x_n \Phi$, where Φ is an atomic formula. By the Matiyasevich theorem on Diophantine presentation

of the recursively enumerable sets, every recursively enumerable predicate $P(x_1, \ldots, x_n)$ is expressible by a Σ-formula $\Phi(x_1, \ldots, x_n)$ in the sense that for every natural numbers $k_1, \ldots, k_n$, $P(k_1, \ldots, k_n)$ holds iff $\Phi(\bar{k}_1, \ldots, \bar{k}_n)$ is true by the standard interpretation of LA.

Let L be the first-order language obtained by adding the two-place predicate symbol T to LA. We call L *the universal language*. The formulas of the language L will be called L-formulas. If Φ and Ψ are L-formulas, then $\Phi \equiv \Psi$ denotes the formula $(\Phi \supset \Psi)\,\&\,(\Psi \supset \Phi)$. By substituting terms $t_1, \ldots, t_n$ for distinct variables $y_1, \ldots, y_n$ in an L-formula Φ we assume that as a preliminary the bound variables in Φ are renamed in order to avoid binding the variables in $t_1, \ldots, t_n$. The result of substituting $t_1, \ldots, t_n$ for $y_1, \ldots, y_n$ in Φ is denoted by $\Phi[t_1/y_1, \ldots, t_n/y_n]$. If for Φ a notation $\Phi(y_1, \ldots, y_n)$ is used, then we write $\Phi(t_1, \ldots, y_n)$ instead of $\Phi[t_1/y_1, \ldots, t_n/y_n]$. Closed L-formulas are called L-propositions.

Let a Gödel numbering of L be fixed. Any technical details of this numbering are not essential except of the following:

- The predicate 'x is the Gödel number of an atomic arithmetical proposition' is expressed by a Σ-formula $\varepsilon(x)$;
- The predicate 'x is the Gödel number of a true atomic arithmetical proposition' is expressed by a Σ-formula $\tau(x)$;
- The predicate 'x is the Gödel number of an atomic proposition of the form $T(\bar{y}, t)$, the value of the term t being z', is expressed by a Σ-formula $\theta(x, y, z)$;
- There exists a one-place primitive recursive function $\phi_\neg$ such that if x is the Gödel number of a formula Φ, then $\phi_\neg(x)$ is the Gödel number of $\neg\Phi$; in this case, the predicate $x = \phi_\neg(y)$ is expressed by a Σ-formula $\nu_\neg(x, y)$;
- For every $\lambda \in \{\&, \vee, \supset\}$ there exists a two-place primitive recursive function ϕ_λ such that if x and y are the Gödel numbers of formulas Φ and Ψ respectively, then $\phi_\lambda(x, y)$ is the Gödel number of $(\Phi\lambda\Psi)$; in this case, the predicate $x = \phi_\lambda(y, z)$ is expressed by a Σ-formula $\nu_\lambda(x, y, z)$;
- For every $\kappa \in \{\forall, \exists\}$ there exists a two-place primitive recursive function ϕ_κ such that if y is the Gödel number of a variable v and z is the Gödel number of a formula Φ, then $\phi_\kappa(y, z)$ is the Gödel number of $\kappa v\,\Phi$; in this case, the predicate $x = \phi_\kappa(y, z)$ is expressed by a Σ-formula $\nu_\kappa(x, y, z)$;
- There exists a three-place primitive recursive function sub such that if y is the Gödel number of a variable v, z is the Gödel number of an L-formula $\Phi(v)$, and u is a natural number, then $sub(y, z, u)$ is the Gödel number of $\Phi(\bar{u})$; in this case, the predicate $x = sub(y, z, u)$ is expressed by a Σ-formula $\sigma(x, y, z, u)$.

Let $\prec$ be a recursive well-ordering of (an initial segment of) the natural numbers $\mathbf{N}$. The order-type of the ordering $\prec$ is a constructive ordinal α (see [6, §11.8]). Then every natural m is a notation of an ordinal $\beta < \alpha$; denote this fact by $|m| = \beta$. Without loss of generality one can suppose that $|0| = 0$.

For every $\beta \leq \alpha$ we define a fragment LA_β of the universal language L. LA_β-formulas, i. e., formulas of the language LA_β, are defined inductively:

- If t_1 and t_2 are terms, then $t_1 = t_2$ is a(n atomic) LA_β-formula;

- If n is a natural number such that $|n| < \beta$ and t is a term, then $T(\bar{n}, t)$ is a(n atomic) LA_β-formula;
- If Φ is an LA_β-formula, then $\neg\Phi$ is an LA_β-formula;
- If Φ and Ψ are LA_β-formulas, then $(\Phi \,\&\, \Psi)$, $(\Phi \vee \Psi)$, and $(\Phi \supset \Psi)$ are LA_β-formulas;
- If Φ is an LA_β-formula and v is a variable, then $\forall v\, \Phi$ and $\exists v\, \Phi$ are LA_β-formulas.

Note that LA_0 is LA and the LA_0-formulas are arithmetical formulas. Further, if $\gamma < \beta$, then every LA_γ-formula is an LA_β-formula, thus LA_α is the union of all the languages LA_β for $\beta \leq \alpha$. *A rank* of an LA_α-formula Φ is the least ordinal β such that Φ is an LA_β-formula.

Constructive semantics of LA_β is defined in terms of realizability. First we adopt some conventions on the notation of partial recursive functions and their indexes. By $\{e\}$ we denote the partial recursive function with index e. If an expression $\varphi(x_1, \ldots, x_n)$ defines an n-place partial recursive function, then $\Lambda x_1, \ldots, x_n.\varphi(x_1, \ldots, x_n)$ will denote an index of that function. We use $(a)_i$ to denote the exponent with which the $(i+1)$th prime number appears in the decomposition of a into prime factors.

The notion of realizability for LA_β is defined by transfinite induction on β. Let for every $\gamma < \beta$ a relation $e\,\mathbf{r}_\gamma\,\Phi$ (*a natural number e is a γ-realization of an LA_γ-proposition Φ*) be defined. Then we define $e\,\mathbf{r}_\beta\,\Phi$ by induction on the number of logical symbols in an LA_β-proposition Φ.

- If Φ is an atomic proposition $t_1 = t_2$, then $e\,\mathbf{r}_\beta\,\Phi \rightleftharpoons [e = 0$ and Φ is true$]$.
- If Φ is an atomic proposition of the form $T(\bar{m}, t)$, where $|m| = \gamma < \beta$ and t is a term whose value is n, then $e\,\mathbf{r}_\beta\,\Phi \rightleftharpoons [n$ is the Gödel number of an LA_γ-proposition Ψ, and $e\,\mathbf{r}_\gamma\,\Psi]$.

Let Φ_0 and Φ_1 be LA_β-propositions. Then

- $e\,\mathbf{r}_\beta\,(\Phi_0 \,\&\, \Phi_1) \rightleftharpoons [(e)_0\,\mathbf{r}_\beta\,\Phi_0$ and $(e)_1\,\mathbf{r}_\beta\,\Phi_1]$;
- $e\,\mathbf{r}_\beta\,(\Phi_0 \vee \Phi_1) \rightleftharpoons [(e)_0 = 0$ and $(e)_1\,\mathbf{r}_\beta\,\Phi_0$ or $(e)_0 = 1$ and $(e)_1\,\mathbf{r}_\beta\,\Phi_1]$;
- $e\,\mathbf{r}_\beta\,(\Phi_0 \supset \Phi_1) \rightleftharpoons [\forall a\,(a\,\mathbf{r}_\beta\,\Phi_0 \Rightarrow \{e\}(a)\,\mathbf{r}_\beta\,\Phi_1)]$;
- $e\,\mathbf{r}_\beta\,\neg\Phi_0 \rightleftharpoons [e\,\mathbf{r}_\beta\,(\Phi_0 \supset 0 = 1)]$.

Let $\Phi_0(x)$ be an LA_β-formula with the only parameter x. Then

- $e\,\mathbf{r}_\beta\,\forall x\,\Phi_0(x) \rightleftharpoons [\forall n\,\{e\}(n)\,\mathbf{r}_\beta\,\Phi_0(\bar{n})]$;
- $e\,\mathbf{r}_\beta\,\exists x\,\Phi_0(x) \rightleftharpoons [(e)_1\,\mathbf{r}_\beta\,\Phi_0(\overline{(e)_0})]$.

Note that if $\gamma < \beta$, then for every LA_γ-proposition Φ we have

$$\forall e\,[e\,\mathbf{r}_\gamma\,\Phi \Leftrightarrow e\,\mathbf{r}_\beta\,\Phi].$$

Therefore we can omit the subscript in $\mathbf{r}_\beta$. Now, if Φ is an LA_γ-proposition, then $e\,\mathbf{r}\,\Phi$ means that $e\,\mathbf{r}_\beta\,\Phi$ for every β such that $\gamma \leq \beta \leq \alpha$, in particular, $\mathbf{r}$ means the same as $\mathbf{r}_\alpha$.

If $e\,\mathbf{r}\,\Phi$, we say that e is *a realization* of Φ. An LA_α-proposition Φ is called *realizable* iff there exists a realization of Φ. Note that if a Σ-proposition is true, then we can effectively find its realization. Let $\mathbf{R}_\beta(x)$ mean that x is the Gödel number of a realizable LA_β-proposition. An LA_α-formula with free variables is realizable iff its universal closure is realizable. LA_α-formulas Φ and Ψ are called *equivalent* iff the universal closure of $\Phi \equiv \Psi$ is realizabale.

We say that an n-place predicate $P(x_1, \ldots, x_n)$ is LA_β-definable, if there exists an LA_β-formula $\Phi(x_1, \ldots, x_n)$ with the only free variables $x_1, \ldots, x_n$ such that for every natural numbers $k_1, \ldots, k_n$, $P(k_1, \ldots, k_n)$ holds iff $\Phi(\bar{k}_1, \ldots, \bar{k}_n)$ is realizable. The following theorem can be proved by just the same arguments as the Tarski theorem on the inexpressibility of the truth predicate for a formal language by a formula of that language, if the language is sufficiently rich.

Proposition 1. *The predicate* $\mathbf{R}_\beta(x)$ *is not* LA_β*-definable.*

We say that a language L_2 is more expressive than another language L_1 if every L_1-definable predicate is L_2-definable, but there exists an L_2-definable predicate which is not L_2-definable.

Proposition 2. *If* $\gamma < \beta \leq \alpha$*, then the language* LA_β *is more expressive than* LA_γ*.*

Proof. If $\gamma = |m|$, then the the LA_β-formula $T(\bar{m}, x)$ defines the predicate $\mathbf{R}_\gamma(x)$ in the language LA_β, but in view of Proposition 1, this predicate can not be defined by any LA_γ-formula.

3 Realizable Predicate Formulas

Predicate formulas are first-order formulas constructed in the usual way from predicate variables A_i^j $(i, j = 0, 1, 2, \ldots)$ and individual variables $x_1, x_2, \ldots$. We say that A_n^m is an m-ary predicate variable.

The notion of *a scheme* is a generalization of both notions of a predicate formula and an arithmetical formula. Schemata are defined inductively.

1. If t_1 and t_2 are arithmetical terms, then $t_1 = t_2$ is a scheme.
2. If P is an n-ary predicate variable, $v_1, \ldots, v_n$ are individual variables, then $P(v_1, \ldots, v_n)$ is a scheme.
3. If $\mathcal{A}$ is a scheme, then $\neg\mathcal{A}$ is a scheme.
4. If $\mathcal{A}$ and $\mathcal{B}$ are schemata, then $(\mathcal{A}\,\&\,\mathcal{B})$, $(\mathcal{A} \vee \mathcal{B})$, $(\mathcal{A} \supset \mathcal{B})$ are schemata.
5. If $\mathcal{A}$ is a scheme, v is an individual variable, then $\forall v\,\mathcal{A}$ and $\exists v\,\mathcal{A}$ are schemata.

Arithmetical formulas are evidently schemata constructed according to rules 1 and 3–5, and predicate formulas are schemata constructed according to rules 2–5.

Let $\mathcal{A}$ be a scheme, $P_1, \ldots, P_n$ be all the predicate variables in $\mathcal{A}$, P_i being m_i-ary. Denote $\mathcal{A}$ by $\mathcal{A}(P_1, \ldots, P_n)$. A list of L-formulas $\Phi_1, \ldots, \Phi_n$ is admissible for substitution in $\mathcal{A}(P_1, \ldots, P_n)$ if for every $i \in \{1, \ldots, n\}$, Φ_i does not contain any free variables other than $x_1, \ldots, x_{m_i}$. If a list $\Phi_1, \ldots, \Phi_n$ is admissible

for substitution in $\mathcal{A}(P_1,\dots,P_n)$, let $\mathcal{A}(\Phi_1,\dots,\Phi_n)$ be an L-formula obtained by replacing each atomic subformula $P_i(y_1,\dots,y_{m_i})$ by $\Phi_i[y_1/x_1,\dots,y_{m_i}/x_{m_i}]$. In this case, the formula $\mathcal{A}(\Phi_1,\dots,\Phi_n)$ is called *an L-instance* of the scheme $\mathcal{A}(P_1,\dots,P_n)$. A closed scheme $\mathcal{A}$ is called

- *Effectively LA_β-realizable* if there exists an algorithm allowing to find a realization of every closed LA_β-instance of $\mathcal{A}$;
- *Uniformly LA_β-realizable* if there exists a natural number realizing every closed LA_β-instance of $\mathcal{A}$.

Obviously, if a scheme is uniformly LA_β-realizable, then it is effectively LA_β-realizable. Since every predicate formula is a scheme, the notions of an effectively LA_β-realizable and a uniformly LA_β-realizable closed predicate formula are defined. The main purpose of this paper is studying relations between the notions of LA_β-realizability for different βs.

The notion of a scheme is useful in view of the following theorem.

Theorem 1 (Theorem on Schemata). *For every closed scheme $\mathcal{A}$ one can effectively construct a closed predicate formula $\mathcal{A}^*$ such that for every $\beta \le \alpha$:*
1) $\mathcal{A}$ is effectively LA_β-realizable iff $\mathcal{A}^*$ is effectively LA_β-realizable;
2) $\mathcal{A}$ is uniformly LA_β-realizable iff $\mathcal{A}^*$ is uniformly LA_β-realizable.

This theorem is a generalization of Theorem 1 in [7] and can be proved in just the same way.

4 The Basic Theorem

Obviously, the predicate 'y is the Gödel number of an $LA_{|x|}$-proposition' is recursive and is defined by a Σ-formula $\pi(x,y)$. Let R be a one-place predicate variable, w be an individual variable, and $[R,w]$ be the conjunction of the following schemata:

$$\forall x\,(\neg\pi(w,x) \supset \neg R(x)); \tag{1}$$

$$\forall x\,(\varepsilon(x) \supset (R(x) \equiv \tau(x))); \tag{2}$$

$$\forall x,y,z\,(\pi(w,x)\,\&\,\theta(x,y,z)\,\&\,\neg\pi(y,z) \supset \neg R(x)); \tag{3}$$

$$\forall x,y,z,u\,(\pi(w,x)\,\&\,\theta(x,y,z)\,\&\,\pi(y,z) \supset (R(x) \equiv R(z)); \tag{4}$$

$$\forall x,y,z\,(\pi(w,x)\,\&\,\nu_\neg(x,y) \supset (R(x) \equiv \neg R(y))); \tag{5}$$

$$\forall x,y,z\,(\pi(w,x)\,\&\,\nu_\&(y,z) \supset (R(x) \equiv (R(y)\,\&\,R(z)))); \tag{6}$$

$$\forall x,y,z\,(\pi(w,x)\,\&\,\nu_\vee(x,y,z) \supset (R(x) \equiv (R(y) \vee R(z)))); \tag{7}$$

$$\forall x,y,z\,(\pi(w,x)\,\&\,\nu_\supset(x,y,z) \supset (R(x) \equiv (R(y) \supset R(z)))); \tag{8}$$

$$\forall x,y,z\,(\pi(w,x)\,\&\,\nu_\exists(x,y,z) \supset (R(x) \equiv \exists v,t\,(\sigma(t,y,z,v)\,\&\,R(t)))); \tag{9}$$

$$\forall x,y,z\,(\pi(w,x)\,\&\,\nu_\forall(y,z,u) \supset (R(x) \equiv \forall v\,\exists t\,(\sigma(t,y,z,v)\,\&\,R(x,t)))). \tag{10}$$

Let $\Phi(x)$ be an LA_α-formula with the only free variable x and $[\Phi(x), w]$ be obtained by substituting $\Phi(x)$ for R in $[R, w]$. Let $|n| = \beta < \alpha$. It is rather evident that the formula $[T(\bar{n}, x), \bar{n}]$ is realizable. The following proposition means that this is the only such formula up to the equivalence.

Theorem 2. *There is a partial recursive function h such that for every formula $\Phi(x)$ and every natural a, if $a\,\mathbf{r}\,[\Phi(x), \bar{n}]$, then $h(a)\,\mathbf{r}\,\forall x\,(\Phi(x) \equiv T(\bar{n}, x))$.*

Proof. We define three-place partial recursive functions f and g such that for every Φ and a, if $a\,\mathbf{r}\,[\Phi(x), \bar{n}]$, then

$$\forall l, x\,[x\,\mathbf{r}\,\Phi(\bar{l}) \Rightarrow f(a, x, l)\,\mathbf{r}\,T(\bar{n}, \bar{l})]; \tag{11}$$

$$\forall l, x\,[x\,\mathbf{r}\,T(\bar{n}, \bar{l}) \Rightarrow g(a, x, l)\,\mathbf{r}\,\Phi(\bar{l})]. \tag{12}$$

After that we set $h(a) = \Lambda l.2^{\Lambda x.f(a,x,l)} \cdot 3^{\Lambda x.g(a,x,l)}$.

Let a formula $\Phi(x)$ and a realization a of $[\Phi(x), \bar{n}]$ be given. If l is not a Gödel number of any LA_β-proposition, then $\neg\pi(\bar{n}, \bar{l})$ is realizable. As (1) is a conjunct in $[R, w]$ and $[\Phi(x), \bar{n}]$ is realizable, $\neg\Phi(\bar{l})$ is realizable and $\Phi(\bar{l})$ is not realizable. On the other hand, in this case, $T(\bar{n}, \bar{l})$ is not realizable too. Therefore for every a and x we can set $f(a, x, l) = x$, $g(a, x, l) = x$.

Let l be the Gödel number of an LA_β-proposition Ψ. Then $\pi(\bar{n}, \bar{l})$ is realizable. We define $f(a, x, l)$ and $g(a, x, l)$ by transfinite induction (see [6, Chapter 11]) on the rank of Ψ.

If rank of Ψ is 0, then Ψ is an arithmetical proposition. Now we use induction on the number of logical connectives and quantifiers in Ψ.

Let Ψ be an atomic arithmetical proposition. Then we have a realization of $\varepsilon(\bar{l})$. As a realization a of $[\Phi(x), \bar{n}]$ is given and (2) is a conjunct in $[R, w]$, we can find a realization c of $\Phi(\bar{l}) \equiv \tau(\bar{l})$. Let $x\,\mathbf{r}\,\Phi(\bar{l})$. Then $\tau(\bar{l})$ is realizable. This means that Ψ is true and realizable, 0 being its realization. Thus $0\,\mathbf{r}\,T(\bar{n}, \bar{l})$ and we can set $f(a, x, l) = 0$. Now let $x\,\mathbf{r}\,T(\bar{n}, \bar{l})$. This means that $x\,\mathbf{r}\,\Psi$. Therefore $\tau(\bar{l})$ is true and we can find its realization t. Then we can set $g(a, x, l) = \{(c)_1\}(t)$.

Let Ψ be of the form $\neg\Psi_0$ and l_0 be the Gödel number of Ψ_0. In this case, we have a realization of $\nu_\neg(\bar{l}, \bar{l}_0)$. As a realization a of $[\Phi(x), \bar{n}]$ is given and (5) is a conjunct in $[R, w]$, we can find a realization c of $\Phi(\bar{l}) \equiv \neg\Phi(\bar{l}_0)$. By inductive hypothesis, $f(a, x, l_0)$ and $g(a, x, l_0)$ are defined in such a way, that

$$\forall x\,[x\,\mathbf{r}\,\Phi(\bar{l}_0) \Rightarrow f(a, x, l_0)\,\mathbf{r}\,T(\bar{n}, \bar{l}_0)]; \tag{13}$$

$$\forall x\,[x\,\mathbf{r}\,T(\bar{n}, \bar{l}_0) \Rightarrow g(a, x, l_0)\,\mathbf{r}\,\Phi(\bar{l}_0)]. \tag{14}$$

Let $x\,\mathbf{r}\,\Phi(\bar{l})$, then $\{(c)_0\}(x)\,\mathbf{r}\,\neg\Phi(\bar{l}_0)$. Therefore $\Phi(\bar{l}_0)$ is not realizable. It follows from (14) that $T(\bar{n}, \bar{l}_0)$ is not realizable too. This means that Ψ_0 is not realizable, $\neg\Psi_0$ is realizable, $0\,\mathbf{r}\,\Psi$, and $0\,\mathbf{r}T(\bar{n}, \bar{l})$. Thus we can set $f(a, x, l) = 0$.

Now let $x\,\mathbf{r}\,T(\bar{n}, \bar{l})$. This means that $x\,\mathbf{r}\,\neg\Psi_0$ and Ψ_0 is not realizable, thus $T(\bar{n}, \bar{l}_0)$ is not realizable. It follows from (13) that $\Phi(\bar{l}_0)$ is not realizable too. Therefore, $0\,\mathbf{r}\,\neg\Phi(\bar{l}_1)$ and $\{(c)_1\}(0)\,\mathbf{r}\,\Phi(\bar{l})$. Thus we can set $g(a, x, l) = \{(c)_1\}(0)$.

Let Ψ be of the form $(\Psi_0 \lambda \Psi_1)$, where $\lambda \in \{\&, \vee, \supset\}$. If l_0, l_1 are the Gödel numbers of Ψ_0 and Ψ_1 respectively, then we have a realization of $\nu_\lambda(\bar{l}, \bar{l}_0, \bar{l}_1)$. By

inductive hypothesis, $f(a,x,l_0)$, $g(a,x,l_0)$, $f(a,x,l_1)$, $g(a,x,l_1)$ are defined in such a way, that (13) and (14) hold and

$$\forall x\,[x\,\mathbf{r}\,\Phi(\bar{l}_1) \Rightarrow f(a,x,l_1)\,\mathbf{r}\,T(\bar{n},\bar{l}_1)]; \tag{15}$$

$$\forall x\,[x\,\mathbf{r}\,T(\bar{n},\bar{l}_1) \Rightarrow g(a,x,l_1)\,\mathbf{r}\,\Phi(\bar{l}_1)]. \tag{16}$$

Let $\lambda = \&$. As a realization a of $[\Phi(x),\bar{n}]$ is given and (6) is a conjunct in $[R,w]$, we can find a realization c of $\Phi(\bar{l}) \equiv (\Phi(\bar{l}_0)\,\&\,\Phi(\bar{l}_1))$.

Let $x\,\mathbf{r}\,\Phi(\bar{l})$, then $\{(c)_0\}(x)\,\mathbf{r}\,(\Phi(\bar{l}_0)\,\&\,\Phi(\bar{l}_1))$. Therefore,

$$(\{(c)_0\}(x))_0\,\mathbf{r}\,\Phi(\bar{l}_0),\ (\{(c)_0\}(x))_1\,\mathbf{r}\,\Phi(\bar{l}_1).$$

It follows from (13) that $f(a,(\{(c)_0\}(x))_0,l_0)\,\mathbf{r}\,T(\bar{n},\bar{l}_0)$. This means that

$$f(a,(\{(c)_0\}(x))_0,l_0)\,\mathbf{r}\,\Psi_0. \tag{17}$$

Similarly, it follows from (15) that $f(a,(\{(c)_0\}(x))_1,l_1)\,\mathbf{r}\,T(\bar{n},\bar{l}_1)$ and

$$f(a,(\{(c)_0\}(x))_1,l_1)\,\mathbf{r}\,\Psi_1. \tag{18}$$

Using (17) and (18), we get $2^{f(a,(\{(c)_0\}(x))_0,l_0)}\cdot 3^{f(a,(\{(c)_0\}(x))_1,l_1)}\,\mathbf{r}\,\Psi$. This means that $2^{f(a,(\{(c)_0\}(x))_0,l_0)}\cdot 3^{f(a,(\{(c)_0\}(x))_0,l_1)}\,\mathbf{r}\,T(\bar{n},\bar{l})$. It follows that we can set

$$f(a,x,l) = 2^{f(a,(\{(c)_0\}(x))_0,l_0)}\cdot 3^{f(a,(\{(c)_0\}(x))_1,l_1)}.$$

Let $x\,\mathbf{r}\,T(\bar{n},\bar{l})$. This means that $x\,\mathbf{r}\,(\Psi_0\,\&\,\Psi_1)$, thus $(x)_0\,\mathbf{r}\,\Psi_0$, $(x)_1\,\mathbf{r}\,\Psi_1$. It follows $(x)_0\,\mathbf{r}\,T(\bar{n},\bar{l}_0)$, $(x)_1\,\mathbf{r}\,T(\bar{n},\bar{l}_1)$. By (16) and (18),

$$g(a,(x)_0,l_0)\,\mathbf{r}\,\Phi(\bar{l}_0),\ g(a,(x)_1,l_1)\,\mathbf{r}\,\Phi(\bar{l}_1),$$

thus $2^{g(a,(x)_0,l_0)}\cdot 3^{g(a,(x)_1,l_1)}\,\mathbf{r}\,(\Phi(\bar{l}_0)\,\&\,\Phi(\bar{l}_1))$ and

$$\{(c)_1\}(2^{g(a,(x)_0,l_0)}\cdot 3^{g(a,(x)_1,l_1)})\,\mathbf{r}\,\Phi(\bar{l}).$$

Thus we can set $g(a,x,l) = \{(c)_1\}(2^{g(a,(x)_0,l_0)}\cdot 3^{g(a,(x)_1,l_1)})$.

Let $\lambda = \vee$. As a realization a of $[\Phi(x),\bar{n}]$ is given and (7) is a conjunct in $[R,w]$, we can find a realization c of $\Phi(\bar{l}) \equiv (\Phi(\bar{l}_0)\vee\Phi(\bar{l}_1))$.

If $x\,\mathbf{r}\,\Phi(\bar{l})$, then $\{(c)_0\}(x)\,\mathbf{r}\,(\Phi(\bar{l}_0)\vee\Phi(\bar{l}_1))$. Thus, 1) $(\{(c)_0\}(x))_0 = 0$ and $(\{(c)_0\}(x))_1\,\mathbf{r}\,\Phi(\bar{l}_0)$ or 2) $(\{(c)_0\}(x))_0 = 1$ and $(\{(c)_0\}(x))_1\,\mathbf{r}\,\Phi(\bar{l}_1)$. In the case 1), by (13), $f(a,(\{(c)_0\}(x))_1,l_0)\,\mathbf{r}\,T(\bar{n},\bar{l}_0)$. This means $f(a,(\{(c)_0\}(x))_1,l_0)\,\mathbf{r}\,\Psi_0$, $2^0\cdot 3^{f(a,(\{(c)_0\}(x))_1,l_0)}\,\mathbf{r}\,\Psi$, and $2^0\cdot 3^{f(a,(\{(c)_0\}(x))_1,l_0)}\,\mathbf{r}\,T(\bar{n},\bar{l})$, thus we can set

$$f(a,x,l) = 2^0\cdot 3^{f((\{(c)_0\}(x))_1,l_0)}.$$

Similarly, in the case 2), we can set

$$f(a,x,l) = 2^1\cdot 3^{f(a,(\{(c)_0\}(x))_1,l_1)}.$$

Let $x\,\mathbf{r}\,T(\bar{n},\bar{l})$. This means $x\,\mathbf{r}\,(\Psi_0 \vee \Psi_1)$, therefore, 1) $(x)_0 = 0$ and $(x)_1\,\mathbf{r}\,\Psi_0$ or 2) $(x)_0 = 1$ and $(x)_1\,\mathbf{r}\,\Psi_1$. In the case 1), $(x)_1\,\mathbf{r}\,T(\bar{n},\bar{l}_0)$. It follows from (16) that $g(a,(x)_1,l_0)\,\mathbf{r}\,\Phi(\bar{l}_0)$, $2^0\cdot 3^{g(a,(x)_1,l_0)}\,\mathbf{r}\,(\Phi(\bar{l}_0)\vee\Phi(\bar{l}_1))$ and

$$\{(c)_1\}(2^0\cdot 3^{g(a,(x)_1,l_0)})\,\mathbf{r}\,\Phi(\bar{l}).$$

Thus we can set $g(a,x,l) = \{(c)_1\}(2^0\cdot 3^{g(a,(x)_1,l_0)})$. In the case 2), we can set $g(a,x,l) = \{(c)_1\}(2^1\cdot 3^{g(a,(x)_1,l_1)})$.

Let $\lambda = \supset$. As a realization a of $[\Phi(x),\bar{n}]$ is given and (8) is a conjunct in $[R,w]$, we can find a realization c of $\Phi(\bar{l}) \equiv (\Phi(\bar{l}_0)\supset\Phi(\bar{l}_1))$.

Let $x\,\mathbf{r}\,\Phi(\bar{l})$, then $\{(c)_0\}(x)\,\mathbf{r}\,(\Phi(\bar{l}_0)\supset\Phi(\bar{l}_1))$. Note that $f(a,x,l)$ has to be a realization of $T(\bar{n},\bar{l})$, i. e., a realization of $\Psi_0\supset\Psi_1$. Let $f(a,x,l)$ be an index of the partial recursive function χ defined in the following way:

$$\chi(z) = f(a,\{\{(c)_0\}(x)\}(g(a,z,l_0)),l_1).$$

We prove that if $z\,\mathbf{r}\,\Psi_0$, then $\chi(z)\,\mathbf{r}\,\Psi_1$. Let $z\,\mathbf{r}\,\Psi_0$. This means that $z\,\mathbf{r}\,T(\bar{n},\bar{l}_0)$. It follows from (14) that $g(a,z,l_0)\,\mathbf{r}\,\Phi(\bar{l}_0)$. Then $\{\{(c)_0\}(x)\}(g(a,z,l_0))\,\mathbf{r}\,\Phi(\bar{l}_1)$ and by (15), $f(a,\{\{(c)_0\}(x)\}(g(a,z,l_0)),l_1)\,\mathbf{r}\,T(\bar{n},\bar{l}_1)$. This just means that $\chi(z)\,\mathbf{r}\,\Psi_1$. Thus we can set $f(a,x,l) = \Lambda z.f(a,\{\{(c)_0\}(x)\}(g(a,z,l_0)),l_1)$.

Let $x\,\mathbf{r}\,T(\bar{n},\bar{l})$. This means that $x\,\mathbf{r}\,(\Psi_0\supset\Psi_1)$. First we find a realization of $\Phi(\bar{l}_0)\supset\Phi(\bar{l}_1)$. It will be an index of the partial recursive function ξ defined in the following way:

$$\xi(b) = g(a,\{x\}(f(a,b,l_0)),l_1).$$

We see that $\Lambda b.\xi(b)\,\mathbf{r}\,(\Phi(\bar{l}_0)\supset\Phi(\bar{l}_1))$. Then $\{(c)_1\}(\Lambda b.\xi(b))\,\mathbf{r}\,\Phi(\bar{l})$, and we can set $g(a,x,l) = \{(c)_1\}(\Lambda b.g(a,\{x\}(f(a,b,l_0)),l_1))$.

Let Ψ be of the form $\kappa v\,\Psi_0(v)$, where $\kappa\in\{\exists,\forall\}$. If p,l_0 are the Gödel numbers of v and $\Psi_0(v)$ respectively, then we have a realization of $\nu_\kappa(\bar{l},\bar{p},\bar{l}_0)$. By inductive hypothesis, if l_1,q are such that $\sigma(\bar{l}_1,\bar{p},\bar{l}_0,\bar{q})$ is true (and realizable), then the values $f(a,x,l_1)$ and $g(a,x,l_1)$ are defined in such a way, that (15) and (16) hold.

Let $\kappa = \exists$. As a realization a of $[\Phi(x),\bar{n}]$ is given and (9) is a conjunct in $[R,w]$, we can find a realization c of $\Phi(\bar{l}) \equiv \exists v,t\,(\sigma(t,\bar{p},\bar{l}_0,v)\,\&\,\Phi(t))$.

Let $x\,\mathbf{r}\,\Phi(\bar{l})$, then

$$\{(c)_0\}(x)\,\mathbf{r}\,\exists v,t\,(\sigma(t,\bar{p},\bar{l}_0,v)\,\&\,\Phi(t)). \tag{19}$$

Denote $((\{(c)_0\}(x))_1)_0$ by l_1, $(\{(c)_0\}(x))_0$ by q, and $((\{(c)_0\}(x))_1)_1$ by d. It is easily shown that $2^q\cdot 3^{f(a,(d)_1,l_1)}\,\mathbf{r}\,\exists v\,\Psi_0(v)$ and $2^q\cdot 3^{f(a,(d)_1,l_1)}\,\mathbf{r}\,T(\bar{n},\bar{l})$. Thus we can set $f(a,x,l) = 2^{(\{(c)_0\}(x))_0}\cdot 3^{f(a,((((\{(c)_0\}(x))_1)_1)_1,(((\{(c)_0\}(x))_1)_0)}$.

Let $x\,\mathbf{r}\,T(\bar{n},\bar{l})$. This means $x\,\mathbf{r}\,\exists v\,\Psi_0(v)$. Then $(x)_1\,\mathbf{r}\,\Psi_0(\overline{(x)_0})$. Let l_1 be the Gödel number of $\Psi_0(\overline{(x)_0})$, then $\sigma(\bar{l}_1,\bar{p},\bar{l}_0,\overline{(x)_0})$ is true and we can find its realization d. It can be proved that $2^{(x)_0}\cdot 3^{2^{l_1}\cdot 3^{2^d\cdot 3^{g(a,(x)_1,l_1)}}}\,\mathbf{r}\,\exists v,t\,(\sigma(t,\bar{p},\bar{l}_0,v)\,\&\,\Phi(t))$ and $\{(c)_1\}(2^{(x)_0}\cdot 3^{2^{l_1}\cdot 3^{2^d\cdot 3^{g(a,(x)_1,l_1)}}})\,\mathbf{r}\,\Phi(\bar{l})$. Thus we can set

$$g(a,x,l) = \{(c)_1\}(2^{(x)_0}\cdot 3^{2^{l_1}\cdot 3^{2^d\cdot 3^{g(a,(x)_1,l_1)}}}).$$

Let $\kappa = \forall$. As a realization a of $[\Phi(x), \bar{n}]$ is given and (10) is a conjunct in $[R, w]$, we can find a realization c of $\Phi(\bar{l}) \equiv \forall v\, \exists t\, (\sigma(t, \bar{p}, \bar{l}_0, v)\, \&\, \Phi(t))$.

Let $x\, \mathbf{r}\, \Phi(\bar{l})$, then $\{(c)_0\}(x)\, \mathbf{r}\, \forall v\, \exists t\, (\sigma(t, \bar{p}, \bar{l}_0, v)\, \&\, \Phi(t))$. Let

$$\zeta(v) = f(a, (e)_1, l_1),$$

where $e = (\{\{(c)_0\}(x)\}(v))_1$, $l_1 = (\{\{(c)_0\}(x)\}(v))_0$. It can be proved that $\Lambda v.\zeta(v)\, \mathbf{r}\, \Psi$. Thus $\Lambda v.\zeta(v)\, \mathbf{r}\, T(\bar{n}, \bar{l})$ and we can set $f(a, x, l) = \Lambda v.f(a, (e)_1, l_1)$.

Let $x\, \mathbf{r}\, T(\bar{n}, \bar{l})$. This means that $x\, \mathbf{r}\, \forall v\, \Psi_0(v)$. Let $\eta(v) = 2^{l_1} \cdot 3^{2^d \cdot 3^{g(a, \{x\}(v), l_1)}}$, where l_1 is the Gödel number of $\Psi_0(\bar{v})$ and d is a realization of $\sigma(l_1, p, l_0, \bar{v})$. It is easy to prove that $\Lambda v.\eta(v)$ is a realization of $\forall v\, \exists t\, (\sigma(t, \bar{p}, \bar{l}_0, v)\, \&\, \Phi(t))$. Then $\{(c)_1\}(\Lambda v.\eta(v))\, \mathbf{r}\, \Phi(\bar{l})$, and we can set $g(a, x, l) = \{(c)_1\}(\Lambda v.2^{l_1} \cdot 3^{2^d \cdot 3^{g(a, \{x\}(v), l_1)}})$.

Let for every $\delta < \gamma = |m|$ and every l being the Gödel number of an LA_β-proposition of rank δ the values $f(a, x, l)$ and $g(a, x, l)$ be defined in such a way that (11) and (12) hold. Let l be the Gödel number of an LA_β-proposition Ψ whose rank does not exceed γ. We have to define $f(a, x, l)$ and $g(a, x, l)$ in such a way that (11) and (12) hold. We use induction on the number of logical connectives and quantifiers in Ψ.

The case of an arithmetical atomic proposition Ψ was considered above. Let Ψ be an atomic LA_γ-proposition of the form $T(\bar{a}, \bar{b})$. Then $|a| = \delta < \gamma$. Note that Ψ is also an LA_β-proposition, thus $\pi(\bar{n}, \bar{l})$ is realizable. Moreover, we have a realization of $\theta(\bar{l}, \bar{a}, \bar{b})$. If b is not a Gödel number of any LA_δ-proposition, then $T(\bar{a}, \bar{b})$ is not realizable, therefore $T(\bar{n}, \bar{l})$ is not realizable. On the other hand, $\neg\pi(\bar{a}, \bar{b})$ is realizable. As (3) is a conjunct in $[R, w]$, the formula $\neg\Phi(\bar{l})$ is realizable and $\Phi(\bar{l})$ is not realizable. Thus we can set $f(a, x, l) = g(a, x, l) = x$.

Let b be the Gödel number of an LA_δ-proposition. Then we have a realization of $\pi(\bar{a}, \bar{b})$. Note that in this case, the rank of the proposition with the Gödel number b does not exceed δ and is less than γ. By inductive hypothesis, $f(a, x, b)$ and $g(a, x, b)$ are defined in such a way that

$$\forall l, x\, [x\, \mathbf{r}\, \Phi(\bar{b}) \Rightarrow f(a, x, b)\, \mathbf{r}\, T(\bar{n}, \bar{b})]; \tag{20}$$

$$\forall l, x\, [x\, \mathbf{r}\, T(\bar{n}, \bar{b}) \Rightarrow g(a, x, b)\, \mathbf{r}\, \Phi(\bar{b})]. \tag{21}$$

As a realization a of $[\Phi(x), \bar{n}]$ is given and (4) is a conjunct in $[R, w]$, we can find a realization c of $\Phi(\bar{l}) \equiv \Phi(\bar{b})$. Let $x\, \mathbf{r}\, \Phi(\bar{l})$, then $\{(c)_0\}(x)\, \mathbf{r}\, \Phi(\bar{b})$. It is easy to prove that $f(a, \{(c)_0\}(x), b)\, \mathbf{r}\, T(\bar{n}, \bar{b})$. Thus $f(a, \{(c)_0\}(x), b)\, \mathbf{r}\, T(\bar{n}, \bar{l})$ and we can set $f(a, x, l) = f(a, \{(c)_0\}(x), b)(\{(c)_0\}(x), b)$.

Now let $x\, \mathbf{r}\, T(\bar{n}, \bar{l})$. This means $x\, \mathbf{r}\, T(\bar{a}, \bar{b})$, i. e., x is a realization of the LA_δ-proposition with the Gödel number b. Then $x\, \mathbf{r}\, T(\bar{n}, \bar{b})$. By (21), $g(a, x, b)\, \mathbf{r}\, \Phi(\bar{b})$ and $\{(c)_1\}(g(a, x, b))\, \mathbf{r}\, \Phi(\bar{l})$. Thus we can set $g(a, x, l) = \{(c)_1\}(g(a, x, b))$.

The cases when Ψ is of the form $\neg\Psi_0$, $(\Psi_0\, \&\, \Psi_1)$, $(\Psi_0 \vee \Psi_1)$, $(\Psi_0 \supset \Psi_1)$, $\exists v\, \Psi_0(v)$, or $\forall v\, \Psi_0(v)$ were considered above. Thus we have defined partial recursive functions f and g satisfying (11) and (12). Then

$$\Lambda x.2^{\Lambda y.f(a,y,x)} \cdot 3^{\Lambda y.g(a,y,x)}\, \mathbf{r}\, \forall x\, (\Phi(x) \equiv T(\bar{n}, x)).$$

Theorem 2 is proved.

5 Predicate Realizability Logics

Theorem 3. *Let $\beta \leq \alpha$. For every LA_β-proposition Ψ one can effectively construct a closed predicate formula Ψ^* such that*
1) if Ψ^* is effectively LA_β-realizable, then Ψ is realizable;
2) if Ψ is realizable, then Ψ^* is uniformly LA_β-realizable.

Proof. By Theorem on schemata (Theorem 1), it is sufficient for every LA_β-proposition Ψ to construct a closed scheme $\mathcal{A}$ such that
1) if $\mathcal{A}$ is effectively LA_β-realizable, then Ψ is realizable;
2) if Ψ is realizable, then $\mathcal{A}$ is uniformly LA_β-realizable.

Let $\beta \leq \alpha$ and Ψ be an LA_β-proposition. As for any natural m and a term t the universal closure of the formula $T(\bar{m}, t) \equiv \exists x\,(t = x \,\&\, T(\bar{m}, x))$ is obviously realizable, Ψ is equivalent to a formula containing T only in atomic formulas of the form $T(\bar{m}, x)$, where $|m| < \beta$ and x is a variable. Suppose that Ψ already has this property. As Ψ contains only a finite number of occurences of the subformulas of the form $T(\bar{m}, x)$, we can find the greatest among such ms relative to the order $\prec$; denote it by M. Replace every atomic subformula of the form $T(\bar{m}, x)$ in Ψ by $\pi(\bar{m}, x) \,\&\, T(\bar{M}, x)$. As the formulas $T(\bar{m}, x)$ and $\pi(\bar{m}, x) \,\&\, T(\bar{M}, x)$ are evidently equivalent, we obtain the formula equivalent to Ψ. Finally, replace all the occurences of $T(\bar{M}, x)$ in the obtained formula by $R(x)$, where R is a one-place predicate variable. We obtain a scheme denoted by $\hat{\Psi}(R)$.

Proposition 3. *If the formula Ψ is realizable, then the scheme $[R, \bar{M}] \supset \hat{\Psi}(R)$ is uniformly LA_β-realizable.*

Proof. Let a be a realization of Ψ and $\Phi(x)$ be an LA_β-formula. Let a realization b of $[\Phi(x), \bar{M}]$ be given. By Theorem 2, $h(b)\,\mathbf{r}\,\forall x\,(\Phi(x) \equiv T(\bar{M}, x))$. By means of arguments usual for theorems on logically equivalent formulas one can find a realization c of the formula $\hat{\Psi}(\Phi(x)) \equiv \hat{\Psi}(T(\bar{M}, x))$ independent on Φ. Note that $\hat{\Psi}(T(\bar{M}, x))$ is Ψ, thus a is a realization of the formula $\hat{\Psi}(T(\bar{M}, x))$. Therefore $\{(c)_1\}(a)\,\mathbf{r}\,\hat{\Psi}(\Phi(x))$. Thus $\lambda a.\{(c)_1\}(a)\,\mathbf{r}\,[\Phi(x), \bar{M}] \supset \hat{\Psi}(\Phi(x))$. Note that this realization of $[\Phi(x), \bar{M}] \supset \hat{\Psi}(\Phi(x))$ does not depend on Φ. Therefore the scheme $\Phi^*(R)$ is uniformly LA_β-realizable. Proposition 3 is proved.

Proposition 4. *If the scheme $[R, \bar{M}] \supset \hat{\Psi}(R)$ is effectively LA_β-realizable, then the formula Ψ is realizable.*

Proof. Let the scheme $[R, \bar{M}] \supset \hat{\Psi}(R)$ be effectively LA_β-realizable. Then we can find a realization b of $[T(\bar{M}, x), \bar{M}] \supset \hat{\Psi}(T(\bar{M}, x))$. Note that the formula $[T(\bar{M}, x), \bar{M}]$ is realizable, let a be its realization. Then $\{b\}(a)\,\mathbf{r}\,\hat{\Psi}(T(\bar{M}, x))$. As $\hat{\Psi}(T(\bar{M}, x))$ is Ψ, the formula Ψ is realizable. Proposition 4 is proved.

Propositions 3 and 4 mean that 1) if the scheme $[R, \bar{M}] \supset \hat{\Psi}(R)$ is effectively LA_β-realizable, then the formula Ψ is realizable, and 2) if the formula Ψ is realizable, then the scheme $[R, \bar{M}] \supset \hat{\Psi}(R)$ is uniformly LA_β-realizable. Theorem 3 is proved.

Theorem 4. *The classes of effectively and uniformly LA_β-realizable predicate formulas are not definable in the language LA_β.*

Proof. Note that the predicate formula Ψ^* constructed for an LA_β-proposition Ψ according to Theorem 3 has the following proprety: 1) if Ψ is realizable, then Ψ^* is both effectively and uniformly LA_β-realizable; 2) if Ψ is not realizable, then Ψ^* is neither effectively nor uniformly realizable. Let φ be a partial recursive function such that forall x, if x is the Gödel number of an LA_β-proposition Ψ, then $\varphi(x)$ is the Gödel number of the predicate formula Ψ^*. Now let $\Phi(x)$ be an LA_β-formula such that for every n the formula $\Phi(\bar{n})$ is realizable iff n is the Gödel number of an effectively (uniformly) LA_β-realizable predicate formula. Then evidently the LA_β-formula $\exists y\,(y = \varphi(x)\,\&\,\Phi(y))$ defines the predicate $\mathbf{R}_\beta(x)$ in the language LA_β, but this is impossible by Proposition 1. Theorem 4 is proved.

Theorem 5. *If $\gamma < \beta \leq \alpha$, then there exists a closed uniformly LA_γ-realizable predicate formula which is not effectively LA_β-realizable.*

Proof. By Theorem 1, it is sufficient to construct a closed uniformly LA_γ-realizable scheme which is not effectively LA_β-realizable. Let $|m| = \gamma < \beta$. We prove that $\neg[R, \bar{m}]$ is a required scheme. As it was noted above, the formula $[T(\bar{m}, x), \bar{m}]$ is realizable, thus the formula $\neg[T(\bar{m}, x), \bar{m}]$ is not realizable. As $T(\bar{m}, x)$ is an LA_β-formula, this means that the scheme $\neg[R, \bar{m}]$ is not effectively LA_β-realizable. On the other hand, if for an LA_γ-formula $\Phi(x)$, the formula $[\Phi(x), \bar{m}]$ is realizable, then by Proposition 2, the formulas $\Phi(x)$ and $T(\bar{m}, x)$ are equivalent. Thus $\Phi(x)$ defines the predicate $\mathbf{R}_\gamma(x)$ in the language LA_γ, but this is impossible by Proposition 1.

References

1. Kleene, S.C.: On the interpretation of intuitionistic number theory. Journal of Symbolic Logic 10, 109–124 (1945)
2. Plisko, V.E.: On realizable predicate formulae (Russian). Doklady Akademii Nauk SSSR 212, 553–556 (1973); Translation Soviet Math. Dokl. 14, 1420–1424
3. Plisko, V.E.: The nonarithmeticity of the class of realizable predicate formulas (Russian). Izvestiya Akademii Nauk SSSR. Seriya Matematicheskaya 41, 483–502 (1977); Translation Math. USSR Izvestiya 11, 453–471
4. Plisko, V.E.: Recursive realizability and constructive predicate logic (Russian). Doklady Akademii Nauk SSSR 214, 520–523 (1974); Translation Soviet Math. Dokl. 15, 193–197
5. Plisko, V.E.: Absolute realizability of predicate formulas (Russian). Izvestiya Akademii Nauk SSSR. Seriya Matematicheskaya 47, 315–334 (1983); Translation Math. USSR Izvestiya 22, 291–308
6. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill Book Company, New York (1967)
7. Plisko, V.E.: Some variants of the notion of realizability for predicate formulas (Russian). Izvestiya Akademii Nauk SSSR. Seriya Matematicheskaya 42, 637–653 (1978); Translation Math. USSR Izvestiya 12, 588–604

The Quantitative Analysis of User Behavior Online — Data, Models and Algorithms

(Invited Talk)

Prabhakar Raghavan

Yahoo! Labs, Santa Clara, USA

Abstract. By blending principles from mechanism design, algorithms, machine learning and massive distributed computing, the search industry has become good at optimizing monetization on sound scientific principles. This represents a successful and growing partnership between computer science and microeconomics. When it comes to understanding how online users respond to the content and experiences presented to them, we have more of a lacuna in the collaboration between computer science and certain social sciences. We will use a concrete technical example from image search results presentation, developing in the process some algorithmic and machine learning problems of interest in their own right. We then use this example to motivate the kinds of studies that need to grow between computer science and the social sciences; a critical element of this is the need to blend large-scale data analysis with smaller-scale eye-tracking and "individualized" lab studies.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, p. 327, 2010.

A Faster Exact Algorithm for the Directed Maximum Leaf Spanning Tree Problem

Daniel Binkele-Raible and Henning Fernau

Univ.Trier, FB 4—Abteilung Informatik, 54286 Trier, Germany
{raible,fernau}@informatik.uni-trier.de

Abstract. Given a directed graph $G = (V, A)$, the Directed Maximum Leaf Spanning Tree problem asks to compute a directed spanning tree with as many leaves as possible. By designing a branching algorithm analyzed with *Measure&Conquer*, we show that the problem can be solved in time $\mathcal{O}^*(1.9044^n)$ using polynomial space. Allowing exponential space, this run time upper bound can be lowered to $\mathcal{O}^*(1.8139^n)$.

1 Introduction

We investigate the Directed Maximum Leaf Spanning Tree (DMLST) problem, where we are given a directed graph $G(V, A)$, and we are asked to find a directed spanning tree (with the arcs directed from the root towards the leaves) for G with a maximum number of leaves. Such a directed spanning tree is sometimes called an *out-branching*.

Known Results. Besides results on approximability [3,4], this problem has also drawn notable attention in the field of parameterized algorithms. Here the problem is known as directed k-leaf spanning tree where k is a lower bound on the number of leaves in the directed spanning tree. For kernelization results, we refer to [3,5]. Branching algorithms both for directed and undirected graphs have been developed, leading to the run times summarized in Table 1. All these algorithms are based on an idea of J. Kneis, A. Langer and P. Rossmanith [8]; they solved this problem in time $\mathcal{O}^*(4^k)$. The given run times marked with (*) follow from the row above using an observation of V. Raman and S. Saurabh [11].

Our Achievements. The main result in this paper improves the currently best upper bound of $\mathcal{O}^*(1.9973^n)$ [2]. Our algorithm is inspired by the one of [6] and achieves a run time of $\mathcal{O}^*(1.9044^n)$. The algorithm of [6] is not trivially transferrable to the directed version. Starting from an initial root the algorithm grows a tree T. The branching process takes place by deciding whether the vertices neighbored to the tree will become final leaves or internal vertices. A crucial ingredient of the algorithm was also to create *floating leaves*, i.e., vertices which are final leaves in the future solution but yet not attached to T. This concept has been already used in [6] and partly by [2]. In the undirected case we guarantee that in the bottleneck case we can generate at least two such leaves. In the directed version there is a situation where only one can be created. Especially for this problem we had to find a workaround.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 328–339, 2010.

Table 1. Records for producing leafy trees; writing $\mathcal{O}^*()$ suppresses polynomial factors

	directed graphs	undirected graphs	randomized (undirected graphs)
parameterized	$\mathcal{O}^*(3.72^k)$ [2]	$\mathcal{O}^*(3.46^k)$ [10]	$\mathcal{O}^*(2^k)$ [9]
exact	$\mathcal{O}^*(1.9973^n)$ [2] (*)	$\mathcal{O}^*(1.8962^n)$ [6,10]	$\mathcal{O}^*(1.7088^n)$ (*)

Preliminaries, Terminology & Notation. We consider directed graphs $G(V, A)$ in the course of our algorithm, where V is the vertex set and A the arc set. The *in-neighborhood* of a vertex $v \in V$ is $N^-_{V'}(v) = \{u \in V' \mid (u, v) \in A\}$ and, analogously, its *out-neighborhood* is $N^+_{V'}(v) := \{u \in V' \mid (v, u) \in A\}$. The *in- and out-degrees* of v are $d^-_{V'}(v) := |N^-_{V'}(v)|$ and $d^+_{V'}(v) := |N^+_{V'}(v)|$ and its *degree* is $d_{V'}(v) = d^-_{V'}(v) + d^+_{V'}(v)$. If $V' = V$, then we might suppress the subscript. For $V' \subseteq V$ we let $N^+(V') := \bigcup_{v \in V'} N^+(v)$ and $N^-(V')$ is defined analogously. Let $A(V') := \{(u, v) \in A \mid \exists u, v \in V'\}$, $N^+_A(v) := \{(v, u) \in A \mid u \in N^+_V(v)\}$ and $N^-_A(v) := \{(u, v) \in A \mid u \in N^-_V(v)\}$. Given a graph $G = (V, A)$ and a graph $G' = (V', A')$, G' is a *subgraph* of G if $V' \subseteq V$ and $A' \subseteq A$. The subgraph of G *induced by* a vertex set $X \subseteq V$ is denoted by $G(X)$ and is defined by $G(X) = (X, A')$ where $A' = A(X)$. The subgraph of G induced by an arc set $Y \subseteq A$ is denoted by $G(Y)$ and is defined by $G(Y) = (V(Y), Y)$ where $V(Y) = \{u \in V \mid \exists (u, v) \in Y \vee \exists (v, u) \in Y\}$.

A *directed path* of length ℓ in G is a set of pairwise different vertices $v_1, \ldots, v_\ell$ such that $(v_i, v_{i+1}) \in A$ for $1 \leq i < \ell$. A subgraph $H(V_H, A_H)$ of G is called a *directed tree* if there is a unique root $r \in V_H$ such that there is a unique directed path P from r to every $v \in V_H \setminus \{r\}$ under the restriction that its arc set obeys $A(P) \subseteq A_H$. Speaking figuratively, in a directed tree the arcs are directed from the parent to the child. If for a directed tree $H = (V_H, A_H)$ that is a subgraph of $G(V, A)$ we have $V = V_H$, we call it *spanning directed tree* of G. The terms *out-tree* and *out-branching* are sometimes used for directed tree and spanning directed tree, respectively. The *leaves* of a directed tree $H = (V_H, A_H)$ are the vertices u such that $d^-_{V_H}(u) = d_{V_H}(u) = 1$. In $leaves(H)$ all leaves of a tree H are comprised and $internal(H) := V(H) \setminus leaves(H)$. The unique vertex v such that $N^-_{V_H}(u) = \{v\}$ for a tree-vertex will be called *parent* of u. A vertex $v \in V_H$ such that $d_{V_H}(v) \geq 2$ will be called *internal*. Let $T(V_T, A_T)$ and $T'(V_{T'}, A_{T'})$ be two trees. T' *extends* T, written $T' \succeq T$, iff $V_T \subseteq V_{T'}$, $A_T \subseteq A_{T'}$. Simplistically, we will consider a tree T also as a set of arcs $T \subseteq A$ such that $G(T)$ is a directed tree. The notions of $\succeq$ and $leaves(T)$ carry over canonically.

An *arc-cut set* is a set of arcs $B \subset A$ such that $G(A \setminus B)$ is a digraph which is not weakly connected. We suppose that $|V| \geq 2$. The function $\chi()$ returns 1 if its argument evaluates to true and 0 otherwise. We now re-define our problem:
Rooted Directed Maximum Leaf Spanning Tree (RDMLST)
Given: A directed graph $G(V, A)$ and a vertex $r \in V$.
Task: Find a spanning directed tree $T' \subseteq A$ such that $|leaves(T')|$ is maximum and $d^-_{T'}(r) = 0$.

Basic Idea of the Algorithm. Once we have an algorithm for RDMLST, this can be used to solve DMLST. As a initial step we simply consider every vertex as a possible root r of the final solution. This yields a total of n cases.

Then in the course of the algorithm for RDMLST we will gradually extend an out-tree $T \subseteq A$, which is predetermined to be a subgraph in the final out-branching. Let $V_T := V(T)$ and $\overline{V}_T := V \setminus V_T$. We will also maintain a mapping $\text{lab} : V \to \{\text{free}, \text{IN}, \text{LN}, \text{BN}, \text{FL}\} =: D$, which assigns different roles to the vertices. If $\text{lab}(v) = \text{IN}$, then v is already fixed to be internal, if $\text{lab}(v) = \text{LN}$ then it will be a leaf. Vertices in IN or LN will also be called *internal nodes* or *leaf nodes*, respectively. If $\text{lab}(v) = \text{BN}$ (designating a *branching node*), then v already has a parent in T, but can be leaf or internal in the final solution. If $\text{lab}(v) = \text{FL}$, then v is constrained to be a leaf but has not yet been attached to the tree T. Such vertices are called *floating leaves*. If $\text{lab}(v) = \text{free}$, then $v \notin V_T$ and nothing has been fixed for v yet. For a *label* $Z \in D$ and $v \in V$ we will often write $v \in Z$ when we mean $\text{lab}(v) = Z$. A given tree T' defines a labeling $V_{T'} \to D$ to which we refer by $\text{lab}_{T'}$. Let $\text{IN}_{T'} := \{v \in V_{T'} \mid d^+_{T'}(v) \geq 1\}$, $\text{LN}_{T'} := \{v \in V_{T'} \mid d^+_G(v) = 0\}$ and $\text{BN}_{T'} = V_{T'} \setminus (\text{IN}_{T'} \cup \text{LN}_{T'}) = \{v \in V_{T'} \mid d^+_{T'}(v) = 0 < d^+_G(v)\}$. Then for any $ID \in D \setminus \{\text{FL}, \text{free}\}$ we have $ID_{T'} = \text{lab}^{-1}(ID)$. We always assure that lab_T and lab are the same on V_T. The subscript might be hence suppressed if $T' = T$. If $T' \succ T$, then we assume that $\text{IN}_T \subseteq \text{IN}_{T'}$ and $\text{LN}_T \subseteq \text{LN}_{T'}$. So, the labels IN and LN remain once they are fixed. For the remaining labels we have the following possible transitions: $\text{FL} \to \text{LN}$, $\text{BN} \to \{\text{LN}, \text{IN}\}$ and $\text{free} \to D \setminus \{\text{free}\}$. Let $\text{BN}_i = \{v \in \text{BN} \mid d^+_G(v) = i\}$, $\text{free}_i = \{v \in \text{free} \mid d^-_G(v) = i\}$ for $i \geq 1$, $\text{BN}_{\geq \ell} := \bigcup_{j=\ell}^{n} \text{BN}_j$ and $\text{free}_{\geq \ell} := \bigcup_{j=\ell}^{n} \text{free}_j$.

Reduction & Halting Rules. We state a set of seven reduction & halting rules. Similar reduction rules for the undirected version can be found in [6,10].

(H) Halt if $\text{BN} = \emptyset$. If $\text{free} \cup \text{FL} = \emptyset$ then return $|\text{LN}|$. Otherwise, answer NO.
(R1) Let $v \in V$. If $\text{lab}(v) = \text{FL}$, then remove $N^+_A(v)$. If $\text{lab}(v) = \text{BN}$, then remove $N^-_A(v) \setminus T$.
(R2) If there exists a vertex $v \in \text{BN}$ with $d^+(v) = 0$, then set $\text{lab}(v) := \text{LN}$.
(R3) If there exists a vertex $v \in \text{free}$ with $d(v) = 1$, then set $\text{lab}(v) := \text{FL}$.
(R4) If $v \in \text{LN}$, then remove $N_A(v) \setminus T$.
(R5) Assume that there exists some $u \in \text{BN}$ such that $N^+_A(u)$ is a an arc-cut set. Then $\text{lab}(u) := \text{IN}$ and for all $x \in N^+(u) \cap \text{FL}$ set $\text{lab}(x) := \text{LN}$, and for all $x \in N^+(u) \cap \text{free}$ set $\text{lab}(x) := \text{BN}$.
(R6) If there is an arc $(a, b) \in A$ with $a, b \in \text{free}$ and $G(A \setminus \{a, b\})$ consists of two strongly connected components of size greater than one. Then contract (a, b) and label the resulting vertex "free".

We mention that we defer the correctness proof for **(H)** to section 2.2. Due to space restrictions, the proof of **(R1)**-**(R3)** is omitted.

Proposition 1. *The reduction rules* **(R1)** - **(R6)** *are sound.*

Proof. **(R4)** The only arcs present in any tree $T' \succeq T$ will be $N_A(v) \cap T$. Thus, $N_A(v) \setminus T$ can be removed.

(R5) As $N_A^+(v)$ is an arc-cut set, setting $v \in \text{LN}$ would cut off a component which cannot be reached from the root r. Thus, $v \in \text{IN}$ is constrained.

(R6) Let G^* be the graph after contracting (a, b). If G^* has a spanning tree with k leaves, then so does G. On the other hand, note that in every spanning tree $T' \succeq T$ for G we have that $a, b \in \text{IN}$ and $(a, b) \in T'$. Hence, the tree $T^\#$ evolved by contracting (h, u) in T' is a spanning tree with k leaves in G^*. □

2 The Algorithm

2.1 Branching Rules

If $N^+(\mathit{internal}(T)) \subseteq \mathit{internal}(T) \cup \mathit{leaves}(T)$, we call T an *inner-maximal* directed tree. We make use of the following fact:

Lemma 1 ([8] Lemma 4.2). *If there is a tree T' with $\mathit{leaves}(T') \geq k$ such that $T' \succeq T$ and $x \in \mathit{internal}(T')$, then there is a tree T'' with $\mathit{leaves}(T'') \geq k$ such that $T'' \succeq T$, $x \in \mathit{internal}(T'')$ and $\{(x, u) \in A \mid u \in V\} \subseteq T''$.*

Look at Algorithm 1 which describes the branching rules. As mentioned before, the search tree evolves by branching on BN-vertices. For some $v \in \text{BN}$ we will set either $\text{lab}(v) = \text{LN}$ or $\text{lab}(v) = \text{IN}$. In the second case we adjoin the vertices $N_A^+(v) \setminus T$ as BN-nodes to the partial spanning tree T. This is justified by Lemma 1. Thus, during the whole algorithm we only consider inner-maximal trees. Right in the beginning we therefore have $A(\{r\} \cup N^+(r))$ as an initial tree where r is the vertex chosen as the root.

We also introduce an abbreviationfor the different cases generated by branching: $\langle v \in \text{LN}; v \in \text{IN}\rangle$ means that we recursively consider the two cases were v becomes a leaf node and an internal node. The semicolon works as a delimiter between the different cases. More complicated expressions like $\langle v \in \text{BN}, x \in \text{BN}; v \in \text{IN}, x \in \text{LN}; v \in \text{LN}\rangle$ describe more complicated branchings.

2.2 Correctness of the Algorithm

In the following we are going to prove two lemmas which are crucial for the correctness and for the run time.

Lemma 2 (Correctness of (H)). *If $BN = \emptyset$ and $\mathit{free} \cup \mathit{FL} \neq \emptyset$, then no spanning tree $T' \succeq T$ exists.*

Proof. Let $x \in \text{free} \cup \text{FL}$ and assume there is a spanning tree $T' \succeq T$. Then there is a directed path $P = r \ldots x$ in $G(T')$. By $\text{free} \cup \text{FL} \neq \emptyset$ and $T' \supseteq T$, there must be a $(a, b) \in A(P)$ such that we can assume that $a \in V(T)$ and $b \notin V(T)$. By the fact $\text{BN} = \emptyset$ and **(R4)** it follows that $a \in \text{IN}_T$. But this is a contradiction to Lemma 1, i.e., T would not be inner-maximal. □

Lemma 3. *Let $T \subseteq A$ be a given tree such that $v \in BN_T$ and $N^+(v) = \{x_1, x_2\}$. Let $T', T^* \subseteq A$ be optimal solutions with $T', T^* \succeq T$ under the restriction that $lab_{T'}(v) = LN$, and $lab_{T^*}(v) = IN$ and $lab_{T^*}(x_1) = lab_{T^*}(x_2) = LN$.*

Data: A directed graph $G = (V, A)$ and a tree $T \subseteq A$.
Result: A spanning tree T' with the maximum number of leaves s.t. $T' \succeq T$.
Check if the halting rule **(H)** applies .
Apply the reduction rules exhaustively.
if $BN_1 \neq \emptyset$ **then**
  Choose some $v \in \mathrm{BN}_1$.
  Let $P = \{v_0, v_1, \ldots, v_k\}$ be a path of maximum length such that (1) $v_0 = v$, (2) for all $1 \leq i \leq k-1$, $d^+_{\overline{P_{i-1}}}(v_i) = 1$ (where $P_{i-1} = \{v_0, \ldots, v_{i-1}\}$) and (3) $P \setminus \mathrm{free} \subseteq \{v_0, v_k\}$.
  if $d^+_{\overline{P_{k-1}}}(v_k) = 0$ **then**
    Put $v \in \mathrm{LN}$. (B1)
  else
    $\langle v \in \mathrm{IN}, v_1, \ldots, v_k \in \mathrm{IN}; v \in \mathrm{LN} \rangle$ (B2)
else
  Choose a vertex $v \in \mathrm{BN}$ with maximum out-degree.
  if *a)* $d^+(v) \geq 3$ ***or*** *b)*$(N^+(v) = \{x_1, x_2\}$ *and* $N^+(v) \subseteq FL)$ **then**
    $\langle v \in \mathrm{IN}; v \in \mathrm{LN} \rangle$ and in case *b)* apply `makeleaves`(v, x_1, x_2) in the 1st branch. (B3)
  else if $N^+(v) = \{x_1, x_2\}$ **then**
    if *for* $z \in (\{x_1, x_2\} \cap free)$ *we have*
    $|N^+(z) \setminus N^+(v)| = 0$ ***or*** *(B4.1)*
    $N^+_A(z)$ *is an arc-cut set* **or** *(B4.2)*
    $N^+(z) \setminus N^+(v) = \{v_1\}$ *(B4.3)*
    then
      $\langle v \in \mathrm{IN}; v \in \mathrm{LN} \rangle$ (B4)
  else if $N^+(v) = \{x_1, x_2\}$, $x_1 \in free, x_2 \in FL$ **then**
    $\langle v \in \mathrm{IN}, x_1 \in \mathrm{IN}; v \in \mathrm{IN}, x_1 \in \mathrm{LN}; v \in \mathrm{LN} \rangle$
    and apply `makeleaves`(v, x_1, x_2) in the 2nd branch. (B5)
  else if $N^+(v) = \{x_1, x_2\}$, $x_1, x_2 \in free$ *and* $\exists z \in (N^-(x_1) \cap N^-(x_2)) \setminus \{v\}$ **then**
    $\langle v \in \mathrm{IN}, x_1 \in \mathrm{IN}; v \in \mathrm{IN}, x_1 \in \mathrm{LN}, x_2 \in \mathrm{IN}; v \in \mathrm{LN} \rangle$ (B6)
  else if $N^+(v) = \{x_1, x_2\}$, $x_1, x_2 \in free$ *and* $|(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}| \geq 2$ **then**
    $\langle v \in \mathrm{IN}, x_1 \in \mathrm{IN}; v \in \mathrm{IN}, x_1 \in \mathrm{LN}, x_2 \in \mathrm{IN}; v \in \mathrm{IN}, x_1 \in \mathrm{LN}, x_2 \in \mathrm{LN}; v \in \mathrm{LN} \rangle$ and apply `makeleaves`(v, x_1, x_2) in the 3rd branch. (B7)
  else
    $\langle v \in \mathrm{IN}; v \in \mathrm{LN} \rangle$ (B8)

Algorithm 1. An Algorithm for solving RDMLST

begin
  $\forall u \in [(N^-(x_1) \cup N^-(x_2)) \setminus \{x_1, x_2, v\}] \cap$ free set $u \in \mathrm{FL}$;
  $\forall u \in [(N^-(x_1) \cup N^-(x_2)) \setminus \{x_1, x_2, v\}] \cap$ BN set $u \in \mathrm{LN}$;
end

Procedure `makeleaves`(v, x_1, x_2)

1. *If there is a vertex* $u \neq v$ *with* $N^+(u) = \{x_1, x_2\}$*, then* $|leaves(T')| \geq |leaves(T^*)|$.
2. *Assume that* $d^-(x_i) \geq 2$ $(i = 1, 2)$*. Assume that there exists some* $u \in (N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}$ *with* $lab_{T^*}(u) = IN$*. Then* $|leaves(T')| \geq |leaves(T^*)|$.

Proof. 1. Let $T^+ := (T^* \setminus \{(v, x_1), (v, x_2)\}) \cup \{(u, x_1), (u, x_2)\}$. We have $\text{lab}_{T^+}(v) = \text{LN}$ and u is the only vertex besides v where $\text{lab}_{T^*}(u) \neq \text{lab}_{T^+}(u)$ is possible. Hence, u is the only vertex where we could have $\text{lab}_{T^*}(u) = \text{LN}$ such that $\text{lab}_{T^+}(u) = \text{IN}$. Thus, we can conclude $|leaves(T^+)| \geq |leaves(T^*)|$. As T' is optimal under the restriction that $v \in \text{LN}$, $|leaves(T')| \geq |leaves(T^+)| \geq |leaves(T^*)|$ follows.

2. W.l.o.g., we have $u \in N^-(x_1) \setminus \{v, x_2\}$. Let $q \in N^-(x_2) \setminus \{v\}$ and $T^+ := (T^* \setminus \{(v, x_1), (v, x_2)\}) \cup \{(u, x_1), (q, x_2)\}$. We have $\text{lab}_{T^+}(v) = \text{LN}$, $\text{lab}_{T^+}(u) = \text{lab}_{T^*}(u) = \text{IN}$ and q is the only vertex besides v where we could have $\text{lab}_{T^*}(q) \neq \text{lab}_{T^+}(q)$ (i.e., possibly $\text{lab}_{T^*}(q) = \text{LN}$ and $\text{lab}_{T^+}(q) = \text{IN}$). Therefore, $|leaves(T')| \geq |leaves(T^+)| \geq |leaves(T^*)|$. □

Correctness of the Different Branching Cases. First note that by Lemma 2 **(H)** takes care of the case that indeed an out-branching has been built. If so, the number of its leaves is returned. Below we will argue that each branching case of Algorithm 1 is correct: it preserves at least one optimal solution. Cases (B3)a), (B4) and (B8) do not have to be considered in detail, being simple binary, exhaustive branchings.

(B1) Suppose there is an optimal extension $T' \supseteq T$ such that $\text{lab}_{T'}(v) = \text{lab}_{T'}(v_0) = \text{IN}$. Due to the structure of P, there must be an i, $0 < i \leq k$, such that $(v_j, v_{j-1}) \in T'$ for $0 < j \leq i$, i.e., $v, v_1, \ldots v_{i-1} \in \text{IN}$ and $v_i \in \text{LN}$. W.l.o.g., we choose T' in a way that i is minimum but T' is still optimal (✢). By **(R5)** there must be a vertex v_z, $0 < z \leq i$, such that there is an arc (q, v_z) with $q \in V_{T'} \setminus P$. Now consider $T'' = (T' \setminus \{(v_{z-1}, v_z)\}) \cup \{q, v_z\}$. In T'' the vertex v_{z-1} is a leaf and therefore $|leaves(T'')| \geq |leaves(T')|$. Additionally, we have that $z - 1 < i$ which is a contradiction to the choice of T' (✢).

(B2) Note that $\text{lab}(v_k) \in \{\text{BN}, \text{FL}\}$ is not possible due to **(R1)** and, thus, $\text{lab}(v_k) = \text{free}$. By the above arguments from **(B1)**, we can exclude the case that $v, v_1, \ldots v_{i-1} \in \text{IN}$ and $v_i \in \text{LN}$ $(i \leq k)$. Thus, under the restriction that we set $v \in \text{IN}$, the only remaining possibility is also to set $v_1, \ldots v_k \in \text{IN}$.

(B3)b) When we set $v \in \text{IN}$, then the two vertices in $N^+(v)$ will become leaf nodes (i.e., become part of LN). Thus, Lemma 3.2 applies. Note that **(R5)** does not apply and therefore $(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\} \neq \emptyset$, as well as $d(x_i) \geq 2$ $(i = 1, 2)$. This means that that every vertex in $(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}$ can be assumed a to be leaf node in the final solution. This justifies to apply `makeleaves`(v, x_1, x_2).

(B5) The branching is exhaustively with respect to v and x_1. Nevertheless, in the second branch, `makeleaves`(v, x_1, x_2) is carried out. This is justified by

Lemma 3.2 (similar to **(B3)b)**), as by setting $v \in \text{IN}$ and $x_1 \in \text{LN}$, x_2 will be attached to v as a LN-node and **(R5)** does not apply.

(B6) In this case, we neglect the possibility that $v \in \text{IN}, x_1, x_2 \in \text{LN}$. But due to Lemma 3.1, a no worse solution can be found in the recursively considered case where we set $v \in \text{LN}$. This shows that the considered cases are sufficient.

(B7) Similar to (B3), Lemma 3.2 justifies applying $\texttt{makeleaves}(v, x_1, x_2)$ in the third branch.

2.3 Analysis of the Run Time

The Measure. To analyze the running-time we follow the *Measure&Conquer*-approach (see [7]) and use the following measure:

$$\mu(G) = \sum_{i=1}^{n} \epsilon_i^{\text{BN}} |\text{BN}_i| + \sum_{i=1}^{n} \epsilon_i^{\text{free}} |\text{free}_i| + \epsilon^{\text{FL}} |\text{FL}|$$

The concrete values are $\epsilon^{\text{FL}} = 0.2251$, $\epsilon_1^{\text{BN}} = 0.6668$, $\epsilon_i^{\text{BN}} = 0.7749$ for $i \geq 2$, $\epsilon_1^{\text{free}} = 0.9762$ and $\epsilon_2^{\text{free}} = 0.9935$. Also let $\epsilon_j^{\text{free}} = 1$ for $j \geq 3$ and $\eta = \min\{\epsilon^{\text{FL}}, (1-\epsilon_1^{\text{BN}}), (1-\epsilon_2^{\text{BN}}), (\epsilon_2^{\text{free}} - \epsilon_1^{\text{BN}}), (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}}), (\epsilon_1^{\text{free}} - \epsilon_1^{\text{BN}}), (\epsilon_1^{\text{free}} - \epsilon_2^{\text{BN}})\} = \epsilon_1^{\text{free}} - \epsilon_2^{\text{BN}} = 0.2013$.

For $i \geq 2$ let $\Delta_i^{\text{free}} = \epsilon_i^{\text{free}} - \epsilon_{i-1}^{\text{free}}$ and $\Delta_1^{\text{free}} = \epsilon_1^{\text{free}}$. Thus, $\Delta_{i+1}^{\text{free}} \leq \Delta_i^{\text{free}}$ with $\Delta_s^{\text{free}} = 0$ for $s \geq 4$.

In the very beginning, $\mu(G) \leq n = |V|$. The values for the different ϵ-variants have been chosen in order to optimize the outcome of the following analysis. Also note that if $\mu(G) = 0$ then **(H)** correctly returns the number of leaf nodes.

Run Time Analysis of the Different Branching Cases. In the following. we state for every branching case by how much μ will be reduced. Especially, Δ_i states the amount by which the i-th branch decreases μ. Each branching case may create several subcases in this analysis, depending on the sizes of the sets BN_i, free_i and FL before and after the branching. We solved each of the resulting recurrences by computing the largest positive roots (also known as *branching numbers*) of the associated characteristic polynomials. The largest of all these branching numbers is used in the formulation of Theorem 1 that summarizes our findings.

If v is the vertex chosen by Algorithm 1 then it is true that for all $x \in N^+(v)$ we have $d^-(x) \geq 2$ by **(R5)** (✤).

(B2) $\langle v \in \text{IN}, v_1, \ldots, v_k \in \text{IN}, v \in \text{LN} \rangle$

Recall that $d^+_{P_{k-1}}(v_k) \geq 2$ and $v_k \in \text{free}$ by **(R1)**. Then $v_1 \in \text{free}_{\geq 2}$ by **(R5)**.

1. v becomes IN-node; $v_1, \ldots, v_k$ become IN-nodes; the free vertices in $N^+(v_k)$ become BN-nodes, the FL-nodes in $N^+(v_k)$ become LN-nodes: $\Delta_1 \geq \epsilon_1^{\text{BN}} + \sum_{i=2}^{k} \epsilon_1^{\text{free}} + \chi(v_1 \in \text{free}_2) \cdot \epsilon_2^{\text{free}} + \chi(v_1 \in \text{free}_{\geq 3}) \cdot \epsilon_3^{\text{free}} + 2 \cdot \eta$
2. v becomes LN-node; the degree of v_1 is reduced: $\Delta_2 \geq \epsilon_1^{\text{BN}} + \sum_{i=2}^{3} \chi(v_1 \in \text{free}_i) \cdot \Delta_i^{\text{free}}$

(B3)a) Branching: $\langle v \in \mathrm{IN}; v \in \mathrm{LN}\rangle$.

1. v becomes IN-node; the free out-neighbors of v become BN-nodes; the FL out-neighbors of v becomes LN-nodes:
$\Delta_1 \geq \epsilon_2^{\mathrm{BN}} + \sum_{x\in N^+(v)\cap \mathrm{free}_{\geq 3}}(1-\epsilon_2^{\mathrm{BN}}) + \sum_{x\in N^+(v)\cap \mathrm{free}_2}(\epsilon_2^{\mathrm{free}} - \epsilon_2^{\mathrm{BN}}) + \sum_{y\in N^+(v)\cap \mathrm{FL}} \epsilon^{\mathrm{FL}}$
2. v becomes LN-node; the in-degree of the free out-neighbors of v is decreased: $\Delta_2 \geq \epsilon_2^{\mathrm{BN}} + \sum_{i=2}^{3} |N^+(v)\cap \mathrm{free}_i| \cdot \Delta_i^{\mathrm{free}}$

(B3)b) Recall that v is a BN-node of maximum out-degree, thus $d^+(z) \leq d^+(v) = 2$ for all $z \in \mathrm{BN}$. On the other hand $\mathrm{BN}_1 = \emptyset$ which implies $\mathrm{BN} = \mathrm{BN}_2$ from this point on. Hence, we have $N^+(v) = \{x_1, x_2\}$, $d^-(x_i) \geq 2$ $(i = 1, 2)$, and $|(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}| \geq 1$ by (✤) in the following branching cases. Therefore, the additional amount of $\min\{\epsilon_1^{\mathrm{free}} - \epsilon^{\mathrm{FL}}, \epsilon_2^{\mathrm{BN}}\}$ in the first branch is justified by the application of `makeleaves`(v, x_1, x_2). Note that by (✤) at least one free-node becomes a FL-node, or one BN-node becomes a LN-node. Also due to **(R1)** we have that $N^+(x_i) \cap \mathrm{BN} = \emptyset$.

1. v becomes IN-node; the FL out-neighbors of v become LN-nodes;
the vertices in $[(N^-(x_1)\cup N^-(x_2)) \setminus \{v, x_1, x_2\}] \cap \mathrm{BN}$ become LN-nodes;
the vertices in $[(N^-(x_1)\cup N^-(x_2)) \setminus \{v, x_1, x_2\}] \cap \mathrm{free}$ become FL-nodes.
$\Delta_1 \geq \epsilon_2^{\mathrm{BN}} + 2 \cdot \epsilon^{\mathrm{FL}} + \min\{\epsilon_1^{\mathrm{free}} - \epsilon^{\mathrm{FL}}, \epsilon_2^{\mathrm{BN}}\}$
2. v becomes LN-node: $\Delta_2 \geq \epsilon_2^{\mathrm{BN}}$.

(B4) The branching is binary: $\langle v \in \mathrm{IN}; v \in \mathrm{LN}\rangle$.

(B4.1)
1. v becomes IN-node; z becomes LN-node by **(R1)**, **(R2)** or both by **(R4)**. The vertex $q \in \{x_1, x_2\} \setminus \{z\}$ becomes LN-node or BN-node (depending on $q \in \mathrm{FL}$ or $q \in \mathrm{free}$).
$\Delta_1 \geq \epsilon_2^{\mathrm{BN}} + \epsilon_2^{\mathrm{free}} + \min\{\epsilon^{\mathrm{FL}}, (\epsilon_2^{\mathrm{free}} - \epsilon_2^{\mathrm{BN}})\}$
2. v becomes LN-node: $\Delta_2 \geq \epsilon_2^{\mathrm{BN}}$

(B4.2)
1. v becomes IN-node; $N_A^+(z)$ is an arc-cut set. Thus, z becomes IN-node as **(R5)** applies; The vertex $q \in \{x_1, x_2\}\setminus\{z\}$ becomes LN-node or BN-node (depending on $q \in \mathrm{FL}$ or $q \in \mathrm{free}$):
$\Delta_1 \geq \epsilon_2^{\mathrm{BN}} + \epsilon_2^{\mathrm{free}} + \min\{\epsilon^{\mathrm{FL}}, (\epsilon_2^{\mathrm{free}} - \epsilon_2^{\mathrm{BN}})\}$
2. v becomes LN-node: $\Delta_2 \geq \epsilon_2^{\mathrm{BN}}$.

Note that in all the following branching cases we have $N^+(x_i)\cap \mathrm{free}_1 = \emptyset$ $(i = 1, 2)$ by this case.

(B4.3) We have $|N^+(z) \setminus N^+(v)| = 1$. Thus, in the next recursive call after the first branch and the exhaustive application of **(R1)**, either **(R6)**, case (B2) or (B1) applies. **(R5)** does not apply due to (B4.2) being ranked higher. Note that the application of any other reduction rule does not change the situation. If (B2) applies we can analyze the current case together with its succeeding one. If (B2) applies in the case we set $v \in \mathrm{IN}$ we deduce that $v_0, v_1, \ldots, v_k \in \mathrm{free}$ where $z = v_0 = x_1$ (w.l.o.g., we assumed $z = x_1$). Observe that $v_1 \in \mathrm{free}_{\geq 2}$ as (B4.2) does not apply.
1. v becomes IN-node; x_1 becomes LN-node; x_2 becomes FL- or BN-node (depending on whether $x_2 \in \mathrm{free}$ or $x_2 \in \mathrm{FL}$); the degree of v_1 drops:
$\Delta_{11} \geq \epsilon_2^{\mathrm{BN}} + \chi(x_1 \in \mathrm{free}_{\geq 3}) \cdot \epsilon_3^{\mathrm{free}} + \chi(x_1 \in \mathrm{free}_2) \cdot \epsilon_2^{\mathrm{free}} +$

$\chi(x_2 \in \text{free}_{\geq 3}) \cdot (\epsilon_3^{\text{free}} - \epsilon_2^{\text{BN}}) + \chi(x_2 \in \text{free}_2) \cdot (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}}) +$
$\chi(x_2 \in \text{FL}) \cdot \epsilon^{\text{FL}} + \sum_{i=2}^{3} \chi(v_1 \in \text{free}_i) \cdot \Delta_i^{\text{free}}$.
2. v becomes IN-node, $x_1, v_1 \in \text{IN}, \ldots, v_k$ become IN-nodes; the free vertices in $N^+(v_k)$ become BN-nodes, the FL-nodes in $N^+(v_k)$ LN-nodes:
$\Delta_{12} \geq \epsilon_2^{\text{BN}} + \chi(x_1 \in \text{free}_{\geq 3}) \cdot \epsilon_3^{\text{free}} + \chi(x_1 \in \text{free}_2) \cdot \epsilon_2^{\text{free}} +$
$\chi(x_2 \in \text{free}_{\geq 3}) \cdot (\epsilon_3^{\text{free}} - \epsilon_2^{\text{BN}}) + \chi(x_2 \in \text{free}_2) \cdot (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}}) +$
$\chi(x_2 \in \text{FL}) \cdot \epsilon^{\text{FL}} + \chi(v_1 \in \text{free}_2) \cdot \epsilon_2^{\text{free}} + \chi(v_1 \in \text{free}_{\geq 3}) \cdot \epsilon_3^{\text{free}} + \sum_{i=2}^{k} \epsilon_1^{\text{free}} + 2\eta$.
3. v becomes LN-node: the degrees of x_1 and x_2 drop:
$\Delta_2 \geq \epsilon_2^{\text{BN}} + \sum_{\ell=2}^{\max_{h \in \{1,2\}} d^-(x_h)} \sum_{j=1}^{2} \chi(x_j \in \text{free}_\ell) \cdot \Delta_\ell^{\text{free}}$.

The worst case branching number from above is 1.897, created by two cases with $k = 1$: 1. $x_2 \in \text{free}_{\geq 4}$, $d^-(v_1) \geq 4$, $d^+(v_1) = 2$, $d^-(x_1) \geq 4$ and 2. $x_2 \in \text{FL}$, $d^-(v_1) \geq 4$, $d^+(v_1) = 2$, $d^-(x_1) \geq 4$.
If case (B1) applies to v_1 the reduction in both branches is as least as great as in (B4.1)/(B4.2).
If **(R6)** applies after the first branch (somewhere in the graph) we get $\Delta_1 \geq \epsilon_2^{\text{BN}} + (\epsilon_2^{\text{free}} - \epsilon_1^{\text{BN}}) + \epsilon_1^{\text{free}} + \min\{\epsilon^{\text{FL}}, (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}})\}$ and $\Delta_2 \geq \epsilon_2^{\text{BN}}$. Here the amount of ϵ_1^{free} in Δ_1 originates from an **(R6)** application. The corresponding branching number is 1.644.

(B5) A more complicated branching arises: $\langle v \in \text{IN}, x_1 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}; v \in \text{LN} \rangle$.
1. v and x_1 become IN-nodes; x_2 is now a LN-node; the vertices in $N^+(x_1) \cap$ free become BN-nodes and those in $N^+(x_1) \cap \text{FL}$ become LN-nodes:
$\Delta_1 \geq \epsilon_2^{\text{BN}} + \epsilon_2^{\text{free}} + \epsilon^{\text{FL}} + \sum_{x \in N^+(x_1) \cap \text{free}} (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}}) + \sum_{x \in N^+(x_1) \cap \text{FL}} \epsilon^{\text{FL}}$
2. v becomes IN-node; x_1 and x_2 become LN-nodes; after the application of `makeleaves`(v, x_1, x_2), the vertices in $[(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}] \cap \text{BN}$ become LN-nodes and those in $[(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}] \cap \text{free}$ become FL-nodes: $\Delta_2 \geq \epsilon_2^{\text{BN}} + \epsilon_2^{\text{free}} + \epsilon^{\text{FL}} + \min\{\epsilon_1^{\text{free}} - \epsilon^{\text{FL}}, \epsilon_2^{\text{BN}}\}$.
3. v becomes LN: $\Delta_3 \geq \epsilon_2^{\text{BN}}$.
The amount of $\min\{\epsilon_1^{\text{free}} - \epsilon^{\text{FL}}, \epsilon_2^{\text{BN}}\}$ in the second branch is due to (✣) and to the application of `makeleaves`(v, x_1, x_2).

(B6) $\langle v \in \text{IN}, x_1 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}, x_2 \in \text{IN}; v \in \text{LN} \rangle$: The branching vector can be derived by considering items 1,2 and 4 of (B7) and the reductions Δ_1, Δ_2 and Δ_4 in μ obtained in each item.

(B7) $\langle v \in \text{IN}, x_1 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}, x_2 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}, x_2 \in \text{LN}; v \in \text{LN} \rangle$: Note that if $N_A^+(x_1)$ or $N_A^+(x_2)$ is an arc-cut set, then (B4.2) applies. Thus, all the branching cases must be applicable. Moreover, due to the previous branching case (B4.3) we have $|N^+(x_1) \setminus N^+(v)| = |N^+(x_1) \setminus \{x_2\}| \geq 2$ and $|N^+(x_2) \setminus N^+(v)| = |N^+(x_2) \setminus \{x_1\}| \geq 2$ (✱). Note that $N^-(x_1) \cap N^-(x_2) = \{v\}$ due to (B6).
For $i \in \{1, 2\}$, let $fl_i = |\{x \in N^+(x_i) \setminus N^+(v) \mid x \in \text{FL}\}|$, $fr_i^{\geq 3} = |\{u \in N^+(x_i) \setminus N^+(v) \mid u \in \text{free}_{\geq 3}\}|$ and $fr_i^2 = |\{u \in N^+(x_i) \setminus N^+(v) \mid u \in \text{free}_2\}|$.
For $i \in \{1, 2\}$ we have $(fl_i + fr_i^{\geq 3} + fr_i^2) \geq 2$ due to (✱).
1.v and x_1 become IN; x_2 becomes BN; the free out-neighbors of x_1 become BN; the FL out-neighbors of x_1 become LN:

$\Delta_1 \geq \epsilon_2^{\mathrm{BN}} + \chi(x_1 \in \mathrm{free}_{\geq 3}) +$
$\chi(x_1 \in \mathrm{free}_2) \cdot \epsilon_2^{\mathrm{free}} + \chi(x_2 \in \mathrm{free}_{\geq 3}) \cdot (\epsilon_3^{\mathrm{free}} - \epsilon_2^{\mathrm{BN}}) + \chi(x_2 \in \mathrm{free}_2) \cdot (\epsilon_2^{\mathrm{free}} -$
$\epsilon_2^{\mathrm{BN}}) + (fl_1 \cdot \epsilon^{\mathrm{FL}} + fr_1^{\geq 3} \cdot (\epsilon_3^{\mathrm{free}} - \epsilon_2^{\mathrm{BN}}) + fr_1^2 \cdot (\epsilon_2^{\mathrm{free}} - \epsilon_2^{\mathrm{BN}}))$
2. v becomes IN; x_1 becomes LN; x_2 becomes IN; the free out-neighbors of x_2 becomes BN; the FL out-neighbors of x_2 become LN;
$\Delta_2 \geq \epsilon_2^{\mathrm{BN}} + (\sum_{i=1}^{2}[\chi(x_i \in \mathrm{free}_{\geq 3}) \cdot \epsilon_3^{\mathrm{free}} + \chi(x_i \in \mathrm{free}_2) \cdot \epsilon_2^{\mathrm{free}}]) +$
$(fl_2 \cdot \epsilon^{\mathrm{FL}} + fr_2^{\geq 3} \cdot (\epsilon_3^{\mathrm{free}} - \epsilon_2^{\mathrm{BN}}) + fr_2^2 \cdot (\epsilon_2^{\mathrm{free}} - \epsilon_2^{\mathrm{BN}}))$
3. v becomes IN; x_1 and x_2 become LN; the free in-neighbors of x_1 become FL; the BN in-neighbors of x_1 become LN; the free in-neighbors of x_2 become FL; the BN in-neighbors of x_2 become LN:
$\Delta_3 \geq \epsilon_2^{\mathrm{BN}} + [\sum_{i=1}^{2}(\chi(x_i \in \mathrm{free}_{\geq 3}) \cdot \epsilon_3^{\mathrm{free}} + \chi(x_i \in \mathrm{free}_2) \cdot \epsilon_2^{\mathrm{free}})] + \max\{2, (d^-(x_1)$ $+ d^-(x_2) - 4)\} \cdot \min\{\epsilon_1^{\mathrm{free}} - \epsilon^{\mathrm{FL}}, \epsilon_2^{\mathrm{BN}}\}$. Note that the additional amount of $\max\{2, (d^-(x_1) + d^-(x_2) - 4)\} \cdot \{\epsilon_2^{\mathrm{free}} - \epsilon^{\mathrm{FL}}, \epsilon_2^{\mathrm{BN}}\}$ is justified by Lemma 3.2 and by the fact that $d^-(x_i) \geq 2$ and $N^-(x_1) \cap N^-(x_2) = \{v\}$ due to (B6). Thus, we have $|(N^-(x_1) \cup N^-(x_2)) \setminus \{x_1, x_2, v\}| \geq \max\{2, (d^-(x_1) + d^-(x_2) - 4)\}$.
4. v becomes LN; the degrees of x_1 and x_2 drop:
$\Delta_4 \geq \epsilon_2^{\mathrm{BN}} + \sum_{j=2}^{\max_{\ell \in \{1,2\}} \{d^-(x_\ell)\}} \sum_{i=1}^{2} (\chi(d^-(x_i) = j) \cdot \Delta_j^{\mathrm{free}})$

(B8) Observe that in the second branch we can apply **(R6)**. Due to the non-applicability of **(R5)** and the fact that (B7) is ranked higher in priority we have $|(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}| = 1$. Especially, (B6) cannot be applied, yielding $N^-(x_1) \cap N^-(x_2) = \{v\}$. Thus, have the situation in Fig. 1. So, w.l.o.g., there are arcs $(q, x_1), (x_1, x_2) \in A$ (and possibly also $(x_2, x_1) \in A$), where $\{q\} = (N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}$, because we can rely on $d^-(x_i) \geq 2$ $(i = 1, 2)$ by (✤).
Firstly, assume that $q \in \mathrm{free}$.
(a) v becomes IN; x_1 and x_2 becomes BN: $\Delta_1 \geq \epsilon_2^{\mathrm{BN}} + 2 \cdot (\epsilon_2^{\mathrm{free}} - \epsilon_2^{\mathrm{BN}})$.
(b) The arc (q, x_1) will be contracted by **(R6)** when we v becomes LN, as x_1 and x_2 only can be reached by using (q, x_1): $\Delta_2 \geq \epsilon_2^{\mathrm{BN}} + \epsilon_1^{\mathrm{free}}$.
The branching number here is ≤ 1.606.
Secondly, assume $q \in \mathrm{BN}$. Then $q \in \mathrm{BN}_2$ due to the branching priorities.
(a) v becomes IN; x_1 and x_2 become BN: $\Delta_1 \geq \epsilon_2^{\mathrm{BN}} + 2 \cdot (\epsilon_2^{\mathrm{free}} - \epsilon_2^{\mathrm{BN}})$.
(b) Then after setting $v \in \mathrm{LN}$, rule **(R5)** will make q internal and subsequently also x_1: $\Delta_2 \geq \epsilon_2^{\mathrm{BN}} + \epsilon_2^{\mathrm{free}} + \epsilon_2^{\mathrm{BN}}$. This amount is justified by the changing roles of the vertices in $N^+(q) \cup \{q\}$.
1.499 upper-bounds the corresponding branching number.

Theorem 1. Directed Maximum Leaf Spanning Tree *can be solved in* $\mathcal{O}^*(1.9044^n)$ *steps.*

Notice that the claimed run time can be read off from Table 2 by computing $c = \max c_i$ as the basis. The proven run time bound admits only a small gap to the bound of $\mathcal{O}^*(1.8962^n)$ for the undirected version where we could benefit from the presence of degree two vertices on a greater scale.

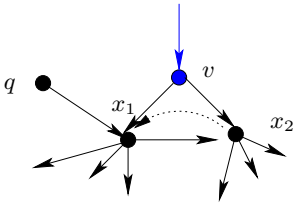


Fig. 1. The unique situation occurring in (B8). The arc pointing towards v is in T.

Table 2. Summarizing worst case branching numbers c_i per branching case (Bi)

(B2): $k = 1$, $d^-(v_1) \geq 4$, $d^+(v_1) = 2$, for all $u \in N^+(v_1)$ we have $u \in \text{free}_1 \rightsquigarrow c_2 < 1.8$.
(B3)a): $d^+(v) = 3$ and $|N^+(v) \cap \text{FL}| + |N^+(v) \cap \text{free}_{\geq 4}| = 3 \rightsquigarrow c_{3a)} \leq 1.9044$.
(B3)b): $c_{3b)} < 1.8$.
(B4.1/2): $c_{4.1/2} < 1.8$.
(B4.3): $c_{4.3} \leq 1.897$. (see main text)
(B5): $N^+(x_1) \subseteq$ free, $d^+(x_1) = 2$, $d^-(x_1) = 2$ and $N^+(x_1) \setminus \{v\} \subseteq \text{free}_1 \rightsquigarrow c_5 < 1.888$.
(B7)/(B6) : The next cases determine branching numbers $1.9043 \leq c_7 \leq 1.9044$.
1. $d^-(x_1) = d^+(x_1) = 2$, $d^-(x_2) = 4$, $d^+(x_2) = 2$, $fr_1^2 = fr_2^2 = 2$
2. $d^-(x_1) = d^-(x_2) = 3$, $d^+(x_1) = d^+(x_2) = 2$, $fr_1^2 = fr_2^2 = 2$
3. $d^-(x_2) = d^+(x_2) = 2$, $d^-(x_1) = 4$, $d^+(x_1) = 2$, $fr_1^2 = fr_2^2 = 2$
(B8): If $q \in$ free, then $c_8 < 1.7$.

3 Conclusions

An Approach Using Exponential Space. The algorithm of J. Kneis *et al.* [8] can also be read in an exact non-parameterized way. It is not hard to see that it yields a running time of $\mathcal{O}^*(2^n)$. Alternatively, keep the cases (B1) and (B2) of Algorithm 1 and substitute all following cases by a simple branch on some BN-node. Using n as a measure we see that $\mathcal{O}^*(2^n)$ is an upper bound. We are going to use the technique of memoization to obtain an improved running time. Let $SG^\alpha := \{G(V') \mid V' \subseteq V, |V'| \leq \alpha \cdot n\}$ where $\alpha = 0.141$. Then we aim to create the following table L indexed by some $G' \in SG^\alpha$ and some $V_{\text{BN}} \subseteq V(G')$: $L[G', V_{\text{BN}}] = T'$ such that $|leaves(T')| = \min_{\tilde{T} \subseteq \mathcal{L}} |leaves(\tilde{T})|$ where
$\mathcal{L} = \{\tilde{T} \mid \tilde{T}$ is directed spanning tree for $G'_{\text{BN}}$with root $r'\}$ and
$G'_{\text{BN}} = (V(G') \cup \{r', y\}, A(G') \cup (\{(r', y)\} \cup \{(r', u) \mid u \in V_{\text{BN}}\})$,
with r', y being new vertices. Entries where such a directed spanning tree $\tilde{T}$ does not exists (e.g., if $V_{\text{BN}} = \emptyset$) get the value $\emptyset$. This table can be filled up in time $\mathcal{O}^*\left(\binom{n}{\alpha \cdot n} \cdot 2^{\alpha n} \cdot 1.9044^{\alpha n}\right) \subseteq \mathcal{O}^*(1.8139^n)$. This run time is composed of enumerating SG^α, then by cycling through all possibilities for V_{BN} and finally solving the problem on G'_{BN} with Alg. 1. By stopping the $\mathcal{O}^*(2^n)$-algorithm[1] whenever $|V \setminus internal(T)| \leq \alpha \cdot n$ and looking up the rest of the solution in the table L, we can show:

[1] In the first phase, we cannot replace the $\mathcal{O}^*(2^n)$-algorithm by Algorithm 1. Namely, **(R6)** might generate graphs which are not vertex-induced subgraphs of G.

Theorem 2. Directed Maximum Leaf Spanning Tree *can be solved in time* $\mathcal{O}^*(1.8139^n)$ *consuming* $\mathcal{O}^*(1.6563^n)$ *space.*

Résumé. The paper at hand presented an algorithm which solves the Directed Maximum Leaf Spanning Tree problem in time $\mathcal{O}^*(1.9044^n)$. Although this algorithm follows the same line of attack as the one of [6] the algorithm itself differs notably. The approach of [6] does not simply carry over. To achieve our run time bound we had to develop new algorithmic ideas. This is reflected by the greater number of branching cases. Let us mention that we produced in [6] also a *lower-bound example*. By this we mean a class of graphs with n nodes such that the suggested algorithm would actually take $\Omega(c^n)$ time. The particular sample graph family from [6] can be easily adapted to the directed case, proving:

Theorem 3. *We can give a lower-bound example of* $\Omega(3^{n/3}) = \Omega(1.4422^n)$ *for the worst case running time of our algorithm.*

References

1. Blum, J., Ding, M., Thaeler, A., Cheng, X.: Connected dominating set in sensor networks and MANETs. In: Handbook of Combinatorial Optimization, vol. B, pp. 329–369. Springer, Heidelberg (2005)
2. Daligault, J., Gutin, G., Kim, E.J., Yeo, A.: FPT algorithms and kernels for the directed k-leaf problem. Journal of Computer and System Sciences 76(2), 144–152 (2010)
3. Daligault, J., Thomassé, S.: On finding directed trees with many leaves. In: Chen, J., Fomin, F.V. (eds.) Parameterized and Exact Computation, 4th International Workshop, IWPEC. LNCS, vol. 5917, pp. 86–97. Springer, Heidelberg (2009)
4. Drescher, M., Vetta, A.: An approximation algorithm for the Maximum Leaf Spanning Arborescence problem. ACM Transactions on Algorithms (in Press, 2008)
5. Fernau, H., Fomin, F.V., Lokshtanov, D., Raible, D., Saurabh, S., Villanger, Y.: Kernel(s) for problems with no kernel: On out-trees with many leaves. In: STACS. Dagstuhl Seminar Proceedings, vol. 09001, pp. 421–432 (2009)
6. Fernau, H., Kneis, J., Kratsch, D., Langer, A., Liedloff, M., Raible, D., Rossmanith, P.: An exact algorithm for the Maximum Leaf Spanning Tree problem. In: Chen, J., Fomin, F.V. (eds.) Parameterized and Exact Computation, 4th International Workshop, IWPEC. LNCS, vol. 5917, pp. 161–172. Springer, Heidelberg (2009)
7. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. Journal of the ACM 56(5) (2009)
8. Kneis, J., Langer, A., Rossmanith, P.: A new algorithm for finding trees with many leaves. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 270–281. Springer, Heidelberg (2008)
9. Koutis, I., Williams, R.: Limits and applications of group algebras for parameterized problems. In: ICALP (1). LNCS, vol. 5555, pp. 653–664. Springer, Heidelberg (2009)
10. Raible, D., Fernau, H.: An amortized search tree analysis for k-Leaf Spanning Tree. In: van Leeuwen, J. (ed.) SOFSEM 2010. LNCS, vol. 5901, pp. 672–684. Springer, Heidelberg (2009)
11. Raman, V., Saurabh, S.: Parameterized algorithms for feedback set problems and their duals in tournaments. Theoretical Computer Science 351, 446–458 (2006)
12. Thai, M.T., Wang, F., Liu, D., Zhu, S., Du, D.-Z.: Connected dominating sets in wireless networks different transmission ranges. IEEE Trans. Mobile Computing 6, 1–10 (2007)

Complexity of Propositional Proofs
(Invited Talk)

Alexander Razborov[1,2]

[1] University of Chicago, Chicago IL 60637, USA
[2] Steklov Mathematical Institute, Moscow, 117966, Russia

The underlying question of propositional proof complexity is amazingly simple: when interesting propositional tautologies possess efficient (which usually means short) proofs in a given propositional proof system? This theory is extremely well connected to very different disciplines like computational complexity, theoretical cryptography, automated theorem proving, mathematical logic, algebra and geometry. And, given its mixed origins, methods and concepts employed in the area are also very diverse.

In this lecture we will try to accomplish as much of the following as possible; our choice of topics is highly biased and personal.

We will begin with a brief discussion of connections to the classical proof theory, notably bounded arithmetic. Essentially, for virtually all propositional proof systems of significance there exists a first-order logical theory such that *provability* in this theory can be almost equated with *efficient provability* in the original proof system. In a sense, both capture the philosophical concept of provability in the world in which all objects that can be used by the prover are of restricted computational complexity (and in particular are restricted in size).

We give a sample of concrete lower and upper bounds for weak proof systems. We definitely will mention the Feasible Interpolation Theorem, as well as more elaborate applications of the "locality" method.

It turns out that (apparently) all lower bounds for restricted models and explicit Boolean functions known in circuit complexity at the moment possess efficient propositional proofs [12, Appendix]. It is an extremely interesting open question whether the same systems can prove strong lower bounds for unrestricted circuits of formulas, i.e., if they can resolve major open problems in complexity theory like $NP \stackrel{?}{\subseteq} P/poly$ or $P \stackrel{?}{\subseteq} NC^1/poly$. While the prospect of giving an *unconditional* answer to these exciting questions does look slim at the moment, we will discuss the possibility of showing this under reasonable complexity assumptions [14, Introduction].

Somewhat surprisingly, it turned out [1] that certain techniques used in proof-complexity studies can give unexpected and strong applications in a very different area, (theoretical) machine learning. We will try to mention this application as well.

Another important by-product of propositional proof complexity consists in applications to studying so-called *integrality gaps* for a variety of linear or positive semi-definite relaxation procedures widely used by the algorithmic community.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 340–342, 2010.

From the side of propositional proof complexity, these procedures correspond to ordinary proof systems with a very distinct algebraic or geometric flavor (see [8] for a comprehensive catalogue), and we hope to spend a considerable time on these results.

All references in the list below not mentioned in the main text are to various books, surveys and other writings on proof complexity that serve various tastes and can be used for further reading on the subject.

References

1. Alekhnovich, M., Braverman, M., Feldman, V., Klivans, A.R., Pitassi, T.: Learnability and automatizability. In: Proceedings of the 45th IEEE Symposium on Foundations of Computer Science, pp. 621–630 (2004)
2. Beame, P.: Proof complexity. In: Computational Complexity Theory. IAS/Park City mathematics series, vol. 10, pp. 199–246. American Mathematical Society, Providence (2004)
3. Beame, P., Pitassi, T.: Propositional proof complexity: Past, present and future. In: Paun, G., Rozenberg, G., Salomaa, A. (eds.) Current Trends in Theoretical Computer Science: Entering the 21st Century, pp. 42–70. World Scientific Publishing, Singapore (2001)
4. Buss, S.R.: Bounded arithmetic and propositional proof complexity. In: Logic of Computation, pp. 67–122 (1997)
5. Buss, S.R.: Towards NP-P via proof complexity and search (2009), http://math.ucsd.edu/~sbuss/ResearchWeb/lfcsSurvey
6. Cook, S.A.: Feasibly constructive proofs and the propositional calculus. In: Proceedings of the 7th Annual ACM Symposium on the Theory of Computing, pp. 83–97 (1975)
7. Cook, S.A., Reckhow, A.R.: The relative efficiency of propositional proof systems. Journal of Symbolic Logic 44(1), 36–50 (1979)
8. Yu Grigoriev, D., Hirsch, E.A., Pasechnik, D.V.: Complexity of semi-algebraic proofs. In: Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science, pp. 419–430 (2002)
9. Krajíček, J.: Bounded arithmetic, propositional logic and complexity theory. Cambridge University Press, Cambridge (1995)
10. Pudlák, P.: The lengths of proofs. In: Buss, S. (ed.) Handbook of Proof Theory, pp. 547–637. Elsevier, Amsterdam (1998)
11. Pudlák, P.: Twelve problems in proof complexity. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) CSR 2008. LNCS, vol. 5010, pp. 13–27. Springer, Heidelberg (2008)
12. Razborov, A.: Bounded Arithmetic and lower bounds in Boolean complexity. In: Clote, P., Remmel, J. (eds.) Feasible Mathematics II. Progress in Computer Science and Applied Logic, vol. 13, pp. 344–386. Birkhaüser, Basel (1995)
13. Razborov, A.: Lower bounds for propositional proofs and independence results in Bounded Arithmetic. In: Meyer auf der Heide, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 48–62. Springer, Heidelberg (1996)
14. Razborov, A.: Pseudorandom generators hard for k-DNF resolution and polynomial calculus resolution. Manuscript (2002), http://www.genesis.mi.ras.ru/~razborov

15. Razborov, A.: Propostional proof complexity. Lecture notes of a course taught at the University of Chicago (2009), http://people.cs.uchicago.edu/~razborov/teaching/winter09/notes.pdf
16. Segerlind, N.: The complexity of propositional proofs. Bulletin of Symbolic Logic 13(4), 417–481 (2007)
17. Urquhart, A.: The complexity of propositional proofs. Bulletin of Symbolic Logic 1, 425–467 (1995)

Quantization of Random Walks: Search Algorithms and Hitting Time (Invited Talk)

Miklos Santha[1,2]

[1] CNRS–LRI, Université Paris–Sud, 91405 Orsay, France
[2] Centre for Quantum Technologies, Nat. Univ. of Singapore, Singapore 117543

Extended Abstract

Many classical search problems can be cast in the following abstract framework: Given a finite set X and a subset $M \subseteq X$ of marked elements, *detect* if M is empty or not, or *find* an element in M if there is any. When M is not empty, a naive approach to the finding problem is to repeatedly pick a uniformly random element of X until a marked element is sampled. A more sophisticated approach might use a Markov chain, that is a random walk on the state space X in order to generate the samples. In that case the resources spent for previous steps are often reused to generate the next sample. Random walks also model spatial search in physical regions where the possible moves are expressed by the edges of some specific graph. The *hitting time* of a Markov chain is the number of steps necessary to reach a marked element, starting from the stationary distribution of the chain.

In this survey talk we discuss quantum walks, the discrete time quantization of classical Markov chains, together with some of their properties and applications to search problems. We proceed by analogy: Szegedy's method of quantization of Markov chains is presented as the natural analogue of a random walk on the edges of a graph. Grover search and the quantum walk based search algorithms of Ambainis, Szegedy, Magniez/Nayak/Roland/Santha and Magniez/Nayak/Richter/Santha are stated as quantum analogues of classical search procedures. The complexities of these search algorithms are analyzed in function of various parameters, including the quantum analogue of the eigenvalue gap of the classical chain, the phase gap, and the quantum hitting time.

Among the many applications of quantum walks we describe two in the query model of computation. In the Element Distinctness problem we are given a function f defined on $\{1, \ldots, n\}$, and we are looking for a pair of distinct elements $1 \leq i, j \leq n$ such that $f(i) = f(j)$. While the classical complexity of this problem is $\Theta(n)$, it can be solved by a quantum walk based algorithm due to Ambainis in time $O(n^{2/3})$, and this upper bound is thight. In the Triangle problem the input is the adjacency matrix of a graph G on vertex set $\{1, \ldots, n\}$, and the output is a triangle if there is any. This problem can easily be solved by Grover's algorithm in time $O(n^{3/2})$, while the obvious quantum lower bound is $\Omega(n)$. We present the quantum walk based algorithm of Magniez, Santha and Szegedy whose time is $O(n^{13/10})$, the exact complexity of the problem is open.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, p. 343, 2010.

Comparing Two Stochastic Local Search Algorithms for Constraint Satisfaction Problems

(Invited Talk)

Uwe Schöning

Institute of Theoretical Computer Science
University of Ulm
D 89069 Ulm, Germany
uwe.schoening@uni-ulm.de

Abstract. In this survey we compare the similarities, differences and the complexities of two very different approaches to solve a general constraint satisfaction probblems (CSP). One is the algorithm used in Moser's ingenious proof of a constructive version of Lovász Local Lemma [3], the other is the k-SAT random walk algorithm from [5,6], generalized to CSP's. There are several similarities, both algorithms use a version of stochastic local search (SLS), but the kind of local search neighborhood is defined differently, also the preconditions for the algorithms to work (efficiently) are quite different.

Keywords: constraint satisfaction problem, CSP, Lovász Local Lemma, SAT, satisfiability algorithm, stochastic local search, SLS.

1 Introduction

Recently there has been a lot of research in reducing the constant $c = c_k$ in the complexity bound $O(c^n)$ of algorithms which solve k-SAT problems, or more general, CSP problems (here also $c = c_{d,k}$ depends on certain parameters, especially the domain size d and the constraint size k.) One type of such algorithms is based on stochastic local search, that is, after choosing a random initial assignment, the algorithm performs a random walk on the space of potential solutions (this is $\{0,1\}^n$ in the SAT case, and more general, D^n for some finite set D in the CSP case.) Usually such random walk is "focussed" in the sense that some clause/constraint is selected (possibly using some particular selection strategy) that is *not* satisfied under the actual assignment, and then the assignment is changed by flipping one bit or some bits that correspond to variable values occuring in the clause or constraint.

It is a striking observation that this general description just given applies to two algorithms originating from quite different contexts. The first algorithm, due to Moser [3], is used to produce a constructive proof of Lovász Local Lemma, the other is an algorithm originally designed for k-SAT, especially 3-SAT (cf. [5]), which reduces the constant c to the smallest possible value.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 344–349, 2010.

2 Moser's Constructive Proof of Lovász Local Lemma

The Lovász Local Lemma states in its original form (cf. [1]) that in a probability space, for a set of (considered bad) events, whenever there is only some weak dependency between the events, there is a positive probability that none of the events will occur. Moser [3] (improved in [4], see also [2]) has proved a constructive version of the Local Lemma, in the sense that a (probabilistic) algorithm is able to find an elementary event which is disjoint with all such bad event (with high probability). Moser presents his proof in terms of the Boolean satisfiability problem. (Here an elementary event is a possible assignment to the variables of the given formula, and the "bad" events correspond to those assignments which cause clauses in the formula to be set to false. "Weak dependency" means that for each clause in the formula there is just a certain bounded number of clauses sharing at least one variable.)

We state the result in more general terms of a *constraint satisfaction problem*, CSP, for short. A CSP is specified by some finite *domain* D. (For simplicity we assume $D = \{1, 2, \ldots, d\}$.) Further, we have a set of *variables* $x_1, \ldots, x_n$ taking their values from D, and most importantly, there is a finite set of *constraints* $C_1, \ldots, C_m$ where each constraint is an inequality which rules out one specific value assignment to some (small) subset of the variables, for example, "$(x_3, x_4, x_8) \neq (1, 5, 2)$". The algorithmic task is to find a value assignment $(a_1, \ldots, a_n) \in D^n$ for the variables such that no constraint is injured. We assume here that each constraint involves at most k variables.

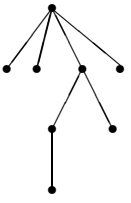
Under this setting the Local Lemma (or, a version of it) states: If for each constraint there are less than $d^k/4$ many constraints which share at least one variable with the respective constraint, then the CSP is satisfiable. Moreover, such a satisfying assignment can be found efficiently (with high probability) using some (probabilistic) algorithm.

Here is the probabilitic algorithm: Initially, give each variable a random value chosen uniformly from domain D. Now consider the constraints C_j, $j = 1, \ldots, m$, successively. If C_j is injured under the actual assignment, then call a recursive procedure Repair(C_j) as described below. Provided that all the recursive procedure calls return to the main program we will have the garantee that the assignment has been changed in a way that the constraint C_j is now satisfied and no other constraint that was satisfied before will be unsatisfied. Now, it should be clear, provided that all these calls of the procedure Repair will terminate, the entire algorithm will terminate and produce a satisfying assignment.

Here is a description of the procedure Repair(C) where C is a constraint being injured by the actual assignment: First give the k variables in constraint C random new values chosen uniformly from D^k. Then, inspect successively all the "neighbors" C' of constraint C (i.e. those constraints which have at least one variable with C in common) whether C' is injured by the new assignment. If so, call Repair(C') recursively (in a depth-first manner).

Suppose we interrupt the computation after t calls of the procedure Repair (as a thought experiment). Up to this point, we have used n values from D for the initial assignment, and for the t calls of the procedure Repair we have used each time k values from D. Altogether this amounts to $n \cdot \log d + t \cdot k \cdot \log d$ bits of information. Since these bits are random, it should not be possible to compress this information considerably (only with very low probability.)

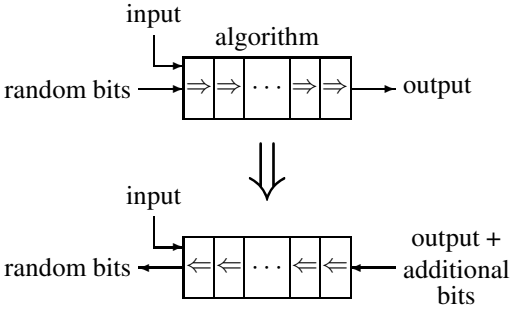
But actually, it is possible to reconstruct these random bits in a compact way, first, by using the output which we consider to be the assignment that has been reached after t Repair steps. Therefore, the output takes $n \log d$ bits. Furthermore, we describe the subset of $\{C_1, C_2, \ldots, C_m\}$ from which a call of Repair is started in the main program by using m bits. The structure of these recursive calls (a "forest" of trees with t edges altogether) can be described by a bitstring of length $2t$. (A "1" meaning a procedure call, a "0" meaning a return from a procedure call.) For example, the following tree



will be described by the bitstring 10101110010010.

Finally, the respective constraints in the trees of procedure calls can be identified by the index in the neighbor list of the predecessor constraint. This part of the information takes, by assumption, at most $(1-\varepsilon) \cdot t \cdot \log(d^k/4) \leq (1-\varepsilon) \cdot k \cdot t \cdot \log d - 2t$ bits, for some $\varepsilon > 0$.

Now the entire computation process can be recovered in a backwards fashion starting from the output assignment. Since we know at each step which constraint was used in the last Repair-call, we can reset the assignment to the values it had before the call. This is because, for each constraint, there is exactly one of the d^k many variable assignments which injures it. (Loosely speaking, the "go to" involves more entropy than the "come from".)



Therefore the entire computation can be recovered (as indicated in the picture) and also all random values which were used in the computation, including the initial assignment. Since "almost all" random values are incompressible, we get the inequality

theoretically best possible compression length $\leq$ achievable compression length

or

$$n \cdot \log d + t \cdot k \cdot \log d \leq n \cdot \log d + m + 2t + (1-\varepsilon) \cdot k \cdot t \cdot \log d - 2t$$

This gives us $\varepsilon \cdot t \cdot k \cdot \log d \leq m$, hence we obtain the following linear upper bound (and also a proof of termination): $t \leq \frac{1}{\varepsilon \cdot k \cdot \log d} \cdot m$.

3 Random Walk Algorithm for CSP

In [5,6] we descibe a k-SAT algorithm based on random walk (stochastic local search) and restarts which can also be applied to the more general situation of a CSP problem. What a CSP is was described in the last section. Given a CSP with domain D, domain size $d = |D|$, number of variables n, number of constraints m, number of variables per constraint k, the (basis) algorithm works as follows. Initially, give each variable a random value chosen uniformly from domain D. Now consider the constraints C_j, $j = 1, \ldots, m$, successively. If C_j is injured under the actual assignment, then call a recursive procedure Repair(C_j) as described below. If the number of (recursive) calls of Repair reaches some value cn, for some constant c, stop the basis algorithm (unsuccessfully). Provided that all the procedure calls return (i.e. terminate) in a regular way we will have the garantee that the assignment has been changed in a way that the constraint C_j is now satisfied and no other constraint that was satisfied before will be unsatisfied. Now, it should be clear that provided that these calls of the procedure Repair will all terminate, the entire algorithm will terminate and produce a satisfying assignment.

The procedure Repair(C), C being a constraint that is injured under the actual assignment, works as follows. Choose one of the variables occuring in C at random (d many choices), and choose for this variable another value (one of $d-1$ choices). Change the actual assignment accordingly. (Now although this is not really necessary for this algorithm to work properly, we proceed in analogy to Moser's procedure above.) Inspect successively all the "neighbors" C' of constraint C (i.e. those constraints which have at least one variable with C in common) whether C' is injured by the new assignment. If so, call Repair(C') recursively (in a depth-first manner).

The analysis below shows that, under the precondition that the given CSP has a solution, the success probability (the probability of finding a solution) during a single run of the basis algorithm (with at most cn many Repair's) is at least $(d \cdot (1 - 1/k))^{-n}$. Therefore, the expected number of repetitions of the basis algorithm until we find a solution is $(d \cdot (1 - 1/k))^n$. Put differently, supposing that the CSP has a solution, the probability that no solution is found after $t \cdot (d \cdot (1 - 1/k))^n$ many unsuccessful runs of the basis algorithm, is at most e^{-t}.

The algorithm can be analyzed using a Markov chain approach. Suppose we have the states of the Markov chain $0, 1, 2, \ldots$ where the interpretation of state j is that the Hamming distance of the actual assignment to some fixed satisfying assignment is j. Each change of the assignment during a call of the procedure repair can change the value of j to $j - 1$, to $j + 1$, or leaves j unchanged. In the worst case these 3 cases happen with the probabilities $\frac{1}{k \cdot (d-1)}$, $\frac{k-1}{k}$, or $\frac{d-2}{k \cdot (d-1)}$, respectively. Letting $p(j)$ be the probability that starting from state j this process finally reaches state 0, we get the equation:

$$p(j) = \frac{1}{k \cdot (d-1)} \cdot p(j-1) + \frac{d-2}{k \cdot (d-1)} \cdot p(j) + \frac{k-1}{k} \cdot p(j+1), \quad p(0) = 1$$

Using the ansatz $p(j) = \alpha^j$ we obtain the characteristic equation

$$0 = \frac{1}{d-1} + (\frac{d-2}{d-1} - k) \cdot \alpha + (k-1) \cdot \alpha^2$$

which has the solution $\alpha = \frac{1}{(d-1)\cdot(k-1)}$. Putting this together with the probability $\binom{n}{j}(\frac{d-1}{d})^j(\frac{1}{d})^{n-j}$ of reaching state j after the random choice of an initial assignment, we obtain the success probability of the basis algorithm, using the binomial theorem, as

$$\sum_{j=0}^{n} \binom{n}{j} \Big(\frac{d-1}{d}\Big)^j \Big(\frac{1}{d}\Big)^{n-j} \Big(\frac{1}{(d-1)\cdot(k-1)}\Big)^j = \Big(d\cdot(1-\frac{1}{k})\Big)^{-n}$$

In the case of 3-SAT this gives the probability $(4/3)^{-n}$. Notice that, provided the stochastic process reaches state 0 at all, it does so within cn steps, for some c, with high probability. Therefore, the above estimations also hold asymptotically if we cut off the stochastic local search process after cn steps and restart the basis algorithm, using a new random initial assignment (see also [7] for a discussion of restarts).

4 Conclusion

The two algorithms show striking similarities. Both belong to the class of stochastic local search algorithms, and both start with a randomly chosen initial assignment. Both algorithms focus in each random walk step on a non-satisfied constraint and change the assignment of the variables of the selected constraint. But here the differences start. In the case of the Moser algorithm it is important that the order in which the non-satisfied clauses are considered follows the depth-first scheme as described above. But the random walk algorithm can, without harm, be adapted to follow the same scheme, although its complexity analysis does not rely on it.

The real big difference, finally, is the local neighborhood to which the assignment is changed in a single step. In the case of the random walk algorithm, it is just a single value flip, which affects the Hamming distance to a fixed satisfying solution, by at most 1. In the case of Moser's algorithm the local neighborhood to which an assignment can be changed consists of a Hamming ball of radius k around the actual assignment. In both cases it is not of any advantage to switch the behavior. The random walk algorithm will not benefit from the Hamming ball neighborhood. (For example, in the case of 3-SAT the bound will go from $1.\overline{3}^n$ to 1.6^n.) Also, the termination proof in the case of Moser's algorithm will break down.

Nevertheless, we believe, even that Moser's algorithm is constructed for the very special purpose of giving a constructive proof of Lovász Local lemma, and needs for functioning a very strong assumption about the neighborhood size of constraints, we believe the underlying idea might flow in algorithmic solutions for constraint oder SAT algorithms.

References

1. Alon, N., Spencer, J.H.: The Probabilistic Method, 2nd edn. Wiley, Chichester (2000)
2. Fortnow's, L.: complexity blog, `http://blog.computationalcomplexity.org/2009/06/kolmogorov-complexity-proof-of-lov.html`
3. Moser, R.A.: A constructive proof of Lovász local lemma. In: Proceedings 41st Ann. ACM Symposium on Theory of Computing, pp. 343–350 (2009)

4. Moser, R.A., Tárdos, G.: A constructive proof of the general Lovász Local Lemma, http://arxiv.org/abs/0903.0544
5. Schöning, U.: A probabilistic algorithm for k-SAT and constraint satisfaction problems. In: Proceedings 40th Annual Symposium on Foundations of Computer Science, pp. 410–414 (1999)
6. Schöning, U.: A probabilistic algorithm for k-SAT based on limited local search and restart. Algorithmica 32(4), 615–623 (2002)
7. Schöning, U.: Principles of stochastic local search. In: Akl, S.G., Calude, C.S., Dinneen, M.J., Rozenberg, G., Wareham, H.T. (eds.) UC 2007. LNCS, vol. 4618, pp. 178–187. Springer, Heidelberg (2007)

Growth of Power-Free Languages over Large Alphabets

Arseny M. Shur

Ural State University

Abstract. We study growth properties of power-free languages over finite alphabets. We consider the function $\alpha(k, \beta)$ whose values are the exponential growth rates of β-power-free languages over k-letter alphabets and clarify its asymptotic behaviour. Namely, we suggest the laws of the asymptotic behaviour of this function when k tends to infinity and prove some of them as theorems. In particular, we obtain asymptotic formulas for $\alpha(k, \beta)$ for the case $\beta \geq 2$.

1 Introduction

The study of words and languages avoiding repetitions is one of the central topics in combinatorics of words since the pioneering work of Thue [19]. For a survey, see [2] and the references therein. The most impressive achievement of the recent years is the proof of Dejean's conjecture, stated in 1972 [8]. Let us briefly recall the story.

The repetitions considered by Dejean are periodic words, which are treated as *fractional powers* of words. The exponent of such a power is the ratio between its length and its minimal period. The β-power-free (or simply β-free) language over a given alphabet consists of all words which contain no fractional powers of the exponent $\gamma \geq \beta$ as factors. Exponent β is *k-avoidable* if the β-free language over the k-letter alphabet is infinite. Thue proved that the exponent 2 is 3-avoidable but not 2-avoidable, while all exponents that are greater than 2 are 2-avoidable. Dejean added the description of 3-avoidable exponents and suggested a simple rule to decide for any given numbers k and β whether a β-free language is k-avoidable. This rule became known as Dejean's conjecture. During more than three decades there were several attempts to attack this conjecture but only a modest success: the cases $k = 4, \ldots, 14$ were proved. The key contribution was made in 2006 by Carpi [4] who proved the conjecture for all large ($k \geq 33$) alphabets by a uniform construction. The remaining finite gap was filled in 2009 by Currie and Rampersad [6] and independently by Rao [12]. We should also note that computer search played a very big role in the solution to the cases $k = 5, \ldots, 32$, while Carpi's construction required no computer.

When a repetition is known to be avoidable, the following problem arises: how can we calculate some quantitative measure of its avoidability? The most natural measure is the order of growth of the avoiding language. The study of growth properties of formal languages has a long history since the paper by

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 350–361, 2010.

Morse and Hedlund [11]. A survey on the growth of power-free languages can be found in [1]. Recently the author proposed a universal method to estimate the growth of power-free languages and gave sharp bounds for the (exponential) growth rates of many particular languages [14,15,16,17].

In this paper we show that our method can be applied not only to individual languages but also to sets of languages. Somewhat similar to [4], we show how the growth of languages avoiding similar repetitions over different alphabets can be estimated in a uniform way. More precisely, we consider growth rate as a function of two numerical parameters: the size k of the alphabet and the avoided exponent β. Our goal is to describe the asymptotic behaviour of this function when k tends to infinity while β either is a constant or depends on k. On the basis of extensive numerical studies we suggest a general model of behaviour of growth rate (Section 3), consisting of five laws. Law 1 concerns very small exponents and immediately follows from Dejean's conjecture. Law 2 deals with "big" exponents ($\beta > 2$) and is proved as Theorem 1 in Section 4. Laws 3–5 are devoted to all other avoidable exponents. Two of them describe the asymptotic behaviour of the growth rate when β is constant (Law 3) or decreases with increasing k (Law 5). Law 4 compares the impacts on growth rate made by an increment of β and by an increment of k. In Section 5, Laws 3–5 are supported by some partial results and restated in a more precise form (see Conjectures 2,3). Laws 3,4 for the case $\beta = 2$ are proved in Theorem 2 in Section 4.

2 Preliminaries

We recall necessary notation and definitions. For more background, see [7,9,10].

1. Words and languages. An *alphabet* Σ is a nonempty finite set, the elements of which are called *letters*. *Words* are finite sequences of letters. As usual, we write Σ^* for the set of all words over Σ, including the *empty word* λ. A word u is a *factor* (respectively *prefix*, *suffix*) of a word w if w can be represented as $\bar{v}u\hat{v}$ (respectively $u\hat{v}$, $\bar{v}u$) for some (possibly empty) words $\bar{v}$ and $\hat{v}$. A *factor* (respectively *prefix*, *suffix*) of w is called *proper* if it does not coincide with w. Subsets of Σ^* are called *languages* (over Σ). A language is *factorial* if it is closed under taking factors of its words.

A word w is *forbidden* for a language L if it is a factor of no word from L. The set of all minimal (w.r.t. taking factors of words) forbidden words for a language is called the *antidictionary* of this language. A factorial language is uniquely determined by its antidictionary. If the antidictionary is regular (in particular, finite), then the language is also regular.

A word $w \in \Sigma^*$ can be viewed as a function $\{1, \ldots, n\} \to \Sigma$. Then a *period* of w is any period of this function; the minimal period is denoted by $\mathrm{per}(w)$. The *exponent* of w is given by $\exp(w) = |w|/\mathrm{per}(w)$, where $|w|$ is the length of w. If $\exp(w) > 1$ then w is a fractional power. The word w is said to be β-free (β^+-free) if all its factors have exponents less than β (respectively, at most β). By β-free (β^+-free) languages we mean the languages of *all* β-free (respectively β^+-free) words over a given alphabet. These languages are obviously factorial and

are also called *power-free* languages. Following [3], we use only the term β-free, assuming that β belongs to the set of "extended rationals". This set consists of all rational numbers and all such numbers with a plus; the number x^+ covers x in the usual $\leq$ order such that the inequalities $y \leq x$ and $y < x^+$ are equivalent.

By $L(k,\beta)$ and $M(k,\beta)$ we denote the β-free language over a k-letter alphabet and its antidictionary, respectively. We set $M_m(k,\beta)=\{w\in M(k,\beta) \mid \mathrm{per}(w)\leq m\}$. The (regular) language $L_m(k,\beta)$ with the antidictionary $M_m(k,\beta)$ is called the *m-approximation* of $L(k,\beta)$. The word $w \in M(k,\beta)$ is called an *s-repetition*, if $s = |w| - \mathrm{per}(w)$. By $L^{(r)}(k,\beta)$ we mean the language whose antidictionary consists of all s-repetitions of M satisfying $s \leq r$. Of course, $L^{(r)}(k,\beta)$ coincides with some m-approximation of $L(k,\beta)$.

The *repetition threshold* is the function $RT(k) = \inf\{\beta \mid L(k,\beta) \text{ is infinite}\}$. *Dejean's conjecture* states that $RT(3) = 7/4$, $RT(4) = 7/5$, $RT(k) = k/(k-1)$ for $k = 2$ and $k \geq 5$, and $L(k, RT(k))$ is always finite.

2. Automata, digraphs, and growth rates. We consider *deterministic finite automata* (dfa's) with *partial* transition function. We view a dfa as a digraph, sometimes even omitting the labels of edges. A *trie* is a dfa that is a tree such that the initial vertex is its root and the set of terminal vertices is the set of all its leaves. Only *directed* walks in digraphs are considered. For a dfa, the number of words of length n in the language it recognizes obviously equals the number of *accepting walks* of length n in the automaton. A dfa is *consistent* if each vertex is contained in some accepting walk.

The growth rate of a language L is defined by $\alpha(L) = \limsup_{n\to\infty}(C_L(n))^{1/n}$, where $C_L(n)$ is the number of words of length n in L. For factorial languages, lim sup can be replaced by lim. We write $\alpha(k,\beta)$ instead of $\alpha(L(k,\beta))$.

A *strongly connected component* (scc) of a digraph G is a subgraph G' maximal w.r.t. inclusion such that there is a walk from any vertex of G' to any other vertex of G'. A digraph is *strongly connected*, if it consists of a unique scc.

The *index* of a digraph G is the maximum absolute value of the eigenvalues of its adjacency matrix A_G, and is denoted by $\mathrm{Ind}(G)$. By the Perron-Frobenius Theorem, $\mathrm{Ind}(G)$ is itself an eigenvalue of A_G, called the *Frobenius root*, and has a nonnegative eigenvector called the *principal eigenvector* of A_G. The index of G equals maximum of the indices of its scc's.

Growth rates of regular languages and indices of digraphs are closely connected. Namely, if a language L is recognized by a consistent dfa $\mathcal{A}$, then $\alpha(L) = \mathrm{Ind}(\mathcal{A})$. A short proof of this fact can be found in [13].

3 Model of Behaviour of the Growth Rate Function

We start with a table presenting some numerical results on the growth rates. These results are obtained by the author using the method of [14,15,16,17]. In the last four columns the exact values of growth rates, rounded off to 6 digits after the comma, are presented. The values in other columns are obtained by extrapolation of a sequence of upper bounds only. Basing on these numerical results, we suggest five laws describing the asymptotic behaviour of the function $\alpha(k,\beta)$.

Table 1. Growth rates of some power-free languages

k	$\frac{k}{k-1}^+$	$\frac{k-1}{k-2}$	$\frac{k-1}{k-2}^+$	$\frac{k-2}{k-3}$	$\frac{k-2}{k-3}^+$	$3/2$	$(3/2)^+$	2	2^+	3	3^+
3						0	0	1,301762	2,605879	2,701562	2,911924
4						1,0968	2,28052	2,621508	3,728494	3,778951	3,948787
5	1,1577	1,1646	2,2485			2,40242	3,492800	3,732539	4,789851	4,822067	4,966241
6	1,2246	1,2289	2,2768	2,3764	3,400	3,540514	4,603386	4,791407	5,827733	5,850362	5,976010
7	1,2369	1,2374	2,2988	2,3265	3,373	4,627498	5,672703	5,828466	6,853725	6,870588	6,982056
8	1,2348	1,2349	2,3125	2,3282	3,362	5,686769	6,720687	6,854117	7,872761	7,885852	7,986065
9	1,24666	1,2467	2,318	2,325	3,361	6,729676	7,756057	7,872990	8,887342	8,897819	8,988863
10	1,23930	1,23933	2,321	2,324	3,364	7,762175	8,783291	8,887486	9,898887	9,907470	9,990893
11	1,24260	1,24263	2,323	2,325	3,366	8,787655	9,804948	9,898981	10,908264	10,915429	10,992414
12	1,24287	1,24288	2,324	2,325	3,369	9,808175	10,822603	10,908328	11,916035	11,922111	11,993583
13	1,24087	1,24087	2,325	2,326	3,371	10,825060	11,837286	11,916080	12,922584	12,927802	12,994501
14	1,24277	1,24277	2,325	2,326	3,373	11,839200	12,849695	12,922617	13,928179	13,932711	13,995235
15	1,24183	1,24183	2,326	2,326	3,374	12,851217	13,860327	13,928203	14,933016	14,936989	14,995831

Law 1. *If $\beta \le RT(k)$ then $\alpha(k,\beta) = 0$.* This law follows immediately from Dejean's conjecture.

Law 2. *If $\beta \ge 2^+$ then $\alpha(k,\beta)$ tends to k as k approaches infinity such that the difference $k-\alpha(k,\beta)$ monotonically decreases at a polynomial rate.* This law can be observed in three last columns of Table 1. Note that $\alpha(k,2^+), \alpha(k,3) \approx k-1/k$, while $\alpha(k,3^+) \approx k-1/k^2$. Theorem 1 in Section 4 implies this law and gives the precise degree of the polynomial rate.

Now suppose that $RT(k) < \beta \le 2$.

Law 3. *If $\beta \in [\frac{n+1}{n}^+, \frac{n}{n-1}]$ for an integer $n \ge 2$, then $\alpha(k,\beta)$ tends to $k+1-n$ as k approaches infinity such that the difference $(k+1-n)-\alpha(k,\beta)$ monotonically decreases at linear rate starting with some number $k = k(n)$.* This law is illustrated by the columns of the exponents $3/2, (3/2)^+$, and 2 in Table 1.

Law 4. *If $\beta = \frac{n}{n-1}$ for an integer $n \ge 2$, then adding $^+$ to β has nearly the same effect as adding 1 to k: the difference $\alpha(k+1, \frac{n}{n-1})-\alpha(k, \frac{n}{n-1}^+)$ tends to zero at nonlinear polynomial rate as k approaches infinity.* One can observe this law comparing the values in the columns of the exponents $3/2$ and $(3/2)^+$, and also of the exponents 2 and 2^+ in Table 1. For $\beta = 2$, Laws 3 and 4 follow from Theorem 2, see Section 4. In the general case these laws follow from Conjecture 2 (Section 5) which suggests the asymptotics of the function $\alpha(k,\beta)$ for arbitrary fixed β from the interval $(RT(k), 2]$.

Law 5. *If β depends on k such that $\beta \in [\frac{k-n}{k-n-1}^+, \frac{k-n-1}{k-n-2}]$ for a fixed nonnegative integer n, then $\alpha(k,\beta)$ tends to some constant α_n as k approaches infinity and, moreover, $\alpha_n+1 < \alpha_{n+1}$.* This is our *Conjecture 1.* It is illustrated by the first five columns of Table 1. In Section 5 we give some partial results supporting this conjecture and suggest the values of α_0, α_1, and α_2.

4 Asymptotic Formulas for the Case $\beta \ge 2$

Theorem 1. *Let $n \ge 2$ be an integer, $\beta \in [n^+, n+1]$ be an exponent. Then the following equality holds:*

$$\alpha(k,\beta) = \begin{cases} k - \frac{1}{k^{n-1}} + \frac{1}{k^n} - \frac{1}{k^{2n-2}} + O\left(\frac{1}{k^{2n-1}}\right) & \text{if } \beta \in [n^+, n{+}\frac{1}{2}]; \\ k - \frac{1}{k^{n-1}} + \frac{1}{k^n} + O\left(\frac{1}{k^{2n-1}}\right) & \text{if } \beta \in [(n{+}\frac{1}{2})^+, n{+}1]. \end{cases} \quad (1)$$

Corollary 1. *For any $\beta \geq 2^+$ one has $(k - \alpha(k,\beta)) \to 0$ as $k \to \infty$, and the rate of convergence is polynomial in k.*

Corollary 2. $\alpha(k,n^+) - \alpha(k,n) = \frac{1}{k^{n-2}} + O\left(\frac{1}{k^{n-1}}\right)$ *for any* $n \geq 3$, $\alpha(k,n{+}1) - \alpha(k,n^+) = \frac{1}{k^{2n-2}} + O\left(\frac{1}{k^{2n-1}}\right)$ *for any* $n \geq 2$.

Thus, for a fixed k the jumps of the function $\alpha(k,\beta)$ at the endpoints of the interval $[n^+, n{+}1]$ are much bigger than the variation of this function inside this interval. But with the growth of k all these jumps become negligible, while the jump at the point $n = 2$ is always big:

Theorem 2. *The following equalities hold:*

$$\begin{aligned} \alpha(k{+}1,2) &= k - \tfrac{1}{k} - \tfrac{1}{k^3} + O\left(\tfrac{1}{k^5}\right); \\ \alpha(k,2^+) &= k - \tfrac{1}{k} - \tfrac{1}{k^3} - \tfrac{1}{k^4} + O\left(\tfrac{1}{k^5}\right). \end{aligned} \quad (2)$$

Corollary 3 (Big Jump Property). $\alpha(k,2^+) - \alpha(k,2) = 1 + \frac{1}{k^2} + O\left(\frac{1}{k^3}\right)$.

Corollary 4 (Law 4 for $n = 2$). $\alpha(k{+}1,2) - \alpha(k,2^+) = \frac{1}{k^4} + O\left(\frac{1}{k^5}\right)$.

Remark 1. All obtained formulas correlate very well with the numerical results of Table 1.

Now turn to the proof of Theorem 1. It consists of two big steps. First we use the method previously developed in [14,15,16] to get upper bounds for the values of $\alpha(k,\beta)$ and then the method of [17] to convert the upper bounds into two-sided ones. Normally one should fix k and β before applying these methods, but here we consider these numbers as parameters.

Proof (upper bounds). Since $L(k,\beta) \subset L_m(k,\beta)$, the growth rate $\alpha(L_m(k,\beta))$ approximates $\alpha(k,\beta)$ from above. To get $\alpha(L_m(k,\beta))$ one can list all words of $M_m(k,\beta)$, build a consistent dfa $\mathcal{A}_m$ for $L_m(k,\beta)$ using a variant of the textbook Aho-Corasick algorithm (see [5]), and calculate the Frobenius root $\mathrm{Ind}(\mathcal{A}_m)$. However, it is more efficient to use symmetry to build a much smaller digraph with the index $\mathrm{Ind}(\mathcal{A}_m)$. We call this digraph the *factor automaton* of the initial automaton $\mathcal{A}_m$. This digraph is built as a dfa, but the language it recognizes does not make much sense, because we need only the index which is a purely graph characteristic. Algorithm FA, which builds factor automata, is described in [16] and the details of its efficient implementation are in [15]. Here we do not need the details, only the algorithm.

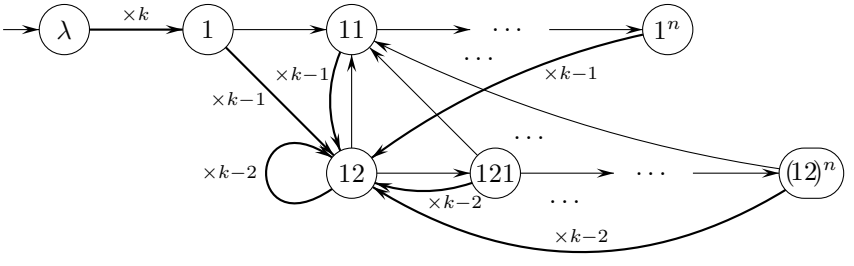
Suppose that Σ is an ordered alphabet. For two words $v, w \in \Sigma^*$ we write $v \sim w$ if $v = \sigma(w)$ for some permutation σ of the alphabet. A word $v \in \Sigma^*$ is called *lexmin*, if $v \sim w$ implies that v is lexicographically smaller than w.

Algorithm FA
Input: the set of all lexmin words of $M_m(k,\beta)$.
Output: factor automaton $\mathcal{FA}_m$ with the index $\alpha(L_m(k,\beta))$.
0. Build the trie $\mathcal{T}$ recognizing the set of input words. Associate each vertex in

$\mathcal{T}$ with the word labeling the accepting walk ending at this vertex.
1. Add all possible edges to $\mathcal{T}$, following the rule:
the edge (u, v) with the label x should be added if (a) u is not terminal, and (b) u has no outgoing edge labeled by x, and (c) $v \sim w$, where w is the longest suffix of the word ux which is $\sim$-equivalent to a vertex of $\mathcal{T}$.
2. Remove all terminal vertices and mark all remaining vertices as terminal to get $\mathcal{FA}_m$.

Note that Algorithm FA diverges from Aho-Corasick's algorithm in two points only: we take lexmin forbidden words instead of all forbidden words as an input and require $\sim$-equivalence of words instead of their equality in step 1.

Now we use Algorithm FA to build the factor automaton for the 2-approximation $L_2(k, \beta)$. Let $\Sigma = \{1, \dots, k\}$. Two different but very similar cases are possible. If $\beta \in [n^+, n{+}\frac{1}{2}]$, the input of Algorithm FA consists of the words 1^{n+1} and $(12)^n1$. If $\beta \in [(n{+}\frac{1}{2})^+, n{+}1]$, the second word is replaced by $(12)^{n+1}$. Below the factor automaton for the first case is given. For the second case we should add one more vertex to the longer branch; this new vertex $(12)^n1$ has the same outgoing edges as the other vertices of this branch. (When drawing the factor automaton we omit the labels of the edges, because they do not matter. Instead, we indicate the multiplicities of multiple edges which are drawn in boldface.)



We proceed with the first case. Since the index of a graph equals the maximum of indices of its scc'c, we discard the vertices λ and 1 from the obtained factor automaton to get a strongly connected digraph with the adjacency matrix

$$A = \left[\begin{array}{cccc|cccc} k{-}2 & 1 & 0 & \dots\ 0 & 1 & 0 & \dots & 0 \\ k{-}2 & 0 & 1 & \dots\ 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ k{-}2 & 0 & 0 & \dots\ 1 & 1 & 0 & \dots & 0 \\ k{-}2 & 0 & 0 & \dots\ 0 & 1 & 0 & \dots & 0 \\ \hline k{-}1 & 0 & 0 & \dots\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ k{-}1 & 0 & 0 & \dots\ 0 & 0 & 0 & \dots & 1 \\ k{-}1 & 0 & 0 & \dots\ 0 & 0 & 0 & \dots & 0 \end{array}\right]_{3n-2 \times 3n-2} \tag{3}$$

The upper and lower square blocks are of size $2n{-}1$ and $n{-}1$ respectively. *Irreducible* nonnegative matrices, which correspond to strongly connected digraphs, possess the following useful property (see [9]): the Frobenius root lies strictly between the minimum and the maximum of row sums of the matrix. So, for the

matrix A the Frobenius root equals $k-\varepsilon$, where $0<\varepsilon<1$. Let us calculate the characteristic polynomial $|A-rE|$, replacing r by $k-\varepsilon$. Adding all columns of the matrix $A-rE$ to the first one, we get

$$|A-rE| = \left|\begin{array}{ccccc|cccc} \varepsilon & 1 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ \varepsilon & \varepsilon{-}k & 1 & \dots & 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \varepsilon & 0 & 0 & \dots & 1 & 1 & 0 & \dots & 0 \\ \varepsilon{-}1 & 0 & 0 & \dots & \varepsilon{-}k & 1 & 0 & \dots & 0 \\ \hline \varepsilon & 0 & 0 & \dots & 0 & \varepsilon{-}k & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \varepsilon & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 1 \\ \varepsilon{-}1 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & \varepsilon{-}k \end{array}\right|$$

This determinant can be explicitly calculated due to the following claim. Here C_{ij} denotes the cofactor of the (i,j)-th entry of the matrix $A-rE$ and

$$\Delta = \left|\begin{array}{ccccc} 1 & 0 & \dots & 0 & 1 \\ \varepsilon{-}k & 1 & \dots & 0 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & 1 \\ 0 & 0 & \dots & \varepsilon{-}k & 1 \end{array}\right|_{2n-1\times 2n-1} = \sum_{i=0}^{2n-2}(-1)^i(\varepsilon{-}k)^i \qquad (4)$$

(to calculate this determinant one can expand along the last row to get a simple recurrence).

Claim. $C_{i1} = \begin{cases} (-1)^{i-1}(\varepsilon{-}k)^{3n-2-i} & \text{if } i<2n; \\ (-1)^{i-1}(\varepsilon{-}k)^{3n-2-i}\cdot\Delta & \text{otherwise.} \end{cases}$

Let us prove this claim. In the case $i<2n$ the matrix of C_{i1} equals $\begin{bmatrix} B & X \\ O & C \end{bmatrix}$, where B is a lower unitriangular matrix of size $(i{-}1)$ while C is an upper triangular matrix of size $(3n{-}2{-}i)$ with the numbers $(\varepsilon{-}k)$ along the diagonal. Multiplying the diagonal entries we get the required formula. In the case $i\ge 2n$ the matrix of C_{i1} equals $\begin{bmatrix} D & O & O \\ X & B & O \\ O & O & C \end{bmatrix}$, where B is a lower unitriangular matrix of size $(i{-}2n)$, C is an upper triangular matrix of size $(3n{-}2{-}i)$ with the numbers $(\varepsilon{-}k)$ along the diagonal, and the determinant Δ of D is given by (4). Thus, we multiply Δ by the diagonal entries of C to finish the claim. Note that in the cases $i=1$, $i=2n$, and $i=3n{-}2$ some matrices in the above representations degenerate, but the formulas obviously remain valid.

Now expand the determinant $|A-rE|$ along the first column:

$$|A-rE| =$$
$$\varepsilon\cdot\sum_{i=1}^{2n-1}(-1)^{i-1}(\varepsilon{-}k)^{3n-2-i} + \Delta\cdot\varepsilon\cdot\sum_{i=2n}^{3n-2-i}(-1)^{i-1}(\varepsilon{-}k)^{3n-2-i} - (\varepsilon{-}k)^{n-1} + (-1)^n\cdot\Delta =$$

$$\varepsilon\cdot\big((\varepsilon{-}k)^{3n-3}{-}2(\varepsilon{-}k)^{3n-4}{+}\cdots{+}(-1)^{n-1}n(\varepsilon{-}k)^{2n-2}{+}\cdots{+}$$
$$(-1)^{2n-2}n(\varepsilon{-}k)^{n-1}+(-1)^{2n-1}(n{-}1)(\varepsilon{-}k)^{n-2}+\cdots+(-1)^{3n-3}\big)+$$
$$(-1)^n\big(1-(\varepsilon{-}k)+(\varepsilon{-}k)^2-\cdots+(\varepsilon{-}k)^{2n-2}\big)-(\varepsilon{-}k)^{n-1}=0 \quad (5)$$

The sums in the first and second parentheses of (5) equal $(-k)^{3n-3}+O(k^{3n-4})$ and $(-1)^n(k^{2n-2}+O(k^{2n-3}))$ respectively. To obtain zero, we must put $\varepsilon = \frac{1}{k^{n-1}}+O(\frac{1}{k^n})$. Then we can simplify (5) so that after cancelling the powers of (-1) the equation looks like

$$\varepsilon\cdot\big(k^{3n-3}{+}2k^{3n-4}{+}\cdots{+}nk^{2n-2}{+}O(k^{2n-3})\big)$$
$$-\,k^{2n-2}{-}k^{2n-3}{-}\cdots{-}k^n{-}2k^{n-1}{+}O(k^{n-2})=0 \quad (6)$$

Only two highest terms in the parentheses in (6) contribute to the coefficient of k^{2n-3}. The second term contributes 2, so the first term must contribute (-1) to annihilate the term $-k^{2n-3}$ outside the parentheses. Hence, $\varepsilon = \frac{1}{k^{n-1}}-\frac{1}{k^n}+O(\frac{1}{k^{n+1}})$. Analyzing the coefficients of $k^{2n-4},\ldots,k^{n-1}$ in the same way, we finally obtain

$$\varepsilon = \frac{1}{k^{n-1}}-\frac{1}{k^n}+\frac{1}{k^{2n-2}}+O\Big(\frac{1}{k^{2n-1}}\Big)$$

(the third term in the expansion of ε is due to the coefficient -2 of k^{n-1}). Since the index of the analyzed 2-approximation equals $k{-}\varepsilon$, we get the upper formula of (1).

Now consider the 2-approximation in the case $\beta\in[(n{+}\frac{1}{2})^+,n{+}1]$. The adjacency matrix has the same structure (3), only the sizes of the upper square block and the whole matrix increase by 1. One can perform the same analysis as above to arrive at the following analogue of (6):

$$\varepsilon\cdot\big(k^{3n-2}{+}2k^{3n-3}{+}\cdots{+}nk^{2n-1}{+}O(k^{2n-2})\big)$$
$$-\,k^{2n-1}{-}k^{2n-2}{-}\cdots{-}k^n{+}O(k^{n-1})=0. \quad (7)$$

Note that the degrees of all monomials in (7) increase by 1 w.r.t. the corresponding monomials in (6). The only exception is the monomial $-k^{n-1}$ which corresponds to the last term of (5). As a result, this monomial is absorbed by the O-expression. So, we have no coefficient -2 outside the parentheses and then no term $\frac{1}{k^{2n-2}}$ in the expansion of ε. Hence, the index of the 2-approximation is $k-\frac{1}{k^{n-1}}+\frac{1}{k^n}+O(\frac{1}{k^{2n-1}})$, and we are done with the upper bound.

Proof (lower bound). We recall the method of [17] to convert the upper bounds obtained from m-approximations to the two-sided bounds. The quality of the lower bound in this method increases multiplicatively with increasing m. Here we improve this method so that it is enough to consider only 2-approximations in the case $\beta\ge 3^+$ and 3-approximations in the case $\beta\le 3$ to prove (1). If $\beta\le 3$, there are four different 3-approximations depending on whether β falls into the interval $[2^+,\frac{7}{3}]$, $[\frac{7}{3}^+,\frac{5}{2}]$, $[\frac{5}{2}^+,\frac{8}{3}]$, or $[\frac{8}{3}^+,3]$. So we must check that the growth rates of these 3-approximation satisfy (1). Since the parameter n is no longer involved, the calculation of determinants becomes routine, so we omit it.

To improve the method of [17], we sketch its main ideas. Let $\mathcal{A}_m$ be the dfa built by Aho-Corasick's algorithm to recognize the m-approximation $L_m(k,\beta)$ (instead, all the reasoning can be done using the factor automaton $\mathcal{FA}_m$, but it would be harder to follow). The dfa $\mathcal{A}_m$ accepts all words of $L(k,\beta)$ and also the forbidden words in which the minimal forbidden factors have minimal periods greater than m. We define the vectors $\boldsymbol{P}(t)$ and $\boldsymbol{Q}(t)$, the components of which correspond to the vertices of $\mathcal{A}_m$. The component $[\boldsymbol{P}(t)]_u$ equals the number of words of length t in the language $L(k,\beta)$ which are accepted by the state (vertex) u. The component $[\boldsymbol{Q}(t)]_u$ equals the number of words w of length t which are accepted by the state u and have the following property: $w \notin L(k,\beta)$ but $v \in L(k,\beta)$ for any proper prefix v of w. Hence we have

$$\boldsymbol{P}(t{+}1) = \boldsymbol{P}(t)A - \boldsymbol{Q}(t{+}1),$$

where A is the adjacency matrix of $\mathcal{A}_m$. Further, if $\mathcal{A}_m$ has a unique nonsingleton scc (for the m-approximations we study in this theorem, this is certainly the case), and $\boldsymbol{x}$ is the principal right eigenvector of A then the growth rate of the scalar product $S(t) = \boldsymbol{P}(t)\boldsymbol{x}$ equals $\alpha(k,\beta)$. Hence,

$$S(t{+}1) = \boldsymbol{P}(t{+}1)\boldsymbol{x} = \boldsymbol{P}(t)A\boldsymbol{x} - \boldsymbol{Q}(t{+}1)\boldsymbol{x} = \mathrm{Ind}(\mathcal{A}_m)S(t) - \boldsymbol{Q}(t{+}1)\boldsymbol{x}. \quad (8)$$

Now estimate $[\boldsymbol{Q}(t{+}1)]_u$. Each word w which is counted in $[\boldsymbol{Q}(t{+}1)]_u$ ends by a suffix v which has (a) the shortest period $j > m$ (b) the length $\lceil \beta j \rceil$ (c) no forbidden powers inside (d) the word u as a suffix. One can prove that $|u|{<}(\beta{-}1)j$. Hence, u occurs in v not only as a suffix but also j symbols to the left from this suffix. Then the prefix w' of length $|w|{-}j$ of the word w is accepted by the state u as well. Since w' determines w and $w' \in L(k,\beta)$ by the choice of w, we get the inequality

$$[\boldsymbol{Q}(t{+}1))]_u \le \sum_{j=m+1}^{\lfloor \frac{t+1}{\beta} \rfloor} [\boldsymbol{P}(t{+}1{-}j)]_u.$$

For arbitrary number γ satisfying the inequality $\gamma^{j-1}S(t{+}1{-}j) \le S(t)$ for any $j = m{+}1, \dots, \lfloor \frac{t+1}{\beta} \rfloor$ we have

$$\boldsymbol{Q}(t{+}1)\boldsymbol{x} \le \sum_{j=m+1}^{\lfloor \frac{t+1}{\beta} \rfloor} S(t{+}1{-}j) \le S(t) \cdot \sum_{j=m+1}^{\lfloor \frac{t+1}{\beta} \rfloor} \frac{1}{\gamma^{j-1}} < \frac{S(t)}{\gamma^{m-1}(\gamma-1)}. \quad (9)$$

Substituting (9) in (8) one can show that

$$\gamma{+}\frac{1}{\gamma^{m-1}(\gamma-1)} \le \mathrm{Ind}(\mathcal{A}_m) \quad \text{implies} \quad \gamma < \alpha(k,\beta). \quad (10)$$

Producing a better estimation of $[\boldsymbol{Q}(t{+}1))]_u$ we can get a better lower bound. In the case $m = 2$ the word u is a prefix of a word a^{n+1} or $(ab)^{n+1}$ for some $a, b \in \Sigma$. In the first case we have $|u| < j$, because otherwise v has period 1, contradicting to the property (a). Similarly, $|u| \le j$ in the second case, because v

is not 2-periodic. Since $\beta > n$ by the conditions of the theorem, we have $|v| > nj$. Then v contains an occurrence of u $(n-1)j$ symbols away from the right end. Hence, the prefix w' of length $|w|-(n-1)j$ of the word w is accepted by the state u, w' determines w and $w' \in L(k,\beta)$ by the choice of w. So, we can recalculate (9) (recall that $m = 2$):

$$\boldsymbol{Q}(t+1)\boldsymbol{x} \le \sum_{j=3}^{\lfloor \frac{t+1}{\beta} \rfloor} S(t+1-(n-1)j) \le S(t) \cdot \sum_{j=3}^{\lfloor \frac{t+1}{\beta} \rfloor} \frac{1}{\gamma^{(n-1)j-1}} < \frac{S(t)}{\gamma^{2(n-1)-1}(\gamma^{n-1}-1)}.$$

Similar to the above analysis,

$$\gamma + \frac{1}{\gamma^{2(n-1)-1}(\gamma^{n-1}-1)} \le \mathrm{Ind}(\mathcal{A}_2) \quad \text{implies} \quad \gamma < \alpha(k,\beta). \tag{11}$$

Let $\bar{\gamma}$ be the best lower bound obtained from (11). Then the equality $\mathrm{Ind}(\mathcal{A}_2) = k - \frac{1}{k^{n-1}} + \frac{1}{k^n} - \frac{1}{k^{2n-2}} + O\big(\frac{1}{k^{2n-1}}\big)$ implies $\mathrm{Ind}(\mathcal{A}_2) - \bar{\gamma} = O\big(\frac{1}{k^{3n-4}}\big)$. Therefore, for $n \ge 3$ this difference is absorbed by $O\big(\frac{1}{k^{2n-1}}\big)$. Hence, any number from the interval $[\bar{\gamma}, \mathrm{Ind}(\mathcal{A}_2)]$, including $\alpha(k,\beta)$, can be represented as $k - \frac{1}{k^{n-1}} + \frac{1}{k^n} - \frac{1}{k^{2n-2}} + O\big(\frac{1}{k^{2n-1}}\big)$.

In the case $n = 2$ the rule (11) does not improve (10). But for $m = 3$ even the rule (10) gives a lower bound within the required distance $O\big(\frac{1}{k^3}\big)$ from the upper bound. The proof is complete.

Proof (of Theorem 2). To prove (2), we use the same ideas as in the proof of Theorem 1. To get the lower bound within the distance $O\big(\frac{1}{k^5}\big)$ from the upper bound we calculate the growth rates of 5-approximations for $L(k,2)$ and $L(k,2^+)$. This routine is done by a computer program. The corresponding factor automaton for the 2-free language has 49 vertices; 42 of them belong to the nonsingleton scc. For the 2^+-free language these numbers are 244 and 226 respectively.

5 Remarks on Lesser Exponents

No algorithm producing sharp two-sided bounds of the growth rates is currently known in the case $\beta < 2$. So, here we give two conjectures about the behaviour of $\alpha(k,\beta)$ in the case $\beta < 2$ and some partial results to support these conjectures.

1. Laws 3 and 4

Conjecture 2. For a fixed integer $n \ge 3$ and arbitrarily large k the equalities $\alpha(k, \frac{n}{n-1}^+) = k+2-n-\frac{n-1}{k}+O\big(\frac{1}{k^2}\big)$ and $\alpha(k, \frac{n}{n-1}) = k+1-n-\frac{n-1}{k}+O\big(\frac{1}{k^2}\big)$ hold.

If Conjecture 2 holds true, it implies Law 3 and the equations

$$\begin{aligned}
&\alpha(k, \tfrac{n}{n-1}^+) - \alpha(k, \tfrac{n}{n-1}) = 1 + O(\tfrac{1}{k^2}) \ ; \\
&\alpha(k, \tfrac{n}{n-1}) - \alpha(k, \tfrac{n+1}{n}^+) = \tfrac{1}{k} + O(\tfrac{1}{k^2}) \ ; \\
&\alpha(k+1, \tfrac{n}{n-1}) - \alpha(k, \tfrac{n}{n-1}^+) = O(\tfrac{1}{k^2}) \text{ (refined Law 4).}
\end{aligned}$$

We can use the method of previous section to find upper bounds for $\alpha(k, \frac{n}{n-1}^+)$ and $\alpha(k, \frac{n}{n-1})$. First, take the language $L^{(1)}(k, \frac{n}{n-1})$ (respectively, $L^{(1)}(k, \frac{n}{n-1}^+)$) avoiding all 1-repetitions of the given power-free language. One can see that this language is given by a very simple condition: each successive $n-1$ (respectively, $n-2$) letters in any word are different. As an easy consequence we have

Proposition 1. $\alpha(k, \frac{n}{n-1}^+) \le k+2-n$, $\alpha(k, \frac{n}{n-1}) \le k+1-n$.

To get the required upper bounds for the growth rates it is crucial to consider 2-repetitions, which are avoided by m-approximations satisfying the conditions $m \in [n-1, 2n-3]$ for $L(k, \frac{n}{n-1}^+)$ and $m \in [n, 2n-2]$ for $L(k, \frac{n}{n-1})$. As in Section 4, for a fixed m these approximations have very similar factor automata. Namely, the following proposition holds.

Proposition 2. *Let $n \ge 3$, $s \le n-2$ be fixed nonnegative integers, $k > 2n-3$, $A(k) = \big(a_{ij}(k)\big)$ be the adjacency matrix of the factor automaton built for the language $L_{n+s}(k, \frac{n}{n-1})$. Then either $a_{ij}(k) \equiv b_{ij}$, or $a_{ij}(k) \equiv k - c_{ij}$ for some constants b_{ij}, c_{ij}. The same results with different constants are true for the language $L_{n+s-1}(k, \frac{n}{n-1}^+)$.*

Due to Proposition 2 we calculate the growth rates of corresponding approximations in the same way as in Section 4. With the aid of computer we have

Proposition 3. *For $0 \le s \le 7$, $\alpha\big(L_{n+s-1}(k, \frac{n}{n-1}^+)\big) \le k+2-n-\frac{s+1}{k}+O\big(\frac{1}{k^2}\big)$, $\alpha\big(L_{n+s}(k, \frac{n}{n-1})\big) \le k+1-n-\frac{s+1}{k}+O\big(\frac{1}{k^2}\big)$. In particular, for $n \le 9$ the values $\alpha(k, \frac{n}{n-1}^+)$ and $\alpha(k, \frac{n}{n-1})$ are bounded from above by the expressions given in Conjecture 2.*

Remark 2. Unfortunately, there is no similarity between the factor automata of the languages $L^{(2)}(k, \frac{n}{n-1})$ (respectively, $L^{(2)}(k, \frac{n}{n-1}^+)$) for different values of n. As a result, the adjacency matrices of corresponding factor automata do not have common structure and we cannot calculate the growth rate manipulating with determinants of variable size, as in Section 4.

2. Law 5 Can be split in two statements: (A) for a fixed n the sequences $\big\{\alpha(k, \frac{k-n}{k-n-1}^+)\big\}$ and $\big\{\alpha(k, \frac{k-n-1}{k-n-2})\big\}$ have limits, (B) these limits coincide.

To justify these statements, first consider the case $n = 0$. It was shown in [18] that $\alpha\big(L^{(2)}(k, \frac{k}{k-1}^+)\big) = \hat{\alpha}_2 \approx 1,3247$ independently of k, while the sequence $\big\{\alpha\big(L^{(m)}(k, \frac{k}{k-1}^+)\big)\big\}$ for any fixed $m > 2$ has the limit $\hat{\alpha}_m$ which is the growth rate of some regular 2-dimensional language. Moreover, $\hat{\alpha}_3 = \hat{\alpha}_4 = \hat{\alpha}_5$ and, from numerical experiments, $\hat{\alpha}_3 \approx 1{,}2421$, $\hat{\alpha}_3 - \hat{\alpha}_6 \approx 4 \cdot 10^{-5}$, $\hat{\alpha}_6 - \hat{\alpha}_7 \approx 4 \cdot 10^{-6}$. So, it is quite probable that the sequence $\{\alpha(k, \frac{k}{k-1}^+)\}$ has a limit. The following proposition supports (B):

Proposition 4. $L^{(m)}(k, \frac{k}{k-1}^+) = L^{(m)}(k, \frac{k-1}{k-2})$ *for any* $m \le k/2$.

Similar properties can be observed for $n = 1$ and $n = 2$. We mention some of them without proofs.

Proposition 5. *For* $k \geq 6$, $\alpha\big(L^{(2)}(k, \frac{k-1}{k-2}^{+})\big) = \bar{\alpha}_2 \approx 2{,}3790$. *The sequences* $\big\{\alpha\big(L^{(3)}(k, \frac{k-1}{k-2}^{+})\big)\big\}$ *and* $\big\{\alpha\big(L^{(2)}(k, \frac{k-2}{k-3}^{+})\big)\big\}$ *have the limits* $\bar{\alpha}_3 \approx 2,3301$ *and* $\tilde{\alpha}_2 \approx 3,4070$ *respectively.* $L^{(m)}(k, \frac{k-1}{k-2}^{+}) = L^{(m)}(k, \frac{k-2}{k-3})$ *for any* $m \leq (k{+}1)/4$.

Conjecture 3. $\alpha_0 \approx 1{,}242$, $\alpha_1 \approx 2{,}326$, $\alpha_2 \approx 3{,}376$.

Remark 3. The values α_1 and α_2 are conjectured on the base of Table 1. But if our methods slightly overestimate α_1 and slightly underestimate α_2, then it may be that $\alpha_1 = \hat{\alpha}_2 + 1$, $\alpha_2 = \bar{\alpha}_2 + 1$.

References

1. Berstel, J.: Growth of repetition-free words – a review. Theor. Comput. Sci. 340, 280–290 (2005)
2. Berstel, J., Karhumäki, J.: Combinatorics on words: A tutorial. Bull. Eur. Assoc. Theor. Comput. Sci. 79, 178–228 (2003)
3. Brandenburg, F.-J.: Uniformly growing k-th power free homomorphisms. Theor. Comput. Sci. 23, 69–82 (1983)
4. Carpi, A.: On Dejean's conjecture over large alphabets. Theor. Comp. Sci. 385, 137–151 (2007)
5. Crochemore, M., Mignosi, F., Restivo, A.: Automata and forbidden words. Inform. Processing Letters 67(3), 111–117 (1998)
6. Currie, J.D., Rampersad, N.: A proof of Dejean's conjecture, http://arxiv.org/PScache/arxiv/pdf/0905/0905.1129v3.pdf
7. Cvetković, D.M., Doob, M., Sachs, H.: Spectra of graphs. Theory and applications, 3rd edn. Johann Ambrosius Barth, Heidelberg (1995)
8. Dejean, F.: Sur un Theoreme de Thue. J. Comb. Theory, Ser. A 13(1), 90–99 (1972)
9. Gantmacher, F.R.: Application of the theory of matrices. Interscience, New York (1959)
10. Lothaire, M.: Combinatorics on words. Addison-Wesley, Reading (1983)
11. Morse, M., Hedlund, G.A.: Symbolic dynamics. Amer. J. Math. 60, 815–866 (1938)
12. Rao, M.: Last Cases of Dejean's Conjecture. In: Proceedings of the 7th International Conference on Words, Salerno, Italy, paper no. 115 (2009)
13. Shur, A.M.: Comparing complexity functions of a language and its extendable part. RAIRO Theor. Inf. Appl. 42, 647–655 (2008)
14. Shur, A.M.: Combinatorial complexity of regular languages. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) Computer Science – Theory and Applications. LNCS, vol. 5010, pp. 289–301. Springer, Heidelberg (2008)
15. Shur, A.M.: Growth rates of complexity of power-free languages. Submitted to Theor. Comp. Sci. (2008)
16. Shur, A.M.: Two-sided bounds for the growth rates of power-free languages. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 466–477. Springer, Heidelberg (2009)
17. Shur, A.M.: Growth rates of power-free languages. Russian Math. (Iz VUZ) 53(9), 73–78 (2009)
18. Shur, A.M., Gorbunova, I.A.: On the growth rates of complexity of threshold languages. RAIRO Theor. Inf. Appl. 44, 175–192 (2010)
19. Thue, A.: Über unendliche Zeichenreihen, Kra. Vidensk. Selsk. Skrifter. I. Mat.-Nat. Kl., Christiana 7, 1–22 (1906)

A Partially Synchronizing Coloring

Avraham N. Trahtman

Bar-Ilan University, Dep. of Math., 52900, Ramat Gan, Israel
trakht@macs.biu.ac.il
http://www.cs.biu.ac.il/∼trakht

Abstract. Given a finite directed graph, a coloring of its edges turns the graph into a finite-state automaton. A k-synchronizing word of a deterministic automaton is a word in the alphabet of colors at its edges that maps the state set of the automaton at least on k-element subset. A coloring of edges of a directed strongly connected finite graph of a uniform outdegree (constant outdegree of any vertex) is k-synchronizing if the coloring turns the graph into a deterministic finite automaton possessing a k-synchronizing word.

For $k = 1$ one has the well known road coloring problem. The recent positive solution of the road coloring problem implies an elegant generalization considered first by Béal and Perrin: a directed finite strongly connected graph of uniform outdegree is k-synchronizing iff the greatest common divisor of lengths of all its cycles is k.

Some consequences for coloring of an arbitrary finite digraph are presented. We describe a subquadratic algorithm of the road coloring for the k-synchronization implemented in the package TESTAS. A new linear visualization program demonstrates the obtained coloring. Some consequences for coloring of an arbitrary finite digraph and of such a graph of uniform outdegree are presented.

Keywords: graph, algorithm, synchronization, road coloring, deterministic finite automaton.

Introduction

The famous road coloring problem was stated almost 40 years ago [1] for a strongly connected directed graph of uniform outdegree where the greatest common divisor (gcd) of lengths of all its cycles is one.

Together with the Černy conjecture [7], [9], the road coloring problem was once one of the most fascinating problems in the theory of finite automata. In the popular Internet Encyclopedia "Wikipedia" it is on the list of most interesting unsolved problems in mathematics. The recent positive solution of the road coloring conjecture [10], [13], [11] has posed a lot of generalizations and new problems.

One of them is k-synchronizing coloring. A solution of the problem based on the method from [11] was appeared first in [4] and repeated later independently in [5], [12].

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 362–370, 2010.

Some consequences for coloring of an arbitrary finite digraph as well as for coloring of such a graph of a uniform outdegree are a matter of our interest. The minimal value of k for k-synchronizing coloring exists for any finite digraph and therefore a partially synchronizing coloring can be obtained.

Here we describe a polynomial time algorithm for k-synchronizing coloring. Our proof also is based on [11] and [12], the more so the proofs in [4] meanwhile have some gaps. The theorems and lemmas from [11] are presented below without proof. The proofs are given only for new or modified results. The realization of the algorithm is demonstrated by a new linear visualization program [3]. For an n-state digraph with uniform outdegree d, the time complexity of the algorithm is $O(n^3d)$ in the worst case and $O(n^2d)$ in the majority of cases. The space complexity is quadratic.

Preliminaries

As usual, we regard a directed graph with colors assigned to its edges as a finite automaton, whose input alphabet Σ consists of these colors.

A directed graph with constant outdegree (the number of outgoing edges) of all its vertices is called a graph of *uniform outdegree*.

A finite directed strongly connected graph of uniform outdegree where the gcd of lengths of all its cycles is k will be called *k-periodic*.

An automaton is *deterministic* if no state has two outgoing edges of the same color. In a *complete* automaton each state has outgoing edges of any color.

If there exists a path in an automaton from the state $\mathbf{p}$ to the state $\mathbf{q}$ and the edges of the path are consecutively labelled by $\sigma_1, ..., \sigma_k \in \Sigma$, then for $s = \sigma_1 ... \sigma_k \in \Sigma^+$ let us write $\mathbf{q} = \mathbf{p}s$.

Let Ps be the map of the subset P of states of an automaton using the word $s \in \Sigma^+$. For the transition graph Γ of an automaton let Γs denote the map of the set of states of the automaton.

Let $|P|$ denote the size of the subset P of states from an automaton (of vertices from a graph).

A word $s \in \Sigma^+$ is called a *k-synchronizing* word of the automaton with transition graph Γ if both $|\Gamma s| = k$ and for all words $t \in \Sigma^*$ holds $|\Gamma t| \geq k$.

A coloring of a directed finite graph is *k-synchronizing* if the coloring turns the graph into a deterministic finite automaton possessing a k-synchronizing word and the value of k is minimal.

A pair of distinct states $\mathbf{p}, \mathbf{q}$ of an automaton (of vertices of the transition graph) will be called *synchronizing* if $\mathbf{p}s = \mathbf{q}s$ for some $s \in \Sigma^+$. In the opposite case, if for any s $\mathbf{p}s \neq \mathbf{q}s$, we call the pair *deadlock*.

A synchronizing pair of states $\mathbf{p}$, $\mathbf{q}$ of an automaton is called *stable* if for any word u the pair of states $\mathbf{p}u, \mathbf{q}u$ is also synchronizing [6], [8].

We call the set of all outgoing edges of a vertex a *bunch* if all these edges are incoming edges of only one vertex.

The subset of states (of vertices of the transition graph Γ) of maximal size such that every pair of states from the set is a deadlock will be called an *F-clique*.

1 A k-Synchronizing Coloring

Lemma 1. *Let a finite directed strongly connected graph Γ with uniform outdegree have a k-synchronizing coloring. Then the greatest common divisor d of lengths of all its cycles is not greater than k.*

If the length of a path from the state $\mathbf{p}$ to the state $\mathbf{q}$ is equal to $i \neq 0$ (modulo d) then for any word s one has $\mathbf{p}s \neq \mathbf{q}s$.

Proof. Let N be the function defined on the states of Γ in the following way - we take an arbitrary vertex $\mathbf{p}$ and let $N(\mathbf{p}) = 0$. Then for each vertex $\mathbf{q}$ with defined $N(\mathbf{q})$ suppose for any next nearest vertex $\mathbf{r}$ $N(\mathbf{r}) = N(\mathbf{q}) + 1$ (modulo d). The graph is strongly connected, whence for each vertex the function N is defined. The enumeration does not imply a contradiction anywhere because the length of each cycle is divided by d. Then by any coloring the mapping by a word t produced the same shift of size $|t|$ (modulo d) of the function N on the states. Therefore the states with distinct values of N could not have common image and $|\Gamma t| \geq d$ for any word t. $|\Gamma s| = k$ for k-synchronizing word s. Consequently, $k \geq d$.

By any coloring, the mapping by a word s produced the same shift of the function N on the set of states. $N(\mathbf{p}s) = N(\mathbf{p}) + |s|$ (modulo d). Therefore the difference of the values of the function N on two states is not changed by the shift.

Theorem 1. *[6], [8], [10] Let us consider an arbitrary coloring of a strongly connected graph Γ with constant outdegree. Stability of states is a binary relation on the set of states of the obtained automaton. Denote the reflexive closure of this relation by ρ. Then ρ is a congruence relation, Γ/ρ presents a directed strongly connected graph with constant outdegree, the gcd d of all its cycles is the same as in Γ, and a k-synchronizing coloring of Γ/ρ implies a k-synchronizing recoloring of Γ.*

Lemma 2. *[10] Let F be an F-clique via some coloring of a strongly connected graph Γ. For any word s the set Fs is also an F-clique and each state [vertex] $\mathbf{p}$ belongs to some F-clique.*

Lemma 3. *Let A and B ($|A| > 1$) be distinct F-cliques via some coloring without stable pairs of the k-periodic graph Γ. Then $|A| - |A \cap B| = |B| - |A \cap B| > 1$.*

Proof. Let us assume the contrary: $|A| - |A \cap B| = 1$. By the definition of F-clique, $|A| = |B|$ and so $|B| - |A \cap B| = 1$, too. Thus $|A| - |A \cap B| = |B| - |A \cap B| = 1$.

The pair of states $\mathbf{p} \in A \setminus B$ and $\mathbf{q} \in B \setminus A$ is not stable. Therefore for some word s the pair $(\mathbf{p}s, \mathbf{q}s)$ is a deadlock. Any pair of states from the F-clique A and from the F-clique B as well as from F-cliques As and Bs is a deadlock. So each pair of states from the set $(A \cup B)s$ is a deadlock, whence $(A \cup B)s$ is an F-clique.

One has $|(A \cup B)s| = |A| + 1 > |A|$ in spite of maximality of the size of F-clique A.

Lemma 4. *[10] Let some vertex of a directed complete graph Γ have two incoming bunches. Then any coloring of Γ has a stable pair.*

Proof. If a vertex $\mathbf{p}$ has two incoming bunches from vertices $\mathbf{q}$ and $\mathbf{r}$, then the pair $\mathbf{q}$, $\mathbf{r}$ is stable for any coloring because $\mathbf{q}\alpha = \mathbf{r}\alpha = \mathbf{p}$ for any letter (color) $\alpha \in \Sigma$.

1.1 The Spanning Subgraph with Maximum of Edges in the Cycles

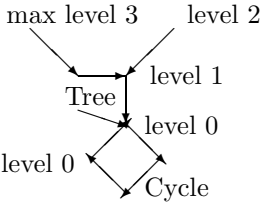
Definition 1. *Let us call a subgraph S of the k-periodic graph Γ a spanning subgraph of Γ if all vertices of Γ belong to S and exactly one outgoing edge of every vertex (in the usual graph-theoretic terms it is 1-outregular spanning subgraph).*

A maximal subtree of the spanning subgraph S with root on a cycle from S and having no common edges with cycles from S is called a tree *of S.*

The length of the path from a vertex $\mathbf{p}$ through the edges of its tree of the spanning set S to the root of the tree is called the level *of $\mathbf{p}$ in S.*

A tree with vertex of maximal level is called a maximal tree.

Remark 1. *Any spanning subgraph S consists of disjoint cycles and trees with roots on cycles. Each tree and cycle of S is defined identically. The level of the vertices belonging to some cycle is zero. The vertices of the trees except the roost have a positive level. The vertices of maximal positive level have no incoming edge in S. The edges labelled by a given color defined by any coloring form a spanning subgraph. Conversely, for each spanning subgraph, there exists a coloring and a color such that the set of edges labelled with this color corresponds to this spanning subgraph.*



Lemma 5. *Let N be a set of vertices of level n from some tree of the spanning subgraph S of the k-periodic graph Γ. Then via a coloring of Γ such that all edges of S have the same color α, for each F-clique F holds $|F \cap N| \leq 1$.*

Proof. Some power of α synchronizes all states of a given level of the tree and maps them into the root. Each pair of states from an F-clique could not be synchronized and therefore could not belong to N.

Lemma 6. *[10] Let d-periodic graph Γ have a spanning subgraph R consisting solely of disjoint cycles (without trees). Then Γ either is a cycle of length d of bunches or has another spanning subgraph with exactly one maximal tree.*

Lemma 7. *[11] Assume that no vertex of the graph Γ has two incoming bunches. Let R be a spanning subgraph with non-trivial tree and let its tree T with the root $\mathbf{r}$ on cycle H have all vertices of maximal level L and one of them is the vertex $\mathbf{p}$. Let us consider*

1) replacing an edge e from R with an edge having the same start vertex as e and ending in $\mathbf{p}$,

2) replacing in R an incoming edge of a root on the path in the tree from $\mathbf{p}$,

3) replacing in R an incoming edge of a root from H.

Suppose that at most two such operations do not increase the number of edges in cycles. Then by the coloring of R by the color α Γ has a spanning subgraph with one maximal tree and the pair $\mathbf{p}\alpha^{L-1}$, $\mathbf{p}\alpha^{L+|H|-1}$ is stable.

Theorem 2. *Any k-periodic graph Γ of size greater than k has a coloring with stable pair.*

Proof. We have $|\Gamma| > k$. By Lemma 4, in the case of vertex with two incoming bunches Γ has a coloring with stable pairs. In opposite case, by Lemmas 7 and 6, Γ has a spanning subgraph R with one maximal tree in R.

Let us give to the edges of R the color α and denote by C the set of all vertices from the cycles of R. Then let us color the remaining edges of Γ by other colors arbitrarily.

By Lemma 2, in a strongly connected graph Γ for every word s and F-clique F of size $|F| > 1$, the set Fs also is an F-clique of the same size and for any state $\mathbf{p}$ there exists an F-clique F such that $\mathbf{p} \in F$.

In particular, some F has non-empty intersection with the set N of vertices of maximal level L. The set N belongs to one tree, whence by Lemma 5 this intersection has only one vertex. The word α^{L-1} maps F on an F-clique F_1 of size $|F|$. One has $|F_1 \setminus C| = 1$ because the sequence of edges of the color α from any tree of R leads to the root of the tree, and the root belongs to a cycle colored by α from C and only for the set N with vertices of maximal level holds $N\alpha^{L-1} \not\subseteq C$. So $|N\alpha^{L-1} \cap F_1| = |F_1 \setminus C| = 1$ and $|C \cap F_1| = |F_1| - 1$.

Let the integer m be a common multiple of the lengths of all considered cycles from C colored by α. So for any $\mathbf{p}$ from C as well as from $F_1 \cap C$ holds $\mathbf{p}\alpha^m = \mathbf{p}$. Therefore for an F-clique $F_2 = F_1\alpha^m$ holds $F_2 \subseteq C$ and $C \cap F_1 = F_1 \cap F_2$.

Thus two F-cliques F_1 and F_2 of size $|F_1| > 1$ have $|F_1| - 1$ common vertices. So $|F_1 \setminus (F_1 \cap F_2)| = 1$. Consequently, in view of Lemma 3, there exists a stable pair in the considered coloring.

Theorem 3. *Every strongly connected graph Γ is k-periodic if and only if the graph has a k-synchronizing coloring. For arbitrary coloring Γ is m-synchronizing for $m \geq k$.*

Proof. From the k-synchronizing coloring of Γ by Lemma 1 follows that if Γ is m-periodic graph then $m \geq k$. So $|\Gamma| \geq k$. If $|\Gamma| = k$ then also $m = k$.

Thus we only need to consider the case $|\Gamma| > k$. By Lemmas 6 and 7 there exists a spanning subgraph with one maximal tree, whence by Theorem 2 there exists a stable pair of states. Theorem 1 reduced the problem to a homomorphic image of Γ of smaller size and the induction finishes the proof.

The theorem 3 implies some consequences, in particular, even for an arbitrary digraph.

Lemma 8. *Let a finite directed graph Γ have a sink component Γ_1. Suppose that by removing some edges of Γ_1 one obtains strongly connected directed graph Γ_2 of uniform outdegree. Let k be the greatest common divisor of lengths of the cycles of Γ_2. Then Γ has k-synchronizing coloring.*

Proof. Γ_2 has a k-synchronizing coloring by Theorem 3. The edges outside Γ_2 can be colored arbitrarily, sometimes by adding new colors for to obtain deterministic automaton.

Lemma 9. *Let Γ_1,..., Γ_m be strongly connected components of a finite directed graph Γ of uniform outdegree such that no edge leaves the component Γ_i $(0 < i \leq m)$. Let k_i be the greatest common divisor of lengths of cycles in Γ_i and suppose $k = \sum_{j=1}^{m} k_j$. Then Γ has k-synchronizing coloring.*

The Lemmas 8, 9 imply

Theorem 4. *Let finite directed graph Γ have a subgraph of uniform outdegree with strongly connected components Γ_1,..., Γ_m such that no edge leaves the component Γ_i $(0 < i \leq m)$ in the subgraph. Let k_i be the greatest common divisor of lengths of cycles in Γ_i and suppose $k = \sum_{j=1}^{m} k_j$. Then Γ has a k-synchronizing coloring.*

The proof for arbitrary k did not use anywhere that $k \neq 1$, whence from the validity of the Road Coloring Conjecture we have the following:

Theorem 5. *[11] [10] Every finite strongly connected graph Γ of uniform outdegree with greatest common divisor of all its cycles equal to one has synchronizing coloring.*

2 Find Minimal k for k-Synchronizing Coloring

The algorithm is based on Theorem 3 and Lemma 8. One must check the existence of strongly connected components (SCC) having no outgoing edges to others SCC and check the condition on gcd in any such SCC H. There exists a subgraph S of C of maximal uniform outdegree. Then we check the condition on gcd in any such S. Let us use the linear algorithm of finding strongly connected component SCC [2]. Then we mark all SCC having outgoing edges to others SCC and study only all remaining SCC H.

The coloring of H: We find in every H the value of k_H for the k_H-synchronizing coloring of H. Let $\mathbf{p}$ be a vertex from H. Suppose $d(\mathbf{p}) = 1$. For an edge $\mathbf{r} \to \mathbf{q}$ where $d(\mathbf{r})$ is already defined and $d(\mathbf{q})$ is not defined suppose $d(\mathbf{q}) = d(\mathbf{r}) + 1$. If $d(\mathbf{q})$ is defined let us add the difference $abs(d(\mathbf{q}) - 1 - d(\mathbf{r}))$ to the set D and count the gcd of the integers from D. If finally $gcd = k_H$, then the SCC H has k_H-synchronizing coloring. The value of k is equal to the sum of all such k_H (Theorem 3). The search of k is linear.

The edges outside subgraphs H can be colored arbitrarily (sometimes by additional colors). The outgoing edges of every state are colored by different colors.

3 The Algorithm for k-Synchronizing Coloring

Let us begin from an arbitrary coloring of a directed graph Γ with n vertices and d outgoing edges of any vertex. The considered d colors define d spanning subgraphs of the graph. We find all SCC H having no outgoing edges and study every such SCC H separately.

We keep images of vertices and colored edges from the generic graph by any transformation and homomorphism.

If there exists a loop in Γ then let us color the edges of a tree with root in the vertex of a loop by one color. The other edges may be colored arbitrarily. The coloring in this case is synchronizing for the considered SCC.

In the case of two incoming bunches of one vertex the beginnings of these bunches form a stable pair by any coloring (Lemma 4). We merge both vertices in the homomorphic image of the graph (Theorem 1) and obtain according to the theorem a new graph of the size less than $|\Gamma|$.

The linear search of two incoming bunches and of loop can be made at any stage of the algorithm.

Find the parameters of the spanning subgraph: levels of all vertices, the number of vertices (edges) in cycles, for any vertex let us keep its tree and the cycle of the root of the tree. We form the set of vertices of maximal level and choose from the set of trees a tree T with vertex $\mathbf{p}$ of maximal level. This step is linear and used by any recoloring.

Lemma 10. *Let graph Γ have two cycles C_u and C_v either with one common vertex $\mathbf{p}_1$ or with a common sequence $\mathbf{p}_1$,..., $\mathbf{p}_k$, such that all incoming edges of $\mathbf{p}_i$ form a bunch from $\mathbf{p}_{i+1}$ $(i < k)$. Let $u \in C_u$ and $v \in C_v$ be the edges of the cycles leaving $\mathbf{p}_1$. Let T be a maximal subtree of Γ with the root $\mathbf{p}_1$ and edges from C_u and C_v except u and v.*

Then the the adding one of the edges u or v turns the subtree T into precisely one maximal tree of the spanning subgraph.

Proof. Let us add to T either u or v and then find the maximal levels of vertices in both cases. The vertex $\mathbf{p}_i$ for $i > 1$ could not be the root of a tree. If any tree of spanning subgraph with vertex of maximal level has the root $\mathbf{p}_1$ then in both cases the lemma holds. If some tree of spanning subgraph with vertex of maximal level has the root only on C_u then let us choose the adding of v. In this case the level of the considered vertex is growing and only the new tree with root $\mathbf{p}_1$ has vertices of maximal level. In the case of the root on C_v let us add u.

1) If there are two cycles with one common vertex then we use Lemma 10 and find a spanning subgraph R_1 such that any vertex $\mathbf{p}$ of maximal level L belongs to one tree with root on a cycle H. Then after coloring edges of R_1 with the color α we find stable pair $\mathbf{q} = \mathbf{p}\alpha^{L-1+|H|}$ and $\mathbf{s} = \mathbf{p}\alpha^{\mathbf{L-1}}$ (Lemma 7) and go to step 3). The search of a stable pair in this case is linear and the whole algorithm therefore is quadratic.

2) Let us consider now the three replacements from Lemma 7 and find the number of edges in cycles and other parameters of the spanning subgraph of the given color. If the number of edges in the cycles is growing, then the new spanning subgraph must be considered and the new parameters of the subgraph must be found. In the opposite case, after at most $3d$ steps, by Lemma 7, there exists a tree T_1 with root on the cycle H_1 of a spanning subgraph R_1 such that any vertex $\mathbf{p}$ of maximal level L belongs to T_1.

Suppose the edges of R_1 are colored by the color α. Then the vertices $\mathbf{q} = \mathbf{p}\alpha^{L-1+|H_1|}$ and $\mathbf{s} = \mathbf{p}\alpha^{\mathbf{L-1}}$ by Lemma 7 form a stable pair.

3) Let us finish the coloring and find the subsequent stable pairs of the pair $(\mathbf{s}, \mathbf{q})$ using appropriate coloring. Then we go to the homomorphic image Γ_i/ρ (Theorem 1) of the considered graph Γ_i ($O(|\Gamma_i|m_i d)$ complexity where m_i is the size of the map Γ_i). Then we repeat the procedure with a new graph Γ_{i+1} of a smaller size. So the overall complexity of this step of the algorithm is $O(n^2 d)$ in the majority of cases and $O(n^3 d)$ if the number of edges in cycles grows slowly, m_i decreases also slowly, loops do not appear and the case of two ingoing bunches rarely emerges (the worst case).

Let $\Gamma_{i+1} = \Gamma_i/\rho_{i+1}$ on some stage $i+1$ have k-synchronizing coloring. For every stable pair $\mathbf{q}, \mathbf{p}$ of vertices from Γ_i there exists a pair of corresponding outgoing edges that reach either another stable pair or one vertex. This pair of edges is mapped on one image edge of Γ_{i+1}. So let us give the color of the image to preimages and obtain in this way a k-synchronizing coloring of Γ_i. This step is linear. So the overall complexity of the algorithm is $O(n^3 d)$ in the worst case. If Lemma 10 is applicable, then the complexity is reduced at this branch of the algorithm (as well as at some other branches) to $O(n^2 d)$. The practical study of the implementation of the algorithm demonstrates mainly $O(n^2 d)$ time complexity.

4 Conclusion

In this paper, we have continued the study of k-synchronizing coloring of a finite directed graph. It is an important and natural generalization of the well known notion of the synchronizing coloring. This new approach gives an opportunity to extend essentially the class of studied graphs, removing the restriction on the size of the great common divisor of the lengths of the cycles of the graph. The restriction is inspired by the Road Coloring Problem and now for k-synchronizing coloring can be omitted.

There exists still another restriction on the properties of the considered graphs concerning the uniform outdegree of the vertex. However, it also can be omitted by consideration of subgraphs of the graph (Corollary 8).

The practical use of this investigation needs an algorithm and the paper presents a corresponding polynomial time algorithm of the road coloring for the k-synchronization implemented in the package TESTAS. The realization of the algorithm demonstrates a new linear visualization program [3].

References

1. Adler, R.L., Weiss, B.: Similarity of automorphisms of the torus, Memoirs of the Amer. Math. Soc., Providence, RI, 98 (1970)
2. Aho, A., Hopcroft, J., Ulman, J.: The Design and Analisys of Computer Algorithms. Addison-Wesley, Reading (1974)
3. Bauer, T., Cohen, N., Trahtman, A.N.: The visualization of digraph based on its structure properties. In: RTAGS, Ramat-Gan, Israel (2007)
4. Béal, M.P., Perrin, D.: A quadratic algorithm for road coloring (2008), arXiv: 0803.0726v2 [cs.DM]
5. Budzban, G., Feinsilver, P.: The Generalized Road Coloring Problem (2008), arXiv:0903.0192v, [cs.DM]
6. Culik, K., Karhumaki, J., Kari, J.: A note on synchronized automata and Road Coloring Problem. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 175–185. Springer, Heidelberg (2002)
7. Černy, J.: Poznamka k homogenym eksperimentom s konechnymi automatami. Math.-Fyz. Čas. 14, 208–215 (1964)
8. Kari, J.: Synchronizing finite automata on Eulerian digraphs. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 432–438. Springer, Heidelberg (2001)
9. Mateescu, A., Salomaa, A.: Many-valued truth function. Černy conjuncture and road coloring, EATCS Bulletin 68, 134–150 (1999)
10. Trahtman, A.N.: Synchronizing Road Coloring. In: 5-th IFIP WCC-TCS, vol. 273, pp. 43–53. Springer, Heidelberg (2008)
11. Trahtman, A.N.: The Road Coloring and Cerny Conjecture. In: Proc. of Prague Stringology Conference, pp. 1–12 (2008)
12. Trahtman, A.N.: The Road Coloring for Mapping on k States (2008), arXiv:0812.4798, [cs.DM]
13. Trahtman, A.N.: The road coloring problem. Israel Journal of Math. 1(172), 51–60 (2009)

An Encoding Invariant Version of Polynomial Time Computable Distributions

Nikolay Vereshchagin*

Moscow State University, Leninskie gory 1,
Moscow 119991, Russia
ver@mccme.ru
http://lpcs.math.msu.su/~ver

Abstract. When we represent a decision problem, like CIRCUIT-SAT, as a language over the binary alphabet, we usually do not specify how to encode instances by binary strings. This relies on the empirical observation that the truth of a statement of the form "CIRCUIT-SAT belongs to a complexity class C" does not depend on the encoding, provided both the encoding and the class C are "natural". In this sense most of the Complexity theory is "encoding invariant".

The notion of a polynomial time computable distribution from Average Case Complexity is one of the exceptions from this rule. It might happen that a distribution over some objects, like circuits, is polynomial time computable in one encoding and is not polynomial time computable in the other encoding. In this paper we suggest an encoding invariant generalization of a notion of a polynomial time computable distribution. The completeness proofs of known distributional problems, like Bounded Halting, are simpler for the new class than for polynomial time computable distributions.

This paper has no new technical contributions. All the statements are proved using the known techniques.

1 Polynomial Time Samplable and Computable Distributions

Let us specify first what we mean by "encoding invariant" notions in Complexity theory. Fix a set X of "objects" (like boolean circuits). An *encoding of* X is an injective mapping g from X to the binary strings. To every decision problem L with instances from X and every encoding g of X we assign the language

$$L_g = \{g(x) \mid x \text{ is a YES-instance of } L\}.$$

We say that decision problem L in encoding g belongs to a complexity class C if $L_g \in C$.

It turns out that different "natural" encodings of boolean circuits are equivalent in the following sense. We call g_1 and g_2 *poly-time equivalent* if both functions $g_1(g_2^{-1}(x))$ and $g_2(g_1^{-1}(x))$ are computable in polynomial time. Note that

* The work was in part supported by a RFBR grant 09-01-00709.

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 371–383, 2010.

these functions map strings to strings and thus the notion of polynomial time computability is meaningful in this context.

If encodings g_1 and g_2 are poly-time equivalent then $L_{g_1} = h(L_{g_2})$ for a polynomial time computable and invertible partial function (namely for $h(x) = g_1(g_2^{-1}(x))$). Let us call a class C of languages over a binary alphabet *encoding invariant* if for every $L \in C$ and every polynomial time computable and invertible partial function h we have $h(L) \in C$. Natural complexity classes above P, like NP or BPP, are encoding invariant.

We do we care about encoding invariance? The point is that natural encodings of the same natural set of objects X are poly-time equivalent (like in the case of boolean circuits). Thus we can freely say that a decision problem L is in an encoding invariant class C meaning that $L_g \in C$ without specifying the encoding g used. For encoding non-invariant class, like the class of languages having AC^0 circuits of depth 5 (say), we should carefully specify the encoding used, which is not convenient. For example, the question of whether the problem "Boolean circuit evaluation" can be solved by AC^0 circuits of depth 5 is meaningless.

Now we are going to introduce the main notions from Average Case Complexity. We will follow the paper [1] of Bogdanov and Trevisan. The main goal of the theory is to show that certain NP problems are "hard on average". More specifically, we want to provide evidence that for certain NP problems L there is a "simple" probability distribution D on their instances such that every efficient algorithm errs with non-negligible probability on a randomly chosen instance. To this end we define reductions between pairs (problem, distribution) and show that the pair in question is complete in a large class of such pairs.

Definition 1 ([2,1]). *A* distribution over $\{0,1\}^*$ *is a function D from $\{0,1\}^*$ to the non-negative reals such that $\sum_x D(x) = 1$. An* ensemble of distributions $\mathcal{D}$ *is a sequence*

$$D_0, D_1, \ldots, D_n, \ldots$$

of probability distributions over $\{0,1\}^*$. (The parameter n is called the security parameter*.) We say that ensemble $\mathcal{D}$ is* polynomial time samplable *if there is a randomized algorithm A that with an input n outputs a string in $\{0,1\}^*$ and:*

- *There is a polynomial p such that, on input n, A runs in time at most $p(n)$ regardless of its internal coin tosses;*
- *For every n and every $x \in \{0,1\}^*$, $\mathbf{Pr}[A(n) = x] = D_n(x)$.*

We will call such an algorithm A a sampler for $\mathcal{D}$.

A distributional problem *is a pair $(L, \mathcal{D})$ where L is a language over the binary alphabet and $\mathcal{D}$ is an ensemble of distributions. We say that a distributional problem $(L, \mathcal{D})$ is in $(NP, PSamp)$ if $L \in NP$ and $\mathcal{D}$ is polynomial time samplable.*

Note that, in the defintion of polynomial time samplability, we do not require that the support of distribution D_n consist of strings of length exactly n. From the definition it follows only that it consists of strings of length at most $p(n)$.

The following straightforward lemma states that PSamp is an encoding invariant class. Assume that we are given computable injective mappings (encodings)

g_1, g_2 from a set X of objects to the binary strings. Assume that D is a probability distribution over binary strings that represents a distribution over X in encoding g_1. Then $D(g_1(g_2^{-1}(x)))$ represents the same distribution in encoding g_2. Obviously $h(x) = g_1(g_2^{-1}(x))$ is a partial polynomial time computable and invertible injective function (i.e., both $h(x), h^{-1}(x)$ are computable in time polynomial in the length of x) and the support of D is included in the range of h.

Lemma 1. *Assume that $\mathcal{D}$ is polynomial time samplable and $h : \{0,1\}^* \to \{0,1\}^*$ is a partial polynomial time computable and invertible injective function and for all n the support of D_n is included in the range of h. Then the ensemble $\mathcal{D}^h$ where*

$$D_n^h(x) = \begin{cases} D_n(h(x)) & \text{if } h(x) \text{ is defined,} \\ 0 & \text{otherwise} \end{cases}$$

is polynomial time samplable.

The goal of Average Case Complexity is to show that certain distributional problems are (NP, PSamp) complete under reductions of certain kinds which preserve "simplicity on average". We will not define here the type of reductions used in the definition of completeness, and refer to [2,4,1] for the definition. We will define simplified reductions (Definition 3.1 from [1]) that come back to [5]. These simplified reductions will suffice for the goal of this paper.

Definition 2. *We say that ensemble $\mathcal{D}$ is* dominated *by an ensemble $\mathcal{D}'$ if there is a polynomial $p(n)$ such that for all n, x,*

$$D_n(x) \leq p(n) D'_n(x).$$

We say that $(L, \mathcal{D})$ reduces to *$(L', \mathcal{D}')$ if there is a function $f(x, n)$ that for every n and every x in the support of D_n can be computed in time polynomial in n and*

- *(Correctness) $x \in L$ if and only if $f(x, n) \in L'$;*
- *(Domination) There is a polynomial $q(n)$ such that the distribution*

$$D''_n(y) = \sum_{x : f(x,n)=y} D_n(x)$$

is dominated by $D'_{q(n)}(y)$.

In particular $(L, \mathcal{D})$ always reduces to $(L, \mathcal{D}')$ provided $\mathcal{D}'$ dominates $\mathcal{D}$.

The following argument justifies this defintion. Assume that there is an algorithm A that on input x and parameter n solves decision problem L' with a negligible error probability ε_n for x randomly chosen with respect to D'_n. Then the algorithm $A(f(x, n), q(n))$ solves decision problem L with (negligible) error probability $p(n)\varepsilon_{q(n)}$ with respect to distribution D_n.

Remark 1. If the function f is injective (for every fixed n) then the domination condition is easier to understand. In this case $D''_n(f(x,n)) = D_n(x)$ and the domination condition boils down to requiring that

$$D_n(x) \le p(n) D'_{q(n)}(f(x,n))$$

for some polynomials p, q.

Now we are going to define polynomial time computable ensembles of distributions. We will follow the exposition of [1]. Let $\preceq$ denote the lexicographic ordering between bit strings. If D is a distribution over $\{0,1\}^*$ we define

$$f_D(x) = D(\{y \mid y \preceq x\}) = \sum_{y \preceq x} D(y).$$

The function f_D is called *cumulative* probability for distribution D.

Definition 3 (Levin [5], Bogdanov–Trevisan [1]). *We say that ensemble $\mathcal{D}$ is* polynomial time computable *if there is an algorithm that given an integer n and a string $x \in \{0,1\}^*$, runs in time polynomial in n and computes $f_{D_n}(x)$. Let $(NP, PComp)$ stand for the class of distributional problems $(L, \mathcal{D})$ where $L \in NP$ and $\mathcal{D}$ is polynomial time computable.*

Neither Levin, nor Bogdanov and Trevisan specify the meaning of polynomial time computability of a real valued function. To interpret Defintion 3 in a right way, we note that [1] claims that polynomial time computability implies polynomial time samplability. Notice that if $\mathcal{D}$ is a polynomial time samplable ensemble then $D_n(x)$ is always a dyadic rational. Therefore we will assume that, in the Definition 3, $f_{D_n}(x)$ is always a dyadic rational and the algorithm computing $f_{D_n}(x)$ outputs the numerator and denominator of $f_{D_n}(x)$ in binary notation. With this interpretation, every polynomial time computable ensemble is indeed polynomial time samplable, see Theorem 2 below.

This interpretation of Definition 3 has an obvious minor point: we restrict possible values of $f_{D_n}(x)$ to dyadic rationals. Therefore it is natural to consider the following relaxation of Definition 3.

Definition 4. *An ensemble $\mathcal{D}$ is called* weakly polynomial time computable *if there is an algorithm that given integers n, m and a string $x \in \{0,1\}^*$, runs in time polynomial in $n+m$ and computes a rational number within a distance at most 2^{-m} from $f_{D_n}(x)$.*

In this definition we allow $D_n(x)$ to be any non-negative real. It is not hard to see that every weakly polynomial time computable ensemble is dominated by a polynomial time computable ensemble (see Theorem 1 below) and thus both definitions are basically the same. In both definitions, the length of all strings in the support of D_n is bounded by a polynomial in n (otherwise it is not clear how a polynomial time algorithm can read x's).

Theorem 1 ([5]). *Every weakly polynomial time computable ensemble is dominated by a polynomial time computable ensemble.*

This theorem was essentially proven in [5] (although not explicitly stated there). For the sake of completeness we present the proof in the Appendix.

It is not clear whether Definitions 3 and 4 are encoding invariant[1]. Indeed, assume that an ensemble of distributions $\mathcal{D}$ is polynomial time computable and h is a polynomial time computable and invertible partial function. There is no guarantee that $\mathcal{D}^h$ is polynomial time computable: the function $h(x)$ might not preserve lexicographical order.

The common scheme to prove $(\mathrm{NP}, \mathrm{PSamp})$ completeness is as follows. Assume that we want to show that $(L, \mathcal{D})$ is $(\mathrm{NP}, \mathrm{PSamp})$ complete. First we show that $(L, \mathcal{D})$ is $(\mathrm{NP}, \mathrm{PComp})$ complete with respect to reductions of Definition 2. Second, we use the result of [4,2] stating that every distributional problem in $(\mathrm{NP}, \mathrm{PSamp})$ reduces to some distributional problem in $(\mathrm{NP}, \mathrm{PComp})$ (using reductions that are weaker than those of Definition 2).

The goal of this paper is to simplify this scheme. Namely, in place of the class $(\mathrm{NP}, \mathrm{PComp})$ we suggest to use a wider class, which we call $(\mathrm{NP}, \mathrm{PISamp})$. Ensembles of distributions from PISamp will be called "polynomial time invertibly samplable". The class PISamp is encoding invariant and proving $(\mathrm{NP}, \mathrm{PISamp})$ completeness is easier than proving $(\mathrm{NP}, \mathrm{PComp})$ completeness.

2 Polynomial Time Invertibly Samplable Distributions

Let A be a polynomial time probabilistic algorithm, as in the definition of a samplable distribution. Think of the source of randomness for A as a real number in the segment $[0, 1)$ with respect to the uniform distribution. More precisely, if a computation of $A(n)$ returns x where $r = r_1 \dots r_l$ is the sequence of outcomes of coin tosses made in that computation, we will think that $A(n)$ maps all reals in the half-interval $[0.r, 0.r + 2^{-l})$ to x. In this way A defines a mapping from the set $\mathbb{N} \times [0, 1)$ to $\{0, 1\}^*$ and we denote by $A(n, \alpha)$ the result of A for input n and randomness $\alpha \in [0, 1)$. Let

$$A^{-1}(n, x) = \{\alpha \in [0, 1) \mid A(n, \alpha) = x\}.$$

In general, $A^{-1}(n, x)$ is a union of a finite number of segments and each of them has the form $[k/2^l, (k + 1)/2^l)$.

Definition 5. *We call A, as above,* polynomial time invertible*, if for all n and all x in the support of D_n, the set $A^{-1}(n, x)$ is* one *subsegment of $[0, 1)$ which can be computed from n, x in time polynomial in n. (Segments are represented by numerators and denominators of their end-points, written in binary notation.) We say that a polynomial time samplable ensemble $\mathcal{D}$ is* polynomial time invertibly samplable *if there is a polynomial time invertible sampler for $\mathcal{D}$.*

[1] And they are not provided one-way permutations exist, see Remark 1.

Remark 2. If $\mathcal{D}$ is polynomial time invertibly samplable then $D_n(x)$ is a dyadic rational that can be computed from x in time polynomial in n. However this does not imply that the cumulative function of D_n is polynomial time computable.

It is easy to see that the class of polynomial time invertibly samplable distribution is encoding invariant (in the sense of Lemma 1). The next theorem shows that PComp $\subseteq$ PISamp and thus PISamp is an encoding invariant generalization of PComp.

Theorem 2 ([5]). *Every polynomial time computable ensemble of distributions is polynomial time invertibly samplable.*

This theorem was essentially proven in [5] (although not explicitly stated there). Actually the main idea of the proof comes back to Fano and Shannon (the so called Shannon-Fano codes). For the sake of completeness we present the proof in the Appendix.

Now we will explain why proving (NP, PISamp) completeness is easier than proving (NP, PComp) completeness. Assume that we have to show that an arbitrary problem $(L', \mathcal{D}') \in$ (NP, PComp) reduces to a fixed (NP, PComp) complete problem $(L, \mathcal{D})$. An analysis of the proof reveals that we use the assumption about polynomial time computability of $\mathcal{D}'$ to construct an invertible sampler for $\mathcal{D}'$. Thus to prove that $(L, \mathcal{D})$ is (NP, PISamp) complete is easier than to prove that it is (NP, PComp) complete: we can skip this step. On the other hand, for all known (NP, PComp) complete problem $(L, \mathcal{D})$ it is immediate that $\mathcal{D}$ is in PISamp. Thus the proof of (NP, PISamp) completeness of $(L, \mathcal{D})$ has one step less than that of its (NP, PComp) completeness. An example of (NP, PISamp) completeness proof is presented in Section 3.

The next theorem shows that under certain plausible assumptions PComp $\neq$ PISamp $\neq$ PSamp.

Theorem 3. *(a) If one-way functions exist, then there is a polynomial time samplable ensemble that is not dominated by any polynomial time invertibly samplable ensemble. (b) Assume that for some polynomial p there is a one-way function $f_n : \{0,1\}^n \to \{0,1\}^{p(n)}$ such that every string $y \in \{0,1\}^{p(n)}$ has at most poly(n) pre-images under f_n. Then there is a polynomial time invertibly samplable ensemble that is not dominated by any polynomial time computable ensemble.*

Item (a) implies that there is a polynomial time samplable ensemble that is not dominated by any polynomial time computable ensemble (provided one-way functions exist). The latter fact was proved in [2] using the techniques of (an earlier version of) [3]. Our proof will use just the result of [3] (the existence of Pseudorandom Generators).

Proof. (a) In [3], it is shown that if one-way functions exist, then there is a Pseudorandom Generator $G_n : \{0,1\}^n \to \{0,1\}^{2n}$. Consider G_n as a sampler and let D_n denote the sampled distribution:

$$D_n(x) = \mathbf{Pr}[G_n(s) = x],$$

where s denotes a randomly chosen string of length n. Assume, by way of contradiction, that $\mathcal{D}$ is dominated by a polynomial time invertibly samplable ensemble $\mathcal{D}'$ so that

$$D_n(x) \leq p(n)D'_n(x) \tag{1}$$

for some polynomial $p(n)$. Using D'_n we will construct a polynomial time test that distinguishes the random variable $G_n(s)$ from the random variable which is uniformly distributed among all strings of length $2n$, which is a contradiction.

As such test consider the set

$$X_n = \{x : |x| = 2n,\ D'_n(x) \geq 2^{-n}/p(n)\}.$$

As $\mathcal{D}'$ is invertible samplable, the set X_n is polynomial time decidable. Indeed, $D'_n(x)$ is a dyadic rational that can be computed from x and n in time polynomial in n. We claim that $G_n(s)$ is in X_n for all s, whereas a random string of length $2n$ falls in X_n with negligible probability $p(n)2^{-n}$.

The first claim follows from (1). Indeed, as G_n is a sampler for D_n, for every $s \in \{0,1\}^*$ we have $D_n(G_n(s)) \geq 2^{-n}$ and hence $D'_n(G_n(s)) \geq 2^{-n}/p(n)$.

The second claim means that $|X_n| \leq 2^n p(n)$. This fact follows from definition of X_n. Indeed, otherwise the cumulative probability of X_n w.r.t. D'_n would be larger than 1:

$$D'_n(X_n) = \sum_{x \in X_n} D'_n(x) \geq |X_n|2^{-n}/p(n) > 1.$$

(b) Let f_n be the function from the assumption in item (b). We claim that the ensemble

$$D_n(z) = \begin{cases} 1/2^n, & \text{if } z = f_n(x)x,\ |x| = n, \\ 0, & \text{otherwise} \end{cases}$$

satisfies the conclusion of item (b).

$\mathcal{D}$ is polynomial time invertibly samplable: The following algorithm A is an invertible sampler for D_n: choose a string x of length n at random, return $f_n(x)x$ and halt. As f_n is polynomial time computable, A is indeed a polynomial time algorithm. Let us show that A is invertible in polynomial time. The following algorithm finds $A^{-1}(z,n)$: represent z in the form yx where $|x| = n$ (if $|z| < n$ then halt); apply f_n to x; if $y \neq f_n(x)$ then halt and otherwise output $[0.x, 0.x + 2^{-n})$ and halt.

$\mathcal{D}$ is not dominated by any polynomial time computable ensemble. Let us first prove a simpler statement: $\mathcal{D}$ is not polynomial time computable. By way of contradiction assume that this is not the case. How do we use this assumption? Under this assumption, we can find in polynomial time $D_n(\{zw \mid w \in \{0,1\}^{p(n)+n-|z|}\})$ for any given z of length at most $p(n)+n$. Indeed, it is equal to $f_{D_n}(z11\ldots1) - f_{D_n}(z'11\ldots1)$ for the predecessor z' of z w.r.t. lexicographical ordering. Here it is important that in the lexicographical ordering the first bits are the most significant ones.

We will show that given *any* y in the range of f_n we can find by binary search in polynomial time its lexicographical first pre-image. To this end we use the following simple fact: a string y has a pre-image with prefix z iff

$$D_n(\{yzw \mid w \in \{0,1\}^{n-|z|}\}) \geq 2^{-n}. \tag{2}$$

Recall that for any y, z we can decide in polynomial time whether (2) holds. Thus given any y of length $p(n)$ we can first find whether it has a pre-image at all, which happens iff (2) holds for the empty z. If this is the case we can find whether y has a pre-image starting with 0, which happens iff (2) holds for $z = 0$, and so on.

Now we will prove that $\mathcal{D}$ is not dominated by any polynomial time computable ensemble. By way a contradiction, assume that $\mathcal{D}'$ is polynomial time computable ensemble with

$$D'_n(z) \geq D_n(z)/q(n)$$

for a polynomial q. We will construct a polynomial time algorithm that inverts $f_n(x)$ for at least half of all x's of length n, which is a contradiction with non-invertibility of f_n.

Let us try first the same algorithm as before, but this time using D'_n instead of D_n. If y has a pre-image with prefix z then

$$D'_n(\{yzw \mid w \in \{0,1\}^{n-|z|}\}) \geq D_n(\{yzw \mid w \in \{0,1\}^{n-|z|}\})/q(n) \geq 1/2^n q(n),$$

but, unfortunately, not the other way around. Indeed, it may happen that

$$D'_n(\{yzw \mid w \in \{0,1\}^{n-|z|}\}) \geq 1/2^n q(n) \tag{3}$$

but y has no pre-image with prefix z.

How to fix this? Note that if we find the list of all x's with $D'_n(yx) \geq 1/2^n q(n)$ then we are done, as all pre-images of y are in that list and we thus can find one of them by probing all x's from the list. The problem is that the list can be super-polynomially big, and thus there is no hope to find it in polynomial time. The solution is as follows: we mentally divide all y's into "good" and "bad" ones. For good y's the set of x's with $D'_n(yx) \geq 1/2^n q(n)$ will have at most polynomial number of elements, and we will be able to find all of them by binary search, as before. For bad y's, we know nothing about the cardinality of this set. However, $f_n(x)$ will be bad for at most half of x of length n, and recall that we tolerate the probability $1/2$ of failure.

Specifically, call y *good* if

$$D'_n(\{yx : |x| = n\}) \leq s(n)/2^{n-1} \tag{4}$$

where $s(n)$ stands for a polynomial upper bound for the number of pre-images under f_n.

We have to prove that there is a polynomial time algorithm that finds a pre-image of every given good y in the range of f_n and that $f_n(x)$ is bad for at most half of x of length n.

Assume that y is good, i.e., Equation (4) holds. There are at most $2q(n)s(n)$ strings x with $D'_n(yx) \geq 1/2^n q(n)$, as otherwise the sum of their probabilities would exceed $s(n)/2^{n-1}$. Moreover, there are at most $2q(n)s(n)$ pairwise incomparable strings z such that (3) holds (we call z' and z'' incomparable if neither is a prefix of the other). Therefore we can find by binary search in polynomial time all maximal z's satisfying (3) (we call z maximal, if no its proper extension satisfies (3)). If a pre-image of y exists, the it is certainly among those z's.

Assume that $f_n(x)$ is bad. That is, x happens to be a pre-image of a bad y. The number of bad y's is at most $2^{n-1}/s(n)$. Indeed, otherwise the sum of $D'_n(\{yx : |x| = n\})$ over all bad y's would exceed 1. Every bad y has by assumption at most $s(n)$ pre-images. Hence the number of x such that $f_n(x)$ is bad is at most 2^{n-1}.

Remark 3. Under the same hypothesis and by the same arguments, as in item (b), we can show that the class PComp is not encoding invariant. Indeed, using the notations from the proof, let h be cyclic shift by n bits on strings of length $n+p(n)$: $h(xy) = yx$. Obviously, h is polynomial time computable and invertible. The distribution $\mathcal{D}^h$ is then defined by the equation

$$D^h_n(z) = \begin{cases} 1/2^n, & \text{if } z = xf_n(x),\ |x| = n, \\ 0, & \text{otherwise.} \end{cases}$$

It is not hard to see that $\mathcal{D}^h$ is polynomial time computable.

3 Completeness Proof

Let $(\mathrm{NP}, \mathrm{PISamp})$ denote the class of all distributional problem $(L, \mathcal{D})$ such that $L \in \mathrm{NP}$ and $\mathcal{D}$ is polynomial time invertibly samplable. We will prove that the "Bounded halting" distributional problem is $(\mathrm{NP}, \mathrm{PISamp})$ complete.

Definition 6. *(Bounded halting distributional problem). Let*

$$BH = \{(M, x, 1^t) \mid M \text{ is a program of a non-deterministic Turing machine that accepts } x \text{ in at most } t \text{ steps}\}$$

We assume here that programs of Turing machines are encoded by binary strings in such a way that a universal Turing machine can simulate M with overhead $poly(|M|)$: each step of machine M is simulated in $poly(|M|)$ steps. We also assume that triples of strings are encoded by strings so that both encoding and decoding functions are polynomial time computable.

Define the probability distribution U_n on triples of binary strings as follows:

$$U_n(M, x, 1^t) = \begin{cases} 2^{-2l-|x|-|M|}, & \text{if } |x| < 2^l,\ |M| < 2^l,\ t = n, \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $l = \lceil \log_2 n \rceil$.

Lemma 2. *Ensemble $\mathcal{U}$ is polynomial time invertibly samplable.*

Proof. The sampler $A(n)$ computes $l = \lceil \log_2 n \rceil$, tosses the coin $2l$ times and interprets the outcome as two integers a, b in the range $0, \ldots, 2^l - 1$. Then it tosses a coin a times to obtain M, then b times to obtain x and finally returns the triple $M, x, 1^n$. It is straightforward that A is polynomial time invertible.

Theorem 4 ([5]). *The distributional problem $(BH,\mathcal{U})$ is $(NP, PISamp)$ complete. That is, every distributional problem $(L,\mathcal{D}) \in (NP, PISamp)$ reduces to $(BH,\mathcal{U})$ in the sense of Definition 2.*

The proof of this theorem is not novice: basically, it is a part of the proof from [5] of $(\mathrm{NP}, \mathrm{PComp})$ completeness of $(\mathrm{BH},\mathcal{U})$. More specifically, to prove Theorem 4 we just skip in the latter proof its first step, which is basically proving that the given ensemble of distributions is in PISamp.

We start the proof with a lemma. We will call segments of the form $[0.r0, 0.r1)$ *standard segments*. The length of this segment is $2^{-|r|}$. The string r will be called the *name* of $[0.r0, 0.r1)$.

Lemma 3. *Let $\mathcal{D}$ be a polynomial time invertibly samplable ensemble and A an polynomial time invertible sampler for $\mathcal{D}$. Given n and any x in the support of D_n we can compute in time polynomial in n a standard segment $I(x,n)$ of length at least $D_n(x)/4$ with $I(x,n) \subseteq A^{-1}(n,x)$.*

Proof. Let $I(x,n)$ be a largest standard segment inside $A^{-1}(n,x)$. By assumption we can find $A^{-1}(n,x)$ and hence $I(x,n)$ in polynomial time from x, n.[2] It remains to show that its length is at most 4 times less than $D_n(x)$. Indeed, otherwise the segment $A^{-1}(n,x)$ contains three consecutive standard segments. Then a pair of them (the first and the second ones or the second and the third ones) can be united into a larger standard segment inside $A^{-1}(n,x)$.

Proof (Proof of Theorem 4). Assume that $(L,\mathcal{D})$ is the given distributional problem to reduce to $(\mathrm{BH},\mathcal{U})$. Why not to do it in the standard way? Namely, fix a non-deterministic machine M accepting L in polynomial time $q(n)$ and consider the reduction

$$f(x,n) = (M, x, 1^{q(n)}). \tag{6}$$

This reduction certainly satisfies the correctness requirement. However it might violate the domination condition. As f is injective, the domination requirement is met iff

$$D_n(x) \le p(n) U_{q(n)}(M, x, 1^{q(n)})$$

for some polynomial p. Up to a polynomial factor, $U_{q(n)}(M, x, 1^{q(n)})$ equals $2^{-|x|}$, which might be much smaller than $D_n(x)$. Thus to ensure the domination condition we need to replace x in the right hand side of (6) by its representation $\hat{x}$

[2] To find the largest standard segment inside a given segment, we can use binary search.

in at most $-\log_2 D_n(x) + O(\log n)$ bits so that $2^{-|\hat{x}|} \geq D_n(x)/p(n)$ for a polynomial p. This is can be done using Lemma 3. Indeed, fix an invertible sampler A for $\mathcal{D}$ and let $I(x,n)$ be the mapping existing by Lemma 3. Let $r_{x,n}$ stand for the name of $I(x,n)$. The string $r_{x,n}$ together with n form a desired succinct representation of x (we need n, as the algorithm A cannot find x from $r_{x,n}$ alone). The number n should be represented by its "self-delimiting" description $\hat{n}$ in $O(\log n)$ bits . Indeed, we need to parse $\hat{n}r_{n,x}$ into $\hat{n}$ and $r_{n,x}$. For instance, we can set $\hat{n}$ to be the binary notation of n with all bits doubled and appended by 01 (e.g., $\hat{5} = 11001101$) so that $|\hat{n}| \leq 2\log_2 n + 2$.

So the reduction f is defined by

$$f(x,n) = (N, \hat{n}r_{x,n}, 1^{q(n)}) \tag{7}$$

where N is a non-deterministic (not necessarily polynomial time) Turing machine working as follows:

(1) on input z, reject if z is not of the form $\hat{n}r$ for any natural n,
(2) otherwise (when $z = \hat{n}r$) interpret r as the name of a standard segment $I \subseteq [0,1)$,
(3) run $A(n)$ using any real in I as randomness needed for $A(n)$; let x be the output of $A(n)$,
(4) run $M(x)$ where M is the fixed non-deterministic polynomial time machine accepting L.

The running time of N for input $\hat{n}r_{x,n}$ is the sum of the running time of $A(n)$, which is polynomial in n, and the running time of $M(x)$, which is also polynomial in n, as so is the length of x. Thus N works in time $q(n)$ for all inputs $\hat{n}r_{x,n}$, where q is a polynomial. This polynomial should be used in (7). Then for every x in the support of D_n machine N accepts $\hat{n}r_{x,n}$ iff M accepts x. Therefore the reduction (7) satisfies the Correctness property. The domination condition holds by the choice of $r_{x,n}$.

4 Conclusion

We have observed that the notion of a polynomial time computable ensemble of distributions is probably not encoding invariant. We have suggested a relaxation of PComp, called PISamp, so that PISamp is encoding invariant. We have provided an evidence that PISamp might be strictly larger than PComp. The notion of (NP, PISamp) completeness may be used instead of that of (NP, PComp) completeness in proofs of (NP,PSamp) completeness, which makes those proofs a little easier.

Acknowledgements

The author is grateful to Alexander Shen for bringing his attention to the fact that PComp might be not encoding invariant. The author is grateful to Dmitry Itsykson and to anonymous referees for a number of helpful suggestions.

References

1. Bogdanov, A., Trevisan, L.: Average-Case Complexity. Foundations and Trends in Theoretical Computer Science 1(2), 1–106 (2006)
2. Ben-David, S., Chor, B., Goldreich, O., Luby, M.: On the Theory of Average Case Complexity. Journal of Computer and System Sciences 44(2), 193–219 (1992)
3. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A Pseudorandom Generator from any One-way Function. SIAM Journal on Computing 28, 12–24 (1999)
4. Impagliazzo, R., Levin, L.A.: No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random. In: FOCS 1990, pp. 812–821 (1990)
5. Levin, L.A.: Average Case Complete Problems. SIAM J. Comput. 15(1), 285–286 (1986)

Appendix

Proof of Theorem 1

It suffices to prove the statement for weakly polynomial time computable ensembles that have the following property: there is a polynomial $p(n)$ such the support of D_n is the set of all strings of length less than $p(n)$ and $D_n(x) \geq 2^{-p(n)-1}$ for all strings in the support of D_n. Indeed, let $p(n)$ stand for a polynomial such that the length of all strings in the support of D_n is strictly less than $p(n)$. Consider the distribution D'_n which is the arithmetic mean of D_n and the uniform distribution over all strings of length less than $p(n)$ (the latter assigns probability $1/(2^{p(n)+1}-1)$ to all strings of length less than $p(n)$). Obviously, $\mathcal{D}'$ is weakly polynomial time computable, dominates D_n and $D'_n(x) > 2^{-p(n)-1}$ for all strings in the support of D'_n. As the domination relation is transitive, every ensemble dominating $\mathcal{D}'$ also dominates $\mathcal{D}$.

So assume that $\mathcal{D}$ is weakly polynomial time computed by an algorithm A and $D_n(x) \geq 2^{-p(n)-1}$ for some polynomial $p(n)$ and all x in the support of D_n, which consists of all strings of length less than $p(n)$. Let $f_n(x)$ stand for the dyadic rational number that is produced by A for the input triple $(n, p(n)+3, x)$. Let, additionally, $f_n(x) = 1$ for the lexicographical largest string x of length less than $p(n)$. Finally, let $D'_n(x)$ denote the difference of $f_n(x)$ and $f_n(y)$ for the predecessor y of x. Then the values of $D'_n(x)$ sum up to 1. We claim that $D'_n(x)$ is always non-negative and dominates $D_n(x)$ and thus satisfies the theorem.

By construction we have

$$|f_{D_n}(z) - f_n(z)| \leq 2^{-p(n)-3}$$

for all z. Applying this inequality to x and its predecessor we conclude that

$$|D'_n(x) - D_n(x)| \leq 2^{-p(n)-2} \leq D_n(x)/2$$

and hence $D'_n(x) \geq D_n(x)/2$.

Proof of Theorem 2

As $f_{D_n}(x)$ is computable in time $p(n)$ (for some polynomial p), the values of $f_{D_n}(x)$ are always of the form $k/2^l$ where $l \leq p(n)$ and $k \leq 2^l$.

The sampler A for D_n is as follows:

> Choose at random a segment I of the form $\big[k/2^{p(n)}, (k+1)/2^{p(n)}\big)$ with $k < 2^{p(n)}$ (tossing a coin $p(n)$ times).
> Using a binary search, find x such that
>
> $$I \subseteq \big[\sum_{y \prec x} D_n(y), \sum_{y \preceq x} D_n(y) + D_n(x)\big).$$
>
> Output x and halt. Here we use the fact that the cumulative distribution is polynomial time computable. We use also that the length of all x's in the support of D_n is bounded by a polynomial in n and thus binary search finishes in a polynomial number of steps.

The algorithm A is polynomial time invertible. Indeed,

$$A^{-1}(n, x) = \big[\sum_{y \prec x} D_n(y), \sum_{y \preceq x} D_n(y) + D_n(x)\big)$$

for every x in the support of A. The endpoints of this interval are $f_{D_n}(x)$ and $f_{D_n}(y)$ for predecessor y of x, thus poly-time computable by the assumptions of the theorem.

Prehistoric Phenomena and Self-referentiality

Junhua Yu

Department of Philosophy, Tsinghua University
Beijing, 100084, China
DF7G5036@hotmail.com

Abstract. By terms-allowed-in-types capacity, the Logic of Proofs **LP** enjoys a system of advanced combinatory terms, while including types of the form $t:\phi(t)$, which have self-referential meanings. This paper suggests a research on possible **S4** measures of self-referentiality introduced by this capacity. Specifically, we define "prehistoric phenomena" in **G3s**, a Gentzen-style formulation of modal logic **S4**. A special phenomenon, namely, "left prehistoric loop", is then shown to be necessary for self-referentiality in realizations of **S4** theorems in **LP**.

1 Introduction

The Logic of Proofs **LP** is introduced systematically in [1] by Sergei Artëmov, where it is shown to be the explicit counterpart of modal logic **S4** by verifying the realization theorem. With terms being allowed in types, **LP** has its polynomials as advanced combinatory terms, and hence, extends the idea of propositions-as-types in proof theory. By this new capacity, types of the form $t:\phi(t)$ are also included. This sort of types, however, has self-referential meaning, and hence, may indicate some essential properties of this capacity. As [6] says, by any arithmetical semantic $*$, $t:\phi(t)$ is interpreted to be the arithmetical sentence $Proof(t^*, \ulcorner(\phi(t))^*\urcorner)$, which is not true in Peano Arithmetic with the standard Gödel numbering, since the Gödel number of a proof can not be smaller than that of its conclusion.

Roman Kuznets has studied this issue and verified the following meaningful result: *there is an* **S4**$-$*theorem,* $\neg\Box\neg(p \to \Box p)$*, with* ***any*** *realizations of it calling for self-referential constant specifications* (see Theorem 3, also [2] and [5]). In Kuznets's papers, self-referentiality was studied at a "logic-level", i.e., whether or not a modal logic can be realized non-self-referentially.

Correspondingly, it is also interesting to consider this topic at a "theorem-level", i.e., self-referentiality in realizations of specified theorems. That is, which **S4**$-$theorems have to call for self-referential constant specifications to prove their realized forms in **LP**? Are there any easy criteria for this? Roughly speaking, if we can fix the class of non-self-referential-realizable **S4**$-$theorems, then we may find some **S4** (and then, intuitionistic) measure of self-referentiality introduced by the terms-allowed-in-types capacity.

In this paper, we define "prehistoric phenomena" in **G3s**, a Gentzen-style formulation of **S4**. This notion is then used to study self-referentiality in realization

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 384–396, 2010.

procedure. A special prehistoric phenomenon, i.e., left-prehistoric-loop, is shown to be necessary for self-referentiality.

At beginning, we enumerate some preliminary notions and results which will be referred directly in this paper. In [7] (see also [2]), a Gentzen-style formulation of **S4**, which enjoys Cut-elimination and subformula property, was presented.

Definition 1 ([7]). G3s *has standard G3 (atomic) axioms (i.e.,* $p, \Gamma \Rightarrow \Delta, p$ *and* $\bot, \Gamma \Rightarrow \Delta$*) and rules together with the following two rules for modalities:*

$$(L\Box)\ \frac{\phi, \Box\phi, \Gamma \Rightarrow \Delta}{\Box\phi, \Gamma \Rightarrow \Delta} \qquad (R\Box)\ \frac{\Box\phi_1, \cdots, \Box\phi_n \Rightarrow \psi}{\Box\phi_1, \cdots, \Box\phi_n, \Gamma \Rightarrow \Delta, \Box\psi}\ .$$

The notion of "family (of $\Box$'s)" was defined in [1]. In a **G3s**−rule: (1) Each occurrence of $\Box$ in a side formula ϕ in a premise is related only to the corresponding occurrence of $\Box$ in ϕ in the conclusion; (2) Each occurrence of $\Box$ in an active formula in a premise is related only to the corresponding occurrence of $\Box$ in the principal formula of the conclusion. A *family* of $\Box$'s is an equivalence class generated by the relation above. We denote families by $f_0, f_1, \cdots$. All rules of **G3s** respect the polarity of formulas, and hence, each family consists of $\Box$'s of a same polarity. A family is *positive (negative)* if it consists of positive (negative) $\Box$'s. If a positive family has at least one of its $\Box$'s correspond to the principal $\Box$ of an $(R\Box)$ rule, then this family is *principal*. Positive families that are not principal are *non-principal* families.

LP enjoys a special AN rule, which allows to introduce formulas of the form $c : A$, where c is a constant, and A is an axiom, at any time. A *constant specification* $\mathcal{CS}$ is a set of **LP**−formulas (see [1]) of the form $c{:}A$. $\mathcal{CS}$ is *injective* if for each c there is at most one formula $c{:}A$ in $\mathcal{CS}$. $\mathcal{CS}$ is *non-direct-self-referential* (or *non-self-referential*, respectively) if $\mathcal{CS}$ does not contain any formulas (or subsets) of the form $c{:}A(c)$ (or $\{\, c_1{:}A_1(c_2), \cdots, c_{n-1}{:}A_{n-1}(c_n), c_n{:}A_n(c_1)\,\}$). A non-direct-self-referential (or non-self-referential) constant specification is denoted by $\mathcal{CS}^*$ (or $\mathcal{CS}^{\circledast}$). For any constant specification $\mathcal{CS}$, if all AN rules are restricted to formulas in $\mathcal{CS}$, then the resulting system is denoted by $\mathbf{LP}(\mathcal{CS})$. For instance, $\mathbf{LP}(\varnothing)$ is the system obtained by dropping AN from **LP**.

Theorem 1 (Lifting Lemma [1] [2]). *If* $x_1{:}\phi_1, \cdots, x_n{:}\phi_n \vdash_{\mathbf{LP}} \psi$*, then there is a term* $t = t(x_1, \cdots, x_n)$ *s.t.* $x_1{:}\phi_1, \cdots, x_n{:}\phi_n \vdash_{\mathbf{LP}} t(x_1, \cdots, x_n){:}\psi$.

Lemma 1 ([1]). *If* $\Gamma, \phi \vdash_{\mathbf{LP}(\mathcal{CS})} \psi$*, then* $\Gamma \vdash_{\mathbf{LP}(\mathcal{CS})} \phi \to \psi$.

As a Gentzen-style formulation of $\mathbf{LP}(\varnothing)$, $\mathbf{LPG}_0$ is presented in [1] on the propositional base **G2c** from [7]. For convenience, we take **G3c** from [7] as our propositional base of $\mathbf{LPG}_0$. While sharing similar non-modal axioms and rules with **G3s** (formulas in them are **LP**−formulas now), $\mathbf{LPG}_0$ also has the following rules:

$$(L{+}_L)\ \frac{\Gamma \Rightarrow \Delta, t{:}\phi}{\Gamma \Rightarrow \Delta, s + t{:}\phi} \quad (L{+}_R)\ \frac{\Gamma \Rightarrow \Delta, t{:}\phi}{\Gamma \Rightarrow \Delta, t + s{:}\phi} \quad (L{:})\ \frac{\phi, t{:}\phi, \Gamma \Rightarrow \Delta}{t{:}\phi, \Gamma \Rightarrow \Delta}$$

$$(R!)\ \frac{\Gamma \Rightarrow \Delta, t{:}\phi}{\Gamma \Rightarrow \Delta, !t{:}t{:}\phi} \quad (L\cdot)\ \frac{\Gamma \Rightarrow \Delta, s{:}(\phi \to \psi) \quad \Gamma \Rightarrow \Delta, t{:}\phi}{\Gamma \Rightarrow \Delta, s \cdot t{:}\psi}\ .$$

LP and **S4** are linked by *forgetful projection* in [1]. The forgetful projection $\circ$ is a function from the language of **LP** to the language of **S4**, which meets the following clauses:

$$p^\circ = p \qquad \bot^\circ = \bot \qquad (\phi \to \psi)^\circ = \phi^\circ \to \psi^\circ \qquad (t{:}\phi)^\circ = \Box\phi^\circ$$

An **LP**$-$formula ϕ is a *realization* of an **S4**$-$formula ψ, provided $\phi^\circ = \psi$.

Theorem 2 (Realization of S4 [1]). *If* $\vdash_{\mathbf{S4}} \phi$ *then there is an* **LP**$-$*formula* ψ *s.t.* $\vdash_{\mathbf{LP}} \psi$ *and* ψ° *is* ϕ.

In [1], the result above is shown by offering a realization procedure, which can mechanically calculate a suitable realization of an **S4**$-$theorem. In this procedure, the constant specification $\mathcal{CS}$ of the resulting **LP**$-$proof is determined only by employing Lifting Lemma (Theorem 1) while dealing with instances of $(R\Box)$ rules. For details, we refer to [1] and [2]. It is stated in [1, page 27] that the realization procedure there may lead to constant specifications of the sort $c{:}\phi(c)$ where $\phi(c)$ contains c. This sort of formulas is interesting since they have self-referential meanings in both *arithmetical semantics* [1] and *Fitting semantics* [3]. The following result shows that self-referentiality is an essential property of **S4**.

Theorem 3 ([2]). *The* **S4**$-$*theorem*

$$\neg\Box\neg(p \to \Box p) \tag{1}$$

can not be realized in any $\mathbf{LP}(\mathcal{CS}^*)$.

[5] and [6] considered self-referentiality of some other "modal-justification" pairs. The results presented there are: each **K**$-$(or **D**$-$)theorem can be realized with a non-self-referential constant specification, while in $\mathbf{T}, \mathbf{K4}, \mathbf{D4}$, self-referentiality is necessary for realization.

As we have stated at the beginning, self-referentiality will be considered at a "theorem-level", instead of a "logic-level" in this paper. In Section 2, a series of notations about **G3s** and the standard realization procedure are introduced. Then in Section 3, "prehistoric phenomena" in **G3s** are defined, and some results are verified. In Section 4, we prove the necessity of "left prehistoric loop" for self-referentiality in **S4-LP** realization. In Section 5, we list some related open problems, which are suggested for further research.

2 Notations and Preliminary Discussions

We introduce a series of notations in this section. Though some of them seem cumbersome, they are employed with a view of denoting notions in realization procedure in detail[1].

[1] In [4], Melvin Fitting introduces annotated formulas, to display $\Box$$-$families in **G3s**$-$proofs. We will go further by displaying instances of $(R\Box)$ rules.

2.1 On Gentzen-Style Formulation G3s

Observations of [5] and [6] indicate that behaviors of $\Box$−families in Gentzen-style proofs are essential to self-referentiality. In this subsection, we introduce some notations, and then, consider in general the behaviors of $\Box$−families in a **G3s**−proof.

A **G3s**−proof (as a tree) is denoted by $\mathcal{T} = (T, R)$, where the node set $T := \{s_0, s_1, \cdots, s_n\}$ is the set of **occurrences** of sequents, and $R := \{ (s_i, s_j) \in T \times T \mid s_i$ is the conclusion of a rule which has s_j as a premise $\}$. The only root of $\mathcal{T}$ is denoted by s_r. Since each path in $\mathcal{T}$ from s_r is associated with a unique end-node, we can denote paths by their end-nodes. In what follows, whenever we say "path s_0", we mean the only path from s_r to s_0. $\mathcal{T} \restriction s$ is the subtree of $\mathcal{T}$ with root s. As usual, the transitive closure and reflexive-transitive closure of P is denoted by P^+ and P^*, respectively, for any binary relation P.

Sometimes, we take $\boxminus$ ($\boxdot, \boxplus$) to denote a negative (non-principal-positive, principal-positive, respectively) occurrence of $\Box$ in $\mathcal{T}$. Particularly, we take $\oplus$ to denote a principal-positive occurrence of $\Box$ in the conclusion of an $(R\Box)$, if this $\Box$ is just introduced principally[2] in this rule.

In $\mathcal{T}$, we have only finitely many principal-positive families, say, $f_1, \cdots, f_m$. An occurrence of $\boxplus$ of family f_i is denoted by $\boxplus_i$. Related $\boxplus$'s may occur in different sequents of $\mathcal{T}$. We take $\boxplus_i^s$ as the notation for an occurrence of $\boxplus_i$ in sequent s. Note that a family can have more than one occurrences in a sequent. For each family, say f_i, there are only finitely many $(R\Box)$ rules, denoted by $(R\Box)_{i.1}, \cdots, (R\Box)_{i.m_i}$. The $\oplus$'s introduced by them are denoted by $\oplus_{i.1}, \cdots, \oplus_{i.m_i}$. We also use $(R\Box)_i$ and $\oplus_i$, if we only concern the family they belong. In $\mathcal{T}$, the premise (conclusion) of $(R\Box)_{i.j}$ are denoted by $I_{i.j}$ ($O_{i.j}$).

We are now ready to present some properties of $\Box$−families in **G3s**−proofs.

Lemma 2. *In a* **G3s**−*proof* $\mathcal{T}$*, each family has exactly one occurrence in* s_r*.*

Proof. By an easy induction on the height (see [7, Definition 1.1.9]) of $\mathcal{T}$*.*

Theorem 4. *In a sequent* s *in a* **G3s**−*proof* $\mathcal{T}$*, any pair of nested* $\Box$*'s belong to different families.*

Proof. By an induction on the height of $\mathcal{T}$*. For the inductive step, employ the fact that no* **G3s**−*rule can relate two nested* $\Box$*'s in a premise to a same one in the conclusion.*

Theorem 5. *If a* $\boxplus_j$ *occurs in the scope of a* $\boxplus_i$ *in a sequent of* $\mathcal{T}$*, then for any* $\boxplus_i$ *in any sequent of* $\mathcal{T}$*, there is a* $\boxplus_j$ *occurs in the scope of this* $\boxplus_i$*.*

Proof. Take $\boxplus_i(\cdots \boxplus_j \cdots)$ *which occurs (as a subformula, proper or not) in a sequent. Since* **G3s** *has the subformula property,* $\boxplus_i(\cdots \boxplus_j \cdots)$ *occurs in* s_r*. Suppose that there is a* $\boxplus_i$*, which has no* $\boxplus_j$ *in its scope, occurs in some sequent, then by a similar reason, this* $\boxplus_i$ *also occurs in* s_r*, without any* $\boxplus_j$ *in its scope. By Lemma 2, this cannot happen.*

[2] For the $(R\Box)$ rule presented in Definition 1, only the displayed $\Box$ in $\Box\psi$ of the conclusion is principally introduced.

2.2 On S4-LP Realization Procedure

Here we need to denote notions from the realization procedure. We refer to [1] and [2] for complete descriptions of the procedure without presenting any more here, since including a detailed instruction of it may considerably prolong this paper.

In the realization procedure, we need to apply two substitutions. The first one is to substitute all occurrences of $\Box$'s in a **G3s**−proof by terms (with provisional variables). Particularly, each $\boxplus_i$ is replaced by the sum of provisional variables of this family, i.e., $u_{i.1}+\cdots+u_{i.m_i}$. After the first substitution described above, the resulting tree is then denoted by $\mathcal{T}'=(T',R')$, while the resulting sequent, set of formulas and formula corresponding to s_i, Γ, ϕ are denoted by s_i', Γ', ϕ', respectively. $(R\Box)$ rules of $\mathcal{T}$ are temporally replaced by "Lifting Lemma Rules" in $\mathcal{T}'$. All the other rules are automatically transferred to corresponding $\mathbf{LPG}_0$−rules.

During the second substitution (in fact, a series of substitutions applied inductively), all provisional variables (denoted by u's) are replaced by (provisional-variable free) **LP** terms (denoted by t's). Roughly speaking, we apply this from leaf-side-most "Lifting Lemma rules" to the root-side-most one. In a proof tree with more than one branches, we may have variant orders to do so. For notational convenience, here we employ a function ϵ s.t. for any $i \in \{1,\cdots,m\}$ and $j \in \{1,\cdots,m_i\}$, the "Lifting Lemma Rule" corresponding to $(R\Box)_{i.j}$ is dealt as the $\epsilon(i.j)$-th one. It should be emphasized that $O_{i_1.j_1}R^+O_{i_2.j_2}$ implies $\epsilon(i_2.j_2) < \epsilon(i_1.j_1)$. Suppose that $\epsilon(i_0.j_0)=1$, then we use $\mathcal{T}^{\epsilon(i.j)}$ to denote $\mathcal{T}'(u_{i_0.j_0}/t_{i_0.j_0})\cdots(u_{i.j}/t_{i.j})$, i.e., the result of substituting all provisional variables which have been dealt till $\epsilon(i.j)$ by corresponding **LP**−terms. $s^{\epsilon(i.j)}, \Gamma^{\epsilon(i.j)}, \phi^{\epsilon(i.j)}$ have similar meanings. Particularly, we have $\mathcal{T}^0=\mathcal{T}'$.

Now we consider the subprocedure to generate, say, $t_{i.j}$. We apply Lifting Lemma (Theorem 1) on an **LP**−derivation of $I_{i.j}^{\epsilon(i.j)-1}$. This derivation is provided by i.h., and is denoted by $d_{i.j}$. During this application, we may need some (finitely many) new constants, which are then denoted by $c_{i.j.1},\cdots,c_{i.j.m_{i.j}}$. In the standard realization procedure presented in [1], the constant specification employed is injective. That is to say, given a constant, say, $c_{i.j.k}$, the corresponding formula being prefixed, denoted by $A_{i.j.k}$, is determined. Applying Lifting Lemma (Theorem 1) on $d_{i.j}$ generates a new **LP**−derivation and the desired term $t_{i.j}$. The set of all formulas introduced by AN rules in the new derivation is denoted by $\mathcal{CS}_{i.j}^{\epsilon(i.j)-1}$. Then we substitute $u_{i.j}$ by $t_{i.j}$ in the new generated derivation. After this u/t−substitution, the resulting derivation **is** an **LP**−derivation of $O_{i.j}^{\epsilon(i.j)}$, and the set of all formulas introduced by AN rules in this derivation is $\mathcal{CS}_{i.j}^{\epsilon(i.j)}$.

After the second (series of) substitution, we take notations $\mathcal{T}'', s_i'', \Gamma'', \phi''$ to denote respectively the tree, sequent, set of formulas and formula. Particularly, $\mathcal{CS}''$ denotes the set of formulas introduced by AN rules in the **LP**−derivation of the root sequent. Note that, if $O_{i_2.j_2}R^*O_{i_1.j_1}$, then $\mathcal{CS}_{i_1.j_1}^{\epsilon(i_2.j_2)} \subseteq \mathcal{CS}_{i_2.j_2}^{\epsilon(i_2.j_2)}$, since our procedure is applied leafside-to-rootside inductively. Therefore, we have $\mathcal{CS}''=\bigcup_{i\in\{1,\cdots,m\}}\bigcup_{j\in\{1,\cdots,m_i\}}\mathcal{CS}_{i.j}''$.

3 Prehistoric Phenomena

In this section, we introduce "prehistoric phenomena", while proving some related results.

Definition 2 (History). *In branch s_0 of the form $s_r R^* O_{i.j} R I_{i.j} R^* s_0$ in a* **G3s** *$-$proof $\mathcal{T}$, the path $O_{i.j}$, i.e., $s_r R^* O_{i.j}$, is called a* history *of f_i in branch s_0.*

Definition 3 (Prehistoric Relation). *For any principal positive families f_i and f_h, any branch s of the form $s_r R^* O_{i.j} R I_{i.j} R^* s$:*

(1) If $I_{i.j}$ has the form of $\boxminus\xi_1, \cdots, \boxminus\xi_k(\boxplus_h^{I_{i.j}}(\cdots)), \cdots, \boxminus\xi_n \Rightarrow \eta$, then f_h is a left prehistoric family of f_i in s. *Notation: $h \prec_L^s i$.*

(2) If $I_{i.j}$ has the form of $\boxminus\xi_1, \cdots, \boxminus\xi_n \Rightarrow \eta(\boxplus_h^{I_{i.j}}(\cdots))$, then f_h is a right prehistoric family of f_i in s. *Notation: $h \prec_R^s i$.*

(3) The relation of prehistoric family in s *is defined by: $\prec^s := \prec_L^s \cup \prec_R^s$.*

In **G3s***$-$proof $\mathcal{T}$, relations of* (left, right) prehistoric *are defined by:* $\prec_L := \bigcup\{ \prec_L^s \mid s \text{ is a leaf}\}, \prec_R := \bigcup\{ \prec_R^s \mid s \text{ is a leaf}\}, \prec := \prec_L \cup \prec_R$.

At a first sight, Definition 3 is not built on Definition 2 directly. We now present a lemma to indicate the desired connection.

Lemma 3. *The following two statements are equivalent: (1) $h \prec^s i$; (2) In branch s, there is a node s' (which is also a sequent), s.t. there is a history of f_i in s that does not include s', and there is an occurrence of $\boxplus_h^{s'}$ in s'.*

Proof. *The ($\Rightarrow$) direction is trivial, since $\boxplus_h^{I_{i.j}}$ mentioned in the definition is the $\boxplus_h$ desired. For the ($\Leftarrow$) direction, the following arguments applies. By the assumption, s has the form of $s_r R^* O_{i.j} R I_{i.j} R^* s' R^* s$ for some j. Since* **G3s** *enjoys the subformula property, we know that no matter what rules are applied from s' to $I_{i.j}$, the $\boxplus_h^{s'}$ will occur in $I_{i.j}$ as a $\boxplus_h^{I_{i.j}}$.*

We have a few remarks here. (1) Intuitively speaking, a history can be seen as a list of sequents, whose **inverse** starts from the conclusion of an ($R\Box$), and ends at the root of the proof tree. Each history in a branch breaks it into two parts, i.e, the "historic period", which is from the conclusion of the ($R\Box$) to the root of the proof tree, and the "prehistoric period", which is from the leaf of the branch to the premise of the ($R\Box$). We know by Lemma 3 that $h \prec^s i$ iff $\boxplus_h$ has an occurrence in the "prehistoric period" of f_i in s. That is the reason why the $\prec$ relation is called "prehistoric relation". (2) If a family is principally introduced into a branch for more than one times, it may have many different histories in that branch. (3) It is possible that $\prec_L^s \cap \prec_R^s \neq \varnothing$, as instanced by the following derivation, in which both $h \prec_L^s i$ and $h \prec_R^s i$ hold:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(Ax)\quad p, \boxminus p, \boxminus\neg \boxplus_i \boxplus_h p \Rightarrow p}}{(L\Box)\quad \boxminus p, \boxminus\neg \boxplus_i \boxplus_h p \Rightarrow p}}{(R\Box)\quad \boxminus p, \boxminus\neg \boxplus_i \boxplus_h p \Rightarrow \oplus_h p}}{(R\Box)\quad \boxminus p, \boxminus\neg \boxplus_i \boxplus_h p \Rightarrow \oplus_i \boxplus_h p}}{(L\neg)\quad \boxminus p, \boxminus\neg \boxplus_i \boxplus_h p, \neg \boxplus_i \boxplus_h p \Rightarrow}}{(L\Box)\quad \boxminus p, \boxminus\neg \boxplus_i \boxplus_h p \Rightarrow}$$

In Section 2, we have gained some properties about **G3s**−proofs. Now in the terminology of prehistoric phenomena, we have the following corollaries:

Corollary 1. *For any principal positive family f_i, $i \not\prec_R i$.*

Proof. Directly from Theorem 4.

Corollary 2. *If $k \prec_R j$ and $j \lhd i$, then $k \lhd i$, where $\lhd$ is one of $\prec$, $\prec_L$, $\prec_R$, $\prec^s$, $\prec^s_L$,$\prec^s_R$.*

Proof. Since $k \prec_R j$, there is a $\boxplus_k$ occurring in the scope of an $\oplus_j$. By Theorem 5, wherever $\boxplus_j$ occurs, there is a $\boxplus_k$ occurring in the scope of it.

For $\lhd = \prec^s_L$, since $j \prec^s_L i$, the branch s has a form of $s_r R^* O_{i.x} R I_{i.x} R^* s$, where $I_{i.x}$ is $\boxminus\xi_1, \cdots, \boxminus\xi_y(\boxplus_j^{I_{i.x}}(\cdots)), \cdots, \boxminus\xi_n \Rightarrow \eta$. By the observation above, we know that there is a $\boxplus_k^{I_{i.x}}$ occurring in the scope of the displayed $\boxplus_j^{I_{i.x}}$. Therefore, $k \prec^s_L i$.

The case that $\lhd = \prec^s_R$ can be shown similarly, while the case that $\lhd = \prec^s$ is an easy consequence of the previous cases. In the same way, we can gain corresponding results for $\prec_L$ and $\prec_R$, and then for $\prec$.

We are now ready to present the notion of "prehistoric loop", which indicates a special structure of principal positive families w.r.t. prehistoric relations.

Definition 4 (Prehistoric Loop). *In a* **G3s**−*proof $\mathcal{T}$, the ordered sequent of principal positive families $f_{i_1}, \cdots, f_{i_n}$ is called a* prehistoric loop *or* left prehistoric loop *respectively, if we have:*
$i_1 \prec i_2 \prec \cdots \prec i_n \prec i_1$ *or* $i_1 \prec_L i_2 \prec_L \cdots \prec_L i_n \prec_L i_1$.

In an ($R\Box$) rule, formulas residing in the left or right of $\Rightarrow$ in the premise play different roles. This property allows us to care about differences between $\prec_R$ and $\prec_L$. With Corollary 2, we know that $\prec_L$'s are the only essential steps in a prehistoric loop, as stated in the following theorem.

Theorem 6. *$\mathcal{T}$ has a prehistoric loop iff $\mathcal{T}$ has a left prehistoric loop. Moreover, for ($\Rightarrow$) direction, the left prehistoric loop claimed is not longer than the given prehistoric loop.*

Proof. The ($\Leftarrow$) direction is trivial. For the ($\Rightarrow$) direction, by assumption, $i_1 \prec i_2 \prec \cdots \prec i_n \prec i_1$. We claim that there must be a $\prec_L$ in the prehistoric loop listed above. Otherwise, $i_1 \prec_R i_2 \prec_R \cdots \prec_R i_n \prec_R i_1$. By Corollary 2, we have $i_1 \prec_R i_1$, which is forbidden by Corollary 1.

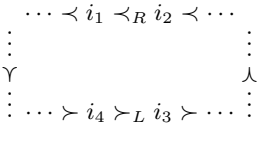


Fig. 1. A prehistoric loop with both $\prec_L$ and $\prec_R$

By the observation above, we know that there is a $\prec_L$ in the loop. If there is no $\prec_R$, then we have done since it is already a left prehistoric loop. So it is sufficient to treat the case that there are both $\prec_L$'s and $\prec_R$'s in the loop, which can be displayed roughly in Figure 1.

As being indicated by the figure, there must be an "RL-border", i.e., $\cdots \prec_R \prec_L \cdots$[3]*. W.l.o.g., we have $\cdots \prec i_1 \prec_R i_2 \prec_L i_3 \prec \cdots$. Then by Corollary 2, we have $\cdots \prec i_1 \prec_L i_3 \prec \cdots$. While having less $\prec_R$'s, it is still a prehistoric loop. Since there are only finitely many $\prec_R$'s in the original loop, we can, eventually, gain a prehistoric loop with only $\prec_L$'s, which is, by Definition 4, a left prehistoric loop.*

In the last of this section, we consider the role of prehistoric phenomena in **G3s**. All axioms and most of rules in **G3s** are common devices in $G3-$systems. The only two exceptions are $(L\Box)$ and $(R\Box)$. $(L\Box)$ is the only rule, which can relate two occurrences of $\Box$ in **one** sequent together. Roughly speaking, it is $(L\Box)$ who determines the family-wise-situation of a proof. $(R\Box)$ is the only rule, which introduces prehistoric relations between (principal positive) families. While the right prehistoric relation behaves so "explicitly" (this relation can be seen from the form of the succedent of an $(R\Box)$'s conclusion), the left prehistoric relation is not very obvious. By the notion of prehistoric relations, the behavior of families in **G3s**$-$proofs are highlighted.

4 Left Prehistoric Loop and Self-referentiality

In this section, we consider (not necessarily direct) self-referentiality with notions of prehistoric phenomena. We offer some lemmas at first, to avoid prolix proofs. Lemma 4 tells, when applying Lifting Lemma (Theorem 1), which provisional variables may be included in the corresponding constant specification. This is important, since putting a later[4] provisional variable in an earlier $\mathcal{CS}$ is the only way that can force a later constant to occur in the axiom associated to an earlier constant (by an AN rule).

Lemma 4. *Any provisional variable $u_{x.y}$, which does not occur in $I_{i.j}^{\epsilon(i.j)-1}$, does not occur in $\mathcal{CS}_{i.j}^{\epsilon(i.j)-1}$.*

Proof. (1) ***Claim:*** *$u_{x.y}$ does not occur in any sequent of $\mathcal{T}^{\epsilon(i.j)-1} \restriction I_{i.j}^{\epsilon(i.j)-1}$. Proof of the claim:* {Case 1: If $\boxplus_x$ does not occur in $I_{i.j}$, we know that $\boxplus_x$ does not occur in any sequent of $\mathcal{T} \restriction I_{i.j}$, since **G3s** enjoys the subformula property.

[3] Otherwise, since a $\prec_R$, say $i_1 \prec_R i_2$, is included, the loop would look like $\cdots i_1 \prec_R i_2 \prec_R i_3 \prec_R \cdots$(loop to the beginning), and then no $\prec_L$ could ever been included. Similarly, there is also an "LR-border". To gain a left prehistoric loop, it is sufficient (see following observations to see why) to consider one of "RL-case" and "LR-case". With Corollary 2 in hand, we take the "RL-case" in what follows.

[4] Here "earlier" and "later" are used w.r.t. the order indicated by ϵ (see Section 2).

Therefore, $u_{x.y}$ does not occur in any sequent of $\mathcal{T}^0 \restriction I^0_{i.j}$, and hence, does not occur in any sequent of $\mathcal{T}^{\epsilon(i.j)-1} \restriction I^{\epsilon(i.j)-1}_{i.j}$. Case 2: If $\boxplus_x$ occurs in $I_{i.j}$, then $u_{x.y}$ occurs in $I^0_{i.j}$. In this case, $u_{x.y}$ does not occur in $I^{\epsilon(i.j)-1}_{i.j}$ implies $\epsilon(x.y) < \epsilon(i.j)$. Therefore, $u_{x.y}$ does not occur in any sequent of $\mathcal{T}^{\epsilon(i.j)-1} \restriction I^{\epsilon(i.j)-1}_{i.j}$.}

(2) ***Claim:*** *Each sequent* $s^{\epsilon(i.j)-1}$ *of* $\mathcal{T}^{\epsilon(i.j)-1} \restriction I^{\epsilon(i.j)-1}_{i.j}$ *has a* $u_{x.y}$*-free* $\mathbf{LP}-$*derivation.*

Proof of the claim: {By an induction on $\mathcal{T}^{\epsilon(i.j)-1} \restriction I^{\epsilon(i.j)-1}_{i.j}$, which is an "$\mathbf{LPG}_0 +$ Lifting Lemma Rule" proof.

(2.1) For base cases, we take (Ax) as an example, since $(L\bot)$ can be treated similarly. Assume that s, which has the form[5] of $p, \Gamma \Rightarrow \Delta, p$, is introduced by an (Ax). Then we can take $\quad p \wedge (\wedge\Gamma), \qquad p \wedge (\wedge\Gamma) \rightarrow (\vee\Delta) \vee p, \qquad (\vee\Delta) \vee p$ to be the $\mathbf{LP}-$derivation desired[6]. Note that all provisional variables in this derivation have occurrences in s, which is $u_{x.y}$-free (by the first claim). Hence, the resulting derivation is also $u_{x.y}$-free.

(2.2) (Sketch) The proof for Boolean cases is a cumbersome routine. We take $(L \rightarrow)$ as an instance. By i.h., we have $u_{x.y}$-free derivations of the two premises. Based on these derivations, we can build a $u_{x.y}$-free derivation of the conclusion. The deduction theorem of **LP** (Lemma 1) is employed, which does not introduce any $u_{x.y}$, if we use the standard algorithm to calculate. A full proof of this case can be found in the Appendix.

(2.3) For non-Boolean $\mathbf{LPG}_0-$rules, we take $(L:)$, which corresponds to $(L\Box)$ in **G3s**, as an example. The proofs for other non-Boolean rules can be found in the Appendix. Suppose that s is obtained by an $(L:)$, i.e.,

$$\frac{\phi, t:\phi, \Gamma \Rightarrow \Delta}{t:\phi, \Gamma \Rightarrow \Delta}$$

By i.h., we have a $u_{x.y}$-free derivation, say d', of the premise. The desired $u_{x.y}$-free derivation of s is then gained by adding $\quad t:\phi, \qquad t:\phi \rightarrow \phi \quad$ to the beginning of d'.

(2.4) For the "Lifting Lemma Rule". Assume that s is obtained by applying a "Lifting Lemma Rule":

$$\frac{x_1:\xi_1, \cdots, x_n:\xi_n \Rightarrow \eta}{x_1:\xi_1, \cdots, x_n:\xi_n, \Gamma \Rightarrow \Delta, t(x_1, \cdots, x_n):\eta}$$

By i.h., we have a $u_{x.y}$-free derivation of the premise. To construct t, we apply Lifting Lemma on this derivation. Note that the resulting derivation is also

[5] Since s is a sequent in $\mathcal{T}^{\epsilon(i.j)-1} \restriction I^{\epsilon(i.j)-1}_{i.j}$, precisely speaking, we should denote it by $s^{\epsilon(i.j)-1}$, with similar superscripts added to formulas in it. However, we omit those superscripts temporarily, since we are now living in the scope of a specified tree, instead of a series of trees.

[6] Strictly speaking, we need to specify an order for conjuncts and disjuncts here. Since this issue is not essential here, we may take the one employed in [2].

$u_{x.y}$-free[7]. Then, we make some Boolean amendment (to allow the weakening) on the resulting derivation, which is similar to the situation in Boolean-rule-cases. This amendment does not introduce occurrences of $u_{x.y}$, since each formula we need to weakened in has an explicit occurrence in s, which is $u_{x.y}$-free (by the first claim).}

Thirdly, as an easy consequence of the second claim, we know that $I_{i.j}^{\epsilon(i.j)-1}$ *has a* $u_{x.y}$*-free derivation, which is constructed inductively on* $\mathcal{T}^{\epsilon(i.j)-1} \restriction I_{i.j}^{\epsilon(i.j)-1}$. *That is to say,* $d_{i.j}$ *is* $u_{x.y}$*-free.*

Lastly, we consider $\mathcal{CS}_{i.j}^{\epsilon(i.j)-1}$, *which consists of formulas of the from* $c : A^{\epsilon(i.j)-1}$. *If* $c \in \{c_{i.j.1}, \cdots, c_{i.j.m_{i.j}}\}$, *then* $A^{\epsilon(i.j)-1}$ *is an axiom employed in* $d_{i.j}$. *If* $c \notin \{c_{i.j.1}, \cdots, c_{i.j.m_{i.j}}\}$, *then* $c : A^{\epsilon(i.j)-1}$ *is introduced by an AN rule in* $d_{i.j}$. *In both cases, we have* $A^{\epsilon(i.j)-1}$ *occurs in* $d_{i.j}$. *Since* $u_{x.y}$ *does not occur in* $d_{i.j}$, $u_{x.y}$ *does not occur in* $A^{\epsilon(i.j)-1}$. *Thus,* $u_{x.y}$ *does not occur in* $\mathcal{CS}_{i.j}^{\epsilon(i.j)-1}$.

When dealing with a "Lifting Lemma Rule", we can **try to** reduce the risk of committing self-referentiality by choosing new constants. Hence, the order ϵ is essential. Having considered this, we present Lemma 5, which says, given a prehistoric-loop-free proof, we can arrange ϵ in such a way that the order respects prehistoric relations.

Lemma 5. *If a* **G3s**$-$*proof* $\mathcal{T}$ *is prehistoric-loop-free, then we can realize it in such a way that: If* $h_2 \prec h_1$, *then* $\epsilon(h_2.j_2) < \epsilon(h_1.j_1)$ *for any* $j_1 \in \{1, \cdots, m_{h_1}\}$ *and* $j_2 \in \{1, \cdots, m_{h_2}\}$.

Proof. We claim that there is a family f_{i_1} *s.t.* $h \not\prec i_1$ *for any principal positive family* f_h. *Otherwise, each family would have a prehistoric family. Since there are only finitely many principal positive families in* $\mathcal{T}$, *there would be a prehistoric loop, which contradicts the assumption.*

Similarly, we can show that there are families $f_{i_2}, f_{i_3}, \cdots$ *s.t.: if* $h \prec i_2$, *then* $f_h \in \{f_{i_1}\}$; *if* $h \prec i_3$, *then* $f_h \in \{f_{i_1}, f_{i_2}\}$; $\cdots$; *if* $h \prec i_x$, *then* $f_h \in \{f_{i_1}, \cdots, f_{i_{x-1}}\}$; $\cdots$.

Since there are only finitely many principal positive families in $\mathcal{T}$, *we will enumerate all of them as* $f_{i_1}, f_{i_2}, \cdots, f_{i_n}$ *in the way above. Thus, the desired* ϵ *is obtained by setting* $\epsilon(i_x.j) = j + \sum_{w=1}^{x-1} m_{i_w}$ *for each family* f_{i_x} *and* $j \in \{1, \cdots, m_{i_x}\}$.

Given a prehistoric-loop-free proof, we have generated an ϵ which respects prehistoric relations. Now Lemma 6 below says, with such an ϵ, the way the constants reside is also restricted.

[7] In applying Lifting Lemma, each axiom ϕ is transferred to $c{:}\phi$ for some new constant c, each premise $x{:}\phi$ is transferred to $x{:}\phi \quad x{:}\phi \rightarrow !x{:}x{:}\phi \quad !x{:}x{:}\phi$, each result of (AN) $a{:}\phi$ is transferred to $a{:}\phi \quad a{:}\phi \rightarrow !a{:}a{:}\phi \quad !a{:}a{:}\phi$, while each result of (MP) with i.h. $s{:}(\phi \rightarrow \psi)$ and $t{:}\phi$ is transferred by applying (MP) on these two, together with $s{:}(\phi \rightarrow \psi) \rightarrow (t{:}\phi \rightarrow (s \cdot t){:}\psi)$. It is easy to see that the whole algorithm does not add any new provisional variables. For further details about Lifting Lemma, we refer to [2].

Lemma 6. *Assume the proof tree is prehistoric-loop-free. Taken the ϵ generated in Lemma 5, we have:*

If $\epsilon(i_0.j_0) \geqslant \epsilon(i.j)$, then for any $k_0 \in \{1, \cdots, m_{i_0.j_0}\}$, any $k \in \{1, \cdots, m_{i.j}\}$, $c_{i_0.j_0.k_0}$ does not occur in $A''_{i.j.k}$.

Proof. *We employ the ϵ generated in Lemma 5. When dealing with $(R\Box)_{i.j}$, we need to apply the lifting procedure on $d_{i.j}$, which is an $\mathbf{LP}-$derivation of $I_{i.j}^{\epsilon(i.j)-1}$. Since $O_{i.j}RI_{i.j}$, by Lemma 5, we know that: each $\boxplus$ in $I_{i.j}$ belongs to a family, say f_w, s.t. $\epsilon(w.j_w) < \epsilon(i.j)$ for any $j_w \in \{1, \cdots, m_w\}$. That is to say, $I_{i.j}^{\epsilon(i.j)-1}$ is provisional-variable-free. Hence, by Lemma 4, $\mathcal{CS}_{i.j}^{\epsilon(i.j)-1}$ is provisional-variable-free, which implies that $\mathcal{CS}_{i.j}^{\epsilon(i.j)-1} = \mathcal{CS}''_{i.j}$ and*

$$A_{i.j.k}^{\epsilon(i.j)-1} = A''_{i.j.k} \quad \text{for any } k \in \{1, \cdots, m_{i.j}\}. \tag{2}$$

Since $\epsilon(i_0.j_0) \geqslant \epsilon(i.j)$, $c_{i_0.j_0.k_0}$ had not been introduced by the procedure when we began to apply lifting procedure on $d_{i.j}$. Therefore, $c_{i_0.j_0.k_0}$ does not occur in $d_{i.j}$, and hence, does not occur in any axioms employed in $d_{i.j}$. That is to say, for any $k \in \{1, \cdots, m_{i.j}\}$, $c_{i_0.j_0.k_0}$ does not occur in $A_{i.j.k}^{\epsilon(i.j)-1}$. By (2), we know that $c_{i_0.j_0.k_0}$ does not occur in $A''_{i.j.k}$.

With the three lemmas above in hand, now we are ready to verify the main theorem.

Theorem 7 (Necessity of Left Prehistoric Loop for Self-referentiality). *If an $\mathbf{S4}-$theorem ϕ has a left-prehistoric-loop-free $\mathbf{G3s}-$proof, then there is an $\mathbf{LP}-$formula ψ s.t. $\psi^\circ = \phi$ and $\vdash_{\mathbf{LP}(\mathcal{CS}^\circledast)} \psi$.*

Proof. *Since the $\mathbf{G3s}-$proof is left-prehistoric-loop-free, by Theorem 6, the proof is prehistoric-loop-free. Hence, we can take the ϵ generated in Lemma 5, which implies the result stated in Lemma 6.*

Assume with the hope of a contradiction that the resulting constant specification $\mathcal{CS}$ is self-referential. That is to say, we have:

$$\left\{\begin{array}{l} c_{i_1.j_1.k_1} : A''_{i_1.j_1.k_1}(c_{i_2.j_2.k_2}), \\ \qquad \cdots\cdots \\ c_{i_{n-1}.j_{n-1}.k_{n-1}} : A''_{i_{n-1}.j_{n-1}.k_{n-1}}(c_{i_n.j_n.k_n}), \\ c_{i_n.j_n.k_n} : A''_{i_n.j_n.k_n}(c_{i_1.j_1.k_1}) \end{array}\right\} \subseteq \mathcal{CS}''.$$

By Lemma 6, we have: $\epsilon(i_n.j_n) < \cdots < \epsilon(i_2.j_2) < \epsilon(i_1.j_1) < \epsilon(i_n.j_n)$, which is impossible. Hence, the resulting constant specification is non-self-referential.

Having finished the proof, we may notice that what have been done are natural. By the left-prehistoric-loop-free condition, we are given an order of families. The order is then inherited by an ϵ function, which indicates the order of lifting

procedures. Eventually, the order is echoed by the way in which the constants reside themselves in the axioms associated to them (by AN rules).

5 Conclusions and Future Research

In this paper, we define "prehistoric phenomena" in a Gentzen-style formulation (**G3s**) of modal logic **S4**. After presenting some basic results about this notion, a proof of the necessity of left prehistoric loop for self-referentiality is given. The presented work constitutes a small step in the journey of finding a criterion for self-referentiality in realization procedures, while the whole journey has its meaning in offering an **S4** (and then, intuitionistic) measure of self-referentiality introduced by terms-allowed-in-types capacity.

There are still spaces for this work to be developed. (1) The current method may be applied to other "modal-justification" pairs that also enjoy realization procedures based on $G3$ Cut-free formulations. (2) It is unclear whether left prehistoric loop is sufficient for self-referentiality. The approach in [2] may help here. (3) We have not answered the question that whether there is an **S4**−theorem, the realization of which necessarily calls for self-referentiality, but not for direct self-referentiality. (4) Despite the applications shown above, we assume that "prehistoric phenomena" have interests of their own, since they describe some family-wise structures of Gentzen-style modal proof trees.

Acknowledgements. The author is indebted to Roman Kuznets and three anonymous referees, who have offered detailed, helpful comments on earlier versions of this paper.

References

1. Artemov, S.N.: Explicit Provability and Constructive Semantics. The Bulletin of Symbolic Logic 7(1), 1–36 (2001)
2. Brezhnev, V.N., Kuznets, R.: Making Knowledge Explicit: How Hard It Is. Theoretical Computer Science 357(1-3), 23–34 (2006)
3. Fitting, M.: The Logic of Proofs, Semantically. Annals of Pure and Applied Logic 132(1), 1–25 (2005)
4. Fitting, M.: Realizations and LP. In: Artemov, S., Nerode, A. (eds.) LFCS 2007. LNCS, vol. 4514, pp. 212–223. Springer, Heidelberg (2007)
5. Kuznets, R.: Self-referentiality of justified knowledge. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) CSR 2008. LNCS, vol. 5010, pp. 228–239. Springer, Heidelberg (2008)
6. Kuznets, R.: Self-referential Justifications in Epistemic Logic. Theory of Computing Systems (online) 1 (2009)
7. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory, 2nd edn. Cambridge University Press, Cambridge (2000)

Appendix

A Detailed Proof of Lemma 4 Subcase (2.2). For Boolean cases, we take $(L\rightarrow)$ as an instance. All other Boolean rules can be dealt in similar ways. Assume that s is introduced by an $(L\rightarrow)$, i.e.,

$$\frac{\Gamma\Rightarrow\Delta,\phi \qquad \psi,\Gamma\Rightarrow\Delta}{\phi\rightarrow\psi,\Gamma\Rightarrow\Delta}$$

By i.h., we have $u_{x.y}$-free derivations d_L and d_R of the two premises. We apply the deduction theorem of **LP** (Lemma 1) to d_R, and denote the resulting derivation $(\wedge\Gamma\vdash\psi\rightarrow\vee\Delta)$ by d'_R. Specifically, if we employ the standard method to calculate d'_R, the resulting d'_R is also $u_{x.y}$-free. Now what follows is a $u_{x.y}$-free derivation of s:

$$\wedge\Gamma,\quad \overrightarrow{(d_L)}\quad (\vee\Delta)\vee\phi,\quad \phi\rightarrow\psi,\quad ((\vee\Delta)\vee\phi)\rightarrow(\phi\rightarrow\psi)\rightarrow((\vee\Delta)\vee\psi),$$

$$(\phi\rightarrow\psi)\rightarrow((\vee\Delta)\vee\psi),\quad (\vee\Delta)\vee\psi,\quad \wedge\Gamma,\quad \overrightarrow{(d'_R)}\quad \psi\rightarrow\vee\Delta,$$

$$((\vee\Delta)\vee\psi)\rightarrow(\psi\rightarrow\vee\Delta)\rightarrow\vee\Delta,\quad (\psi\rightarrow\vee\Delta)\rightarrow\vee\Delta,\quad \vee\Delta\ .$$

A Detailed Proof of Lemma 4 Subcase (2.3). There are five non-Boolean $\mathbf{LPG}_0-$rules. Among them, the subcase of $(L:)$ has been presented in the main text. We now give the proofs for the rest four.

For $(L{+}_R)$, i.e.,

$$\frac{\Gamma\Rightarrow\Delta,t:\phi}{\Gamma\Rightarrow\Delta,t+s:\phi}\ .$$

By i.h., we have a $u_{x.y}-$free derivation of the premise. The desired $u_{x.y}-$free derivation of the conclusion is gained by adding $t:\phi\rightarrow t+s:\phi\qquad (t:\phi\rightarrow t{+}s:\phi)\rightarrow((\vee\Delta)\vee t:\phi)\rightarrow((\vee\Delta)\vee t{+}s:\phi)\qquad ((\vee\Delta)\vee t:\phi)\rightarrow((\vee\Delta)\vee t{+}s:\phi)\qquad (\vee\Delta)\vee t+s:\phi\qquad$ to the end of the original proof.

The subcases of $(L{+}_L)$, $(R!)$ and $(L\cdot)$ are similar to that of $(L{+}_R)$. For these cases, we also employ a corresponding **LP**$-$axiom, and then verify truth-functionally.

Author Index

Albers, Susanne 1
Aubrun, Nathalie 12

Béal, Marie-Pierre 12
Benaissa, Nazim 25
Berlinkov, Mikhail V. 37
Binkele-Raible, Daniel 328
Bollig, Benedikt 48
Bourgeois, N. 60
Brihaye, Thomas 72
Bruyère, Véronique 72
Brzozowski, Janusz 84

Carton, Olivier 96

De Pril, Julie 72

Fernau, Henning 328
Fomin, Fedor V. 107

Gajardo, Anahí 109
Gate, James 120
Gawrychowski, Paweł 132
Giannakos, A. 60
Gimadeev, R.A. 144
Gravin, Nick 156
Guillon, Pierre 109

Heinemann, Bernhard 169
Hélouët, Loïc 48
Hromkovič, Juraj 181
Hung, Ling-Ju 195

Itsykson, Dmitry 204

Jäger, Gerold 216
Jansen, Maurice 228
Jeż, Artur 132
Jeż, Łukasz 132
Jirásková, Galina 84
Johnson, Matthew 240
Jozsa, Richard 252

Kalinina, Elena 259
Karpovich, Pavel 266
Kloks, Ton 195
Krupski, Vladimir N. 276

Lucarelli, G. 60

Martyugin, P.V. 288
Méry, Dominique 25
Milis, I. 60

Pan, Victor Y. 303
Paschos, V. Th. 60
Patel, Viresh 240
Paulusma, Daniël 240
Plisko, Valery 315
Pottié, O. 60

Qian, Guoliang 303

Raghavan, Prabhakar 327
Razborov, Alexander 340

Santha, Miklos 343
Sarma M.N., Jayalal 228
Schöning, Uwe 344
Shur, Arseny M. 350
Stewart, Iain A. 120

Trahtman, Avraham N. 362
Trunck, Théophile 240

Vereshchagin, Nikolay 371
Vyalyi, M.N. 144

Yu, Junhua 384

Zhang, Weixiong 216
Zheng, Ai-Long 303
Zou, Chenglong 84