TUCS

Mika Hirvensalo | Vesa Halava and
Igor Potapov | Jarkko Kari (Eds.)

Proceedings of the

Satellite Workshops of DLT 2007

Workshop on Probabilistic and Quantum Automata
Workshop on Reachability Problems
Workshop on Tilings and Self-Assembly

Turku Centre *for* Computer Science

TUCS General Publication
No 45, June 2007

TUCS

Proceedings of the

Satellite Workshops of DLT 2007

PART 1
Workshop on Probabilistic and Quantum Automata

PART 2
Workshop on Reachability Problems

PART 3

Workshop on Tilings and Self-Assembly

TUCS General Publication
No 45, June 2007

TUCS

Workshop on

Probabilistic and Quantum Automata

7 July 2007, Turku, Finland

Editor:

Mika Hirvensalo

Preface

The workshop on Probabilistic and Quantum Automata (PQA 2007) was held in Turku, Finland on July 7 2007 as a satellite event to the annual conference Developments in Language Theory (DLT 2007). The date is characterized by an extraordinary sequence 070707 not going to repeat before a century has elapsed.

The main topic of the workshop was as its title indicates, but other interesting topics included quantum query algorithms and connections to the first-order logic.

The basic pillar of the workshop consisted of three invited talks given by experts on quantum and probabilistic automata. In addition to that, the program committee consisting of F. Ablayev (Kazan, Russia), A. Ambainis (Waterloo, Canada), and M. Hirvensalo (Turku, Finland) accepted four papers to be presented as regular contributions. Unfortunately the third invited paper was not able to reach the print shop in time, so it cannot be published in this volume.

I want to thank the organizers of DLT 2007, in particular to Vesa Halava and Alexander Okhotin for their help with the arrangements, and Elisa Mikkola for secretarial assistance. Thanks also go to Turku Centre for Computer Science and the print shop Painosalama Oy for the production of this volume.

PQA 2007, DLT 2007, and International Visitor Program 2006–2007 was supported by The Academy of Finland, The Finnish Cultural Foundation, Finnish Academy of Science and Letters (Vilho, Yrjö and Kalle Väisälä Fund), Nokia Foundation, Turku University Foundation, and Centro Hotel.

Turku, Finland June 2007 Mika Hirvensalo

Program

10:00–10:30	Registration + COFFEE
10:30–11:30	*Invited talk:* Rūsiņš Freivalds
11:30–12:00	Marats Golovkins, Maksim Kravtsev, and Vasilijs Kravcevs: *On a Class of Languages Recognizable by Probabilistic Reversible Decide-and-Halt Automata*
12:00–12:30	Ilze Dzelme-Bērziņa: *First Order Logic and Acceptance Probability of Quantum Finite State Automata*
	LUNCH
14:00–14:30	Lelde Lāce: *Nondeterministic quantum query with minimal complexity*
14:30–15:30	*Invited talk:* Andris Ambainis
	COFFEE
16:00–17:00	*Invited talk:* Farid Ablayev
17:00–17:30	Alina Dubrovska, Taisia Mischenko-Slatenkova, and Alexander Rivosh: *Quantum Query Algorithms for Certain Problems and Generalization of Algorithm Designing Techniques*

Table of Contents

Classical Simulation Complexity of Bounded-error and Unbounded-error Quantum Branching Programs

Farid Ablayev[⋆]

Abstract. We present classical simulation techniques for measure once quantum branching programs.

For bounded-error quantum branching program of width w that computes a function with margin ϵ we present a classical deterministic branching program of the same length and width at most $(1+1/\epsilon)^{2w}$ that computes the same function.

Second technique is a classical stochastic simulation technique for bounded-error quantum branching programs. Our result is that a resulting stochastic classical branching program is of the same length and almost the same width, but we lost bounded-error acceptance.

1 Introduction

Investigations of different aspects of quantum computations in the last decade became a very intensively growing area of mathematics, computer science, and physics. A good source of information on quantum computations is Nielsen's and Chuang's book [7]. The interest in models of quantum computation has been steadily increasing since the discovery of a polynomial time algorithm for factoring by Peter Shor [12]. During the last decade different types of quantum computation models based on Turing Machines, finite automata, circuits, and branching programs have been considered. For several of these models of computations, some examples of functions were presented for which quantum models appear to be much more efficient than their classical counterparts.

Complexity of classical simulation of quantum computations for different models of computations were investigated in numerous papers [4, 8, 11].

In the paper we present two classical simulation techniques for measure once quantum branching programs.

Our first result is the following. For bounded-error quantum branching program of width w that computes a function with margin ϵ we present a classical deterministic branching program of the same length and width at most $(1+1/\epsilon)^{2w}$ that computes the same function. The construction of corresponding deterministic branching program based on the properties that

1. quantum states are unit vectors (a set of all quantum states are form bounded set for $||\cdot||_2$ norm) and

[⋆] Work done in part while visiting Max-Plank Institute for Mathematics Bonn in 2007 Email: `ablayev@ksu.ru`

2. that unitary transformation of quantum states preserves a distance.

Second technique is a classical stochastic simulation technique for bounded-error quantum branching programs. Our result is that a resulting stochastic classical branching program is of the same length and almost the same width, but we lost bounded-error acceptance. Our construction of stochastic branching program is based on:

1. Replacing complex matrices with real ones with dimension doubled and tensor product construction as a bridge between $||\cdot||_1$ norm and $||\cdot||_2$ norm (Lemma 3). This construction gives new matrices with quadratic increase in dimension.
2. A Turakainen-type construction [13] to replace arbitrary real matrices with stochastic ones with "good properties" of the original ones (Lemma 4).

2 Definitions and Results

We start with definition of branching programs according to [14] (we call it *constructive* definition). Then we give an algebraic definition of branching programs and present results of the paper.

Definition 1 *A* branching program *(BP) on the variable set $X = \{x_1, \ldots, x_n\}$ is a finite directed acyclic graph with one source node and sink nodes partitioned into two sets –* Accept *and* Reject*. Each non-sink node is labeled by a variable x_i and has two outgoing edges labeled* 0 *and* 1 *respectively. An input σ is* accepted *if and only if it induces a chain of transitions leading to a node in* Accept*, and the set of such inputs is the language accepted by the program.*

A branching program is *oblivious* if the nodes can be partitioned into levels $V_1, \ldots, V_\ell$ and a level $V_{\ell+1}$ such that the nodes in $V_{\ell+1}$ are the sink nodes, nodes in each level V_j with $j \leq \ell$ have outgoing edges only to nodes in the next level V_{j+1}, and all nodes in a given level V_j query the same bit σ_{i_j} of the input.

Definition 2 *The* size *$Size(P)$ of a branching program P is the number of its non-sink nodes. The* length *$Length(P)$ of branching program P is the maximum length of a path from the source to one of the sinks. The* width *$Width(P)$ of oblivious branching program P is the number $Width(P) = \max_j |V_j|$.*

In this paper we deal with polynomial size branching programs. Recall that arbitrary branching program can be transformed to oblivious branching program (see for example [6]) with only polynomial increasing the size. So without loss of generality we consider only oblivious branching programs in this paper.

Now we give a definition of a *linear branching program* based on oblivious model. This definition is a generalization of the definition of quantum branching program presented in [2]. Deterministic, stochastic, and quantum oblivious branching programs are particular cases of linear branching programs. Let $\mathbf{V}^k$

be a k-dimensional vector space. We use $|\mu\rangle$ and $\langle\mu|$ to denote column vectors and row vectors of $\mathbf{V}^k$, respectively, and $\langle\mu_1 \mid \mu_2\rangle$ denotes the complex inner product. We write μ when it is not important whether μ is a column or a row vector.

Definition 3 (Linear branching program) *A Linear Branching Program (LBP) P over $\mathbf{V}^k$ is defined as*

$$\mathcal{P} = \langle T, |\mu_0\rangle, \mathrm{Accept}\rangle \ ,$$

where $T = (T_1, \ldots, T_\ell)$ is a sequence (of length ℓ) of instructions. Each instruction T_j is a triple $T_j = \{i_j, M_j(0), M_j(1)\}$, where i_j determines a variable x_{i_j} tested on the step j, $M_j(0)$ and $M_j(1)$ are k-dimensional linear transformations of the vector space $\mathbf{V}^k$. Vectors $|\mu\rangle \in \mathbf{V}^k$ are called states (state vectors) of P, $|\mu_0\rangle \in \mathbf{V}^k$ is the initial state of P, and Accept $\subseteq \{1, \ldots, k\}$ *is the accepting set.*

According to the definition 2 it is natural to define the $Width(P)$ of linear BP as the dimension *of state space $\mathbf{V}^k$. Further for LBP P it is natural do define its size as $Size(P) = Width(P)Length(P)$.*

We define a computation on P with an input $\sigma = \sigma_1 \ldots \sigma_n \in \{0,1\}^n$ as follows:

1. *A computation of P starts from the initial state $|\mu_0\rangle$;*
2. *The j'th step of computation of P applies instruction T_j: program P queries a variable x_{i_j}, and applies the transition matrix $M_j(\sigma_{i_j})$ to the current state μ to obtain the state $\mu' = M_j(\sigma_{i_j})\mu$;*
3. *The final state (i.e., the state after step ℓ) is*

$$|\mu(\sigma)\rangle = \prod_{j=\ell}^{1} M_j(\sigma_{i_j})|\mu_0\rangle \ .$$

Now oblivious deterministic, stochastic, and quantum branching programs can be presented as follows:

Deterministic branching programs. A *deterministic* branching program is a linear branching program over a vector space $\mathbf{R}^k$. A state μ of such a program is a Boolean vector with exactly one 1. The transition matrices M have exactly one 1 in each column.

Stochastic branching programs. The concept of deterministic branching programs naturally generalizes to stochastic branching programs (SBP), by letting μ be a probability distribution, and by letting the M_j be *stochastic* matrices, i.e., matrices with non-negative entries where each column sums to 1.

For a deterministic and stochastic branching program P, for an input $\sigma \in \{0,1\}^n$ we define the acceptance probability of σ as follows

$$\mathrm{Pr}^P(\sigma) = Pr(\mu(\sigma)) = \sum_{i \in \mathrm{Accept}} |\langle i \mid \mu(\sigma)\rangle| = \left\|\mathit{\Pi}_{\mathrm{Accept}}\mu(\sigma)\right\|_1 \ . \qquad (1)$$

Here $|i\rangle$ is the basis vector with support on the node i (unit vector with value 1 at i and 0 elsewhere), and Π_{Accept} is a projection operator on the *accepting subspace* $\text{span}\{|i\rangle : i \in \text{Accept}\}$.

Quantum branching programs. We define a *quantum* branching program (QBP) as a linear branching program over a Hilbert space $\mathbf{C}^k$. The μ for such a program are complex state vectors with $\|\mu\|_2 = 1$, and the M_j are complex-valued unitary matrices. For a quantum branching program P, for an input $\sigma \in \{0,1\}^n$ we define the acceptance probability of σ as follows

$$\Pr^P(\sigma) = Pr(\mu(\sigma)) = \sum_{i \in \text{Accept}} |\langle i \mid \mu(\sigma)\rangle|^2 = \|\Pi_{\text{Accept}}\mu(\sigma)\|_2^2 \,, \tag{2}$$

that is, the probability that if we measure $\mu(\sigma)$, we will observe it in the accepting subspace. Note that this is a "measure-once" model analogous to the model of quantum finite automata in [9], in which the system evolves unitarily except for a single measurement at the end.

Notice that in contrast to algebraic definition of quantum and stochastic BPs one can define these models in a (so called) *constructive* form. See for example book [14] for constructive definition of SBP and the paper [11] for constructive definition of QBP.

Acceptance criteria. We say that a LBP P computes a Boolean function f *with unbounded error* if $\Pr^P(\sigma) > 1/2$ if $f(\sigma) = 1$ and $\Pr^P(\sigma) \leq 1/2$ if $f(\sigma) = 0$. *We say that P computes f with threshold* $1/2$.

We say that a LBP P computes a Boolean function f *with bounded error* if there is some $\epsilon > 0$ such that $\Pr^P(\sigma) \geq 1/2 + \epsilon$ if $f(\sigma) = 1$ and $\Pr^P(\sigma) \leq 1/2 - \epsilon$ if $f(\sigma) = 0$. *We say that P computes f with margin ϵ.*

2.1 Deterministic Classical Simulations of Stochastic and Quantum Branching Programs

Syntactic Stochastic and Quantum Programs For unbounded and bounded error stochastic and quantum branching programs we define two subsets $\mathcal{A}$ and $\mathcal{R}$ of the set $\mathcal{F}$ of sink state vectors (consistent and inconsistent) as follows. For unbounded error programs, we define

$$\mathcal{A} = \{\mu \in \mathcal{V}_{\ell+1} : \Pr(\mu) > 1/2\} \quad \text{and} \quad \mathcal{R} = \{\mu \in \mathcal{V}_{\ell+1} : \Pr(\mu) \leq 1/2\};$$

and for bounded error programs, we define

$$\mathcal{A} = \{\mu \in \mathcal{V}_{\ell+1} : \Pr(\mu) \geq 1/2 + \epsilon\} \quad \text{and} \quad \mathcal{R} = \{\mu \in \mathcal{V}_{\ell+1} : \Pr(\mu) \leq 1/2 - \epsilon\}$$

We call $\mathcal{A}$ and $\mathcal{R}$ the *accepting* and *rejecting* sets respectively.

Recall that $\mathcal{V}_{\ell+1}$ includes the final states reachable by all possible paths, both consistent and inconsistent. Then:

Definition 4 *We call a stochastic or a quantum branching program* syntactic *if its accepting and rejecting set of state vectors form a partition of the set of sink states, i.e., if $\mathcal{V}_{\ell+1} = \mathcal{A} \cup \mathcal{R}$.*

Note that without the syntactic restriction, it might happen that $\mathcal{V}_{\ell+1} \neq \mathcal{A} \cup \mathcal{R}$, and that some inconsistent final state vector $\mu \in \mathcal{V}_{\ell+1}$ has the property that $1/2 - \epsilon < \Pr(\mu) < 1/2 + \epsilon$.

Theorem 1 (Deterministic Simulation Theorem). *If a function is computed with bounded probability $1/2 + \epsilon$ by a width-w syntactic stochastic branching program, then it is also computed by a deterministic branching program of the same length, and width*

$$w' \leq \left(1 + \frac{1}{\epsilon}\right)^{w}.$$

Similarly, if a function is computed with bounded probability $1/2 + \epsilon$ by a width-w syntactic quantum branching program, it is computed by a deterministic program of the same length, and width

$$w' \leq \left(1 + \frac{1}{\epsilon}\right)^{2w}.$$

The proof of Theorem 1 is in the section Proofs.

2.2 Stochastic Classical Simulation of Quantum Branching Programs

Theorem 2 (Stochastic Simulation Theorem). *Let function f be unbounded error (bounded error) computed by QBP Q. Then there exists SBP P that unbounded error computes f of the same length $Length(P) = Length(Q)$ and width $Width(P) = 4Width^2(Q) + 3$.*

Now we define probabilistic and quantum complexity classes based on branching programs as follows.

Definition 5 *Let BPP-BP and PP-BP be the classes of functions computable with bounded error and unbounded error respectively by stochastic branching program of polynomial size;*

Let BQP-BP and PrQP-BP be the classes of functions computable with bounded error and unbounded error respectively by quantum branching program of polynomial size.

Theorem 3.
PrQP-BP $\subseteq$ PP-BP
BQP-BP $\subseteq$ PP-BP

Proof. The proof of the Theorem is the consequence of the Simulation Theorem 2. □

We present the proof of Theorem 2 in the section Proofs.

3 Proofs

3.1 Proof of Deterministic Simulation Theorem 1

We start with the idea of lower bounds proof and notations we use for formal proof.

Let us denote (w,l)-P an w width and l length BP. The idea of proofs of theorem 1 is that having syntactic stochastic (quantum) (w,l)-P that computes a function f with margin ϵ we construct a deterministic BP (w',l) - P' such that P' computes the same function f and

$$w' \leq \left(1+\frac{1}{\epsilon}\right)^{w}$$

when P is stochastic BP and

$$w' \leq \left(1+\frac{1}{\epsilon}\right)^{2w}$$

when P is quantum BP.

The construction of P' is based on the following properties. We will view on computation by oblivious (stochastic and quantum) BP (w,l)-P as an l step linear process of transformation of vectors α of a current internal description (ID) of P. In stochastic case $\alpha=\mu$, where $\mu=(p_1,\ldots,p_w)$ is a current probability distribution of a states of P, and in quantum case $\alpha=|\psi\rangle$, where $|\psi\rangle=(z_1,\ldots,z_w)^T$ is a current distribution of amplitudes of states of P.

We represent this l step linear process as an $(l+1)$-leveled deterministic branching program DP, as follows: each node of DP labeled by vector α corresponds to an ID of P, and each level $i\in\{0,\ldots,l\}$ represents a step of the computation. The level 0 contains initial node labeled α_0 — the initial distribution of P. From each node α on the level i, $i\in\{0,\ldots,l-1\}$, out goes two edges labeled $x_{j_i}=0$ and $x_{j_i}=1$ where x_{j_i} is a variable tested in the computational step i. Edge $x_{j_i}=1$, directed from parent α on a level i to child α' on the level $i+1$ iff P being on the step i of computation in ID α tests $x_{j_i}=1$ and transforms its ID on the step $i+1$ to α'.

Denote *Accept* (*Reject*) a set of all accepting (rejecting) sink nodes of P. Sink nodes of DP on the level l are labeled in addition by 1 ("accept") and 0 ("reject") as follows: in stochastic case sink node $\alpha=(p_1,\ldots,p_w)$ labeled 1 if $Pr(\alpha)=\sum_{j\in Accept}p_j\geq 1/2+\epsilon$, and labeled 0 if $Pr(\alpha)=\sum_{j\in Accept}p_j\leq 1/2-\epsilon$; in quantum case node $\alpha=(z_1,\ldots,z_w)^T$ labeled 1 if $Pr(\alpha)=\sum_{j\in Accept}|z_j|^2\geq 1/2+\epsilon$, and labeled 0 if $Pr(\alpha)=\sum_{j\in Accept}|z_j|^2\leq 1/2-\epsilon$. Denote $\mathcal{A}$ ($\mathcal{R}$) a set of all sink nodes of DP labeled 1 (0).

From the above we have the following property.

Property 1 *Deterministic branching program DP computes the same Boolean function f as P.*

In the next section we use metric point of view to DP for constructing deterministic BP (w',l)-P' that computes the same function as DP and hence the same function as P.

Metric Properties of DP. Denote Ψ_i, $i \in \{0,\dots,l\}$, a set all possible IDs of P on step i of computation. Let $\Psi = \cup_{i=0}^{l}\Psi_i$. For stochastic P we define metric on the space Ψ based on norm $||\cdot||_1$. That is, for $\alpha = (p_1,\dots,p_w)$ and $\alpha' = (p'_1,\dots,p'_w)$ $\rho(\alpha,\alpha') = ||\alpha-\alpha'||_1 = \sum_{i=1}^{w}|p_i - p'_i|$. For quantum P we define metric on the space Ψ based on norm $||\cdot||_2$: for $\alpha = (z_1,\dots,z_w)$ and $\alpha' = (z'_1,\dots,z'_w)$ $\rho(\alpha,\alpha') = ||\alpha-\alpha||_2$. We will also use notation $||\cdot||$ for norm $||\cdot||_2$.

Recall known notions of metric spaces we need in the proof (see for example [5]). Denote $\mathcal{H}_w$ an w-dimensional vector space (real valued or complex valued) with metric ρ. Points μ,μ' from $\mathcal{H}_w$ are connected through θ-chain if there exists a finite set of points $\mu_1,\mu_2,\dots,\mu_m$ from $\mathcal{H}_w$ such that $\mu_1 = \mu, \mu_m = \mu'$ and $\rho(\mu_i,\mu_{i+1}) < \theta$ for $i \in \{1,\dots,m-1\}$. For $\mathcal{H}_w$ its subset C is called θ-component if arbitrary two points $\mu,\mu' \in \mathcal{C}$ are connected through θ-chain. It is known [5] that if $\mathcal{D}$ is a finite diameter subset of a subspace of $\mathcal{H}_w$ (diameter of $\mathcal{D}$ is defined as $\sup_{\mu,\mu'\in\mathcal{D}}\{\rho(\mu,\mu')\}$ then for $\theta > 0$ $\mathcal{D}$ is partitioned to a finite number t of its θ-components.

Lemma 1 *Let f be a Boolean function $(1/2+\epsilon)$-computed by P. Let DP be a corresponding deterministic BP for P. Let $\theta > 0$ and let for sink nodes of DP the following holds: for arbitrary $\alpha \in \mathcal{A}$ and $\alpha' \in \mathcal{R}$ it is holds that $\rho(\alpha,\alpha') \geq \theta$. Then, there exists a deterministic BP (w',l)-P which computes f and*

$$w' \leq \left(1+\frac{2}{\theta}\right)^{w}$$

when P is stochastic BP, and

$$w' \leq \left(1+\frac{2}{\theta}\right)^{2w}$$

when P is quantum BP.

Proof. Consider $\mathcal{D}$ a sphere of radius 1 with center in $(0,\dots,0)$. We clearly have that $\Psi \subseteq \mathcal{D}$. From the condition of the lemma it follows that subsets $\mathcal{A}$ and $\mathcal{R}$ of Ψ_l is a union of some θ-components of Ψ_l. Next. Oblivious property of BP P provides the following: on the each level i, $i \in \{0,\dots,l-1\}$, of DP for all nodes $\alpha \in \Psi_i$ it is tested the same variable x_{j_i} and applied the same linear transition $M_{x_{j_i}}(1)$ if $x_{j_i} = 1$ ($M_{x_{j_i}}(0)$ if $x_{j_i} = 0$) where $M_{x_{j_i}}(a)$ is stochastic matrix when P is stochastic BP and $M_{x_{j_i}}(a)$ is unitary when P is quantum BP.

It is known (and it can be easily verified) that transformations determined by stochastic matrix M does not increase the distance. That is, if $\alpha' = M\alpha$ and $\beta' = M\beta$ then $\rho(\alpha',\beta') \leq \rho(\alpha,\beta)$. Unitary matrix M preserves the distance. That is, it is holds that $||\alpha'-\beta'|| = ||\alpha-\beta||$.

Denote $\mathcal{C}_i$ the set of all θ-components of Ψ_i. For $C \in \mathcal{C}_i$ and matrix M we denote $MC = \{\alpha' : \alpha = M\alpha, \alpha \in C\}$. From the property of non increasing distance linear transformations (stochastic or uniform) it is holds that for $C \in \mathcal{C}_i$ and for $a \in \{0,1\}$ there exists $C' \in \mathcal{C}_{i+1}$ such that $MC \subseteq C'$ for the stochastic P and $MC = C'$ for the quantum P.

Now we describe deterministic BP P' that computes f as follows: P' is an l-leveled oblivious BP. On the level i tested variable x_{j_i} (as in BP) and all nodes are labeled by θ-components $C \in \mathcal{C}_i$. From the node $C \in \mathcal{C}_i$ edge labeled $x_{j_i} = a$ goes to to node $C' \in \mathcal{C}_{i+1}$ iff $M_{x_{j_i}}(a)C \subseteq C'$.

From the above it follows that P' computes the same function as DP

The width $w(P')$ of P' is $w' = \max\{t_0, \dots, t_l\}$ where t_i is the number θ-components of Ψ_i. Let $w' = t_i$. We estimate the number t of θ-components (number of nodes of B) of Ψ_j as follows.

For each θ-component $C \in \mathcal{C}_i$ select one point $\alpha \in C$. If we draw a sphere of the radius $\theta/2$ with the center $\alpha \in C$ then all such spheres do not intersect pairwise. All these w' spheres are in large sphere of radius $1 + \theta/2$ which has center $(0, 0, \dots, 0)$. The volume of a sphere of a radius r in $\mathcal{H}_w$ is cr^w when $\mathcal{H}_w$ is real space and is cr^{2w} when $\mathcal{H}_w$ is complex space (in the complex space $\mathcal{H}_w$ each complex point is a 2-dimensional point). Constant c depends on the metric of $\mathcal{H}_w$. Now for stochastic and quantum case we have respectively that

$$w' \le \frac{c\,(1+\theta/2)^w}{c\,(\theta/2)^w} = \left(1 + \frac{2}{\theta}\right)^w, \qquad w' \le \frac{c\,(1+\theta/2)^{2w}}{c\,(\theta/2)^{2w}} = \left(1 + \frac{2}{\theta}\right)^{2w},$$

□

Below we present technical lemma that estimates parameter θ of the lemma 1 depending on margin ϵ of computation.

Lemma 2 *Let f be a Boolean function $(1/2+\epsilon)$-computed by (w,l)-P. Let DP be a corresponding deterministic BP for P.*

Then for arbitrary $\alpha \in \mathcal{A}$ and $\alpha' \in \mathcal{R}$ for the case of stochastic P it is holds that:

$$||\alpha - \alpha'||_1 \ge \theta = 2\epsilon,$$

for the case of quantum P it is holds that:

$$||\alpha - \alpha'|| \ge \theta = 2\epsilon.$$

Proof. Consider the case of stochastic P. $\alpha = (p_1 \dots, p_w)^T$, $\alpha' = (p'_1 \dots, p'_w)^T$,

$$||\alpha - \alpha'||_1 = \sum_{i=1}^{w} |p_i - p'_i| \ge \sum_{i \in Accept} |p_i - p'_i| \ge |\sum_{i \in Accept} p_i - \sum_{i \in Accept} p'_i|$$

From the condition of the lemma we have that $\sum_{i \in Accept} p_i \ge 1/2 + \epsilon$ and $\sum_{i \in Accept} p'_i \le 1/2 - \epsilon$. From this we have that

$$||\alpha - \alpha'||_1 \ge 1/2 + \epsilon - (1/2 - \epsilon) = 2\epsilon.$$

Consider the case of quantum P. $\alpha = (z_1, \dots, z_d)^T$ and $\alpha' = (z'_1, \dots z'_d)^T$. From the condition of the lemma it is holds that

$$2\epsilon \le \sum_{i \in Accept} (|z_i|^2 - |z'_i|^2) = \sum_{i \in Accept} (|z_i| - |z'_i|)(|z_i| + |z'_i|)$$
$$\le \sum_{i \in Accept} (|z_i - z'_i|)(|z_i| + |z'_i|)$$

and similarly

$$2\epsilon \le \sum_{i \in Reject} (|z'_i|^2 - |z_i|^2) = \sum_{i \in Reject} (|z'_i| - |z_i|)(|z_i| + |z'_i|)$$
$$\le \sum_{i \in Reject} (|z_i - z'_i|)(|z_i| + |z'_i|)$$

or

$$4\epsilon \le \sum_{i=1}^{w} (|z_i - z'_i|)(|z_i| + |z'_i|).$$

From the above using known inequality $\sum_{i=1}^d a_i b_i \le \sqrt{\sum_{i=1}^d a_i^2}\sqrt{\sum_{i=1}^d b_i^2}$ and triangle inequality for the norm we get that

$$4\epsilon \le ||\alpha - \alpha'||(||\,|\alpha| + |\alpha'|\,|| \le ||\alpha - \alpha'||(||\alpha|| + ||\alpha'|| = 2||\alpha - \alpha'||$$

□

Now the lower bounds of the theorem 1 follows immediately from lemmas 1 and 2. This completes the proof of theorem 1 □

3.2 Proof of Stochastic Simulation Theorem 2

We call LBP P a program of Type I, if it uses metric (1) when defining the acceptance probability, and a program of Type II, if metric (2) is used.

The simulation technique we use in the proof explores the same idea as in [3]. The nonuniformity property of the model and oblivious property makes proof clearer. The proof is based on three lemmas we present below.

Lemma 3 *Let function f be computed by QBP Q. Then there exists LBP P of Type I that computes f with $Width(P) = 4Width(Q)^2$ and $Length(P) = Length(Q)$ such that $Pr^Q(\sigma) = Pr^P(\sigma)$ for each $\sigma \in \{0,1\}^n$.*

Proof sketch. First using the known real-valued simulation of complex-valued matrix multiplication from QBP Q over $\mathbf{C}^k$ we construct LBP P' of Type II over $\mathbf{R}^{2k}$ with $Width(P') = 2Width(Q)$ and $Length(P') = Length(Q)$ such that $Pr^{P'}(\sigma) = Pr^Q(\sigma)$ for each $\sigma \in \{0,1\}^n$ (see [2] for more details).

Next we construct LBP P of Type I from LBP P' of Type II with $Width(P) = Width(P')^2$ and $Length(P) = Length(P')$ such that $Pr^{P'}(\sigma) = Pr^{P}(\sigma)$ for each $\sigma \in \{0,1\}^n$. Here we use relation among LBP of Type I and Type II (among "linear" and "non linear" extracting a result of computation) used in [9, 3]. For completeness of presentation we display it here for branching programs.

Let $d = 2k$. Let LBP $P' = \langle T, |\mu_0\rangle, \text{Accept}\rangle$ where $T = (\{i_j, M_j(0), M_j(1)\})_{j=1}^{\ell}$. We construct $P = \langle T', |\tau_0\rangle, \text{Accept}'\rangle$ as follows. The initial state $|\tau_0\rangle = |\mu_0 \otimes \mu_0\rangle$ — is d^2-dimension vector, $T' = (\{i_j, W_j(0), W_j(1)\})_{j=1}^{\ell}$ where $W_j(\sigma) = M_j(\sigma) \otimes M_j(\sigma)$ is $d^2 \times d^2$ matrix. Accepting set $\text{Accept}' \subseteq \{1, \ldots, d^2\}$ of states is defined according to *Accept* $\subseteq \{1, \ldots, d\}$ as follows $\text{Accept}' = \{j : j = (i-1)d + i,\ i \in \text{Accept}\}$.

Using the fact that for real valued vectors c, b it holds that $\langle c|b\rangle^2 = \langle c \otimes c | b \otimes b\rangle$ we have that $\prod_{j=\ell}^{1} W_j(\sigma_{i_j}) = \prod_{j=\ell}^{1} (M_j(\sigma_{i_j}) \otimes M_j(\sigma_{i_j})) = \prod_{j=\ell}^{1} M_j(\sigma_{i_j}) \otimes \prod_{j=\ell}^{1} M_j(\sigma_{i_j})$.

$$Pr^{P}(\sigma) = \sum_{i \in F'} \langle i| \prod_{j=\ell}^{1} W_j(\sigma_{i_j}) |\tau_0\rangle = \sum_{i \in F} \langle i \otimes i| \prod_{j=\ell}^{1} (M_j(\sigma_{i_j}) \otimes M_j(\sigma_{i_j})) |\mu_0 \otimes \mu_0\rangle$$
$$= \sum_{i \in F} \langle i| \prod_{j=\ell}^{1} M_j(\sigma_{i_j}) |\mu_0\rangle^2 = Pr^{P'}(\sigma).$$

From the construction of LBP P we have that $Width(P) = 4Width(Q)^2$ and $Length(P) = Length(Q)$. $\square$

Lemma 4 *Let function f be computed by LBP P of Type I. Then there exists SBP P' that computes f with $Width(P') = Width(P) + 2$, $Length(P') = Length(P)$ such that for each $\sigma = \{0,1\}^n$ it is true that*

$$Pr^{P'}(\sigma) = c^{\ell} Pr^{P}(\sigma) + 1/(d+2)$$

where $\ell = Length(P)$, $d = Width(P)$, constant $c \in (0,1]$ depends on program P.

Proof sketch. Let $P = \langle T, |\mu_0\rangle, \text{Accept}\rangle$, where $T = (T_1, \ldots, T_\ell)$ and $T_j = \{i_j, M_j(0), M_j(1)\}$. Without loss of generality we suppose that a set Accept consists only of one node. One can easily construct LBP with unique accepting node from LBP with several accepting nodes (without increasing width and length) using standard technique from Linear Automata Theory (see for example [10, 13]).

Let $d = Width(P)$. We construct SBP $P' = \langle T', |\mu_0'\rangle, \text{Accept}'\rangle$ as follows. For each instruction T_j of program P we define instruction $T_j' = \{i_j, W_j(0), W_j(1)\}$ of P' as follows.

First for each $d \times d$ matrix M of instruction T_j we define $(d+2) \times (d+2)$ matrix

$$A = \left(\begin{array}{c|c|c} 0 & 0\dots 0 & 0 \\ \hline b & M & \vdots \\ \hline \beta & q & 0 \end{array}\right),$$

such that sum of elements of each row and each column of A is zero (we are free to select elements of column b, row q and number β).

Matrix A has the property: sum of elements of each row and each column of A is zero. It is easy to verify that product of such matrices also would be a matrix of the such type.

Now let R be stochastic $(d+2) \times (d+2)$ matrix who's (i, j)-entry is $1/(d+2)$. Select positive constant $c \leq 1$ such that matrix W, defined as

$$W = cA + R$$

is stochastic matrix. Further by induction on ℓ we have that the product of matrices of type W is also stochastic matrix of the same structure. Now for an input $\sigma = \sigma_1 \dots \sigma_n$ we have that

$$W(\sigma) = \prod_{j=\ell}^{1} W_j(\sigma_{i_j}) = c^\ell \prod_{j=\ell}^{1} A_j(\sigma_{i_j}) + R.$$

By selecting suitable initial probabilities distribution $|\mu'_0\rangle$ and accepting nodes we can pick up from $W(\sigma)$ entry we need (entry that gives σ accepting probability). From the construction of SBP P' we have that $Width(P') = Width(P)+2$, $Length(P') = Length(P)$, $Pr^{P'}(\sigma) = c^\ell Pr^P(\sigma)+1/(d+2)$ for each $\sigma = \{0, 1\}^n$. □

Lemma 4 says that having Type I LBP P that process its input σ with threshold $1/2$ one can construct SBP P' that process σ with threshold $\lambda = c^\ell 1/2 + 1/(d + 2)$, where $\ell = Length(P)$ and $d = Width(P)$.

Lemma 5 *Let SBP P computes f with threshold $\lambda \in [0, 1)$. Then for arbitrary $\lambda' \in (\lambda, 1)$ there exists SBP P that computes f with threshold λ' such that $Width(P') = Width(P) + 1$, $Length(P') = Length(P)$.*

Proof: The proof uses standard technique from Probabilistic Automata Theory (see for example the book [10]) and is omitted. □

Lemmas 3,4,5 prove the statement of Theorem 2.

References

1. F. Ablayev, C. Moore, and C. Pollett, Quantum and Stochastic Branching Programs of Bounded Width, *in Proceedings of the ICALP'2002, Lecture Notes in Computer Science, Springer-Verlag*, 343-354. See also *Electronic Colloquium on Computational Complexity*, TR02-013, 2002,
available at `http://www.eccc.uni-trier.de/eccc/`

2. F. Ablayev, A. Gainutdinova, and M. Karpinski. On computational Power of quantum branching programs. *Proc. FCT 2001*, Lecture Notes in Computer Science 2138: 59–70, 2001.
3. F.Ablayev and A.Gainutdinova. Classical simulation complexity of quantum machines. *Proc. of the FCT 2003, Lect. Notes in Comp. Sci., Springer,* 2003, v 2751, 296-302.
4. L. Adleman, J. Demarrais, M. Huang. Quantum computability, SIAM J. on Computing. 26(5), 1997, 1524–1540.
5. P. Alexandrov. *Introduction to set theory and general topology.* Berlin, 1984.
6. D.A. Barrington. Bounded-width polynomial branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences* 38(1): 150–164, 1989.
7. M.A. Nielson and I.L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press. 2000.
8. L. Fortnow. One complexity theorist's view of quantum computing. *Theoretical Computer Science*, 2003, 292(3), 597–610.
9. C.Moore, J.Crutchfield. Quantum Automata and Quantum Grammars. *Theoretical Computer Science,* 2000, 237, 275–306.
10. A. Paz. Introduction to Probabilistic Automata. Academic-Press, 1971.
11. M. Sauerhoff and D. Sieling. Quantum branching programs and space-bounded nonuniform quantum complexity. *ph/0403164, March 2004.*
12. P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing* 26(5): 1484–1509, 1997.
13. P. Turakainen. Generalized automata and stochastic languages, *Proc. of AMS 21*, 1969, 303–309.
14. I. Wegener. *Branching Programs and Binary Decision Diagrams.* SIAM Monographs on Discrete Mathematics and Applications. 2000.

Permutation Groups and the Strength of Quantum Finite Automata with Mixed States [⋆]

Rūsiņš Freivalds, Māris Ozols, Laura Mančinska

Institute of Mathematics and Computer Science, University of Latvia,
Raiņa bulvāris 29, Rīga, Latvia

Abstract. It was proved earlier by A. Ambainis and R. Freivalds that the quantum finite automata with pure states can have exponentially smaller number of states than the deterministic finite automata recognizing the same language. There is a never published "folk theorem" claiming that the quantum finite automata with mixed states are no more than super-exponentially concise than the deterministic finite automata. It is not known whether the super-exponential advantage of the quantum automata with mixed states is really achievable.
We show how this problem can be reduced to a certain problem about permutation groups, namely: (1) if there is a fixed constant c and an infinite sequence of distinct integers n such that for each n there is a group G_n of permutations of the set $\{1, 2, \ldots, n\}$ such that $|G_n| = e^{\Omega(n \log n)}$ and the pairwise Hamming distance of permutations is at least $c \cdot n$, then (2) there is an infinite sequence of languages L_n such that for each language there is a quantum finite automata with mixed states that recognizes the language L_n and has $O(n)$ states, while any deterministic finite automaton recognizing L_n must have at least $e^{\Omega(n \log n)}$ states.
We do not know whether (1) is true, but we provide a list of several known results on permutation groups, that possibly could be used to prove (1). However, note that if (1) turns out to be false, it does not imply, that (2) is also false.

1 Introduction

A. Ambainis and R. Freivalds proved in [1] that for recognition of some languages the quantum finite automata can have smaller number of states than the deterministic ones, and this difference can even be exponential. The proof contained a slight non-constructiveness, and the exponent was not shown explicitly. For probabilistic finite automata exponentiality of such a distinction was not yet proved. The best (smaller) gap was proved by Ambainis [2]. The languages recognized by automata in [1] were presented explicitly but the exponent was not. In a very recent paper by R.Freivalds [3] the non-constructiveness is modified, and an explicit (and seemingly much better) exponent is obtained at the expense of having only non-constructive description of the languages used. Moreover, the best estimate in this paper was proved under the assumption of

[⋆] This research is supported by Grant No.05.1528 from the Latvian Council of Science.

the well-known Artin's Conjecture (1927) in Number Theory. [3] contains also a theorem that does not depend on any open conjectures but the estimate is worse, and the description of the languages used is even less constructive. This seems to be the first result in finite automata depending on open conjectures in Number Theory.

The following two theorems are proved in [3]:

Theorem 1. *Assume Artin's Conjecture. There exists an infinite sequence of regular languages $L_1, L_2, L_3, \dots$ in a 2-letter alphabet and an infinite sequence of positive integers $z_1, z_2, z_3, \dots$ such that for arbitrary j:*

(1) there is a probabilistic reversible automaton with z_j states that recognizes the language L_j with the probability $\frac{19}{36}$,
(2) any deterministic finite automaton recognizing L_j has at least $(2^{\frac{1}{4}})^{z_j} = (1.189207115\dots)^{z_j}$ states.

Theorem 2. *There exists an infinite sequence of regular languages $L_1, L_2, L_3, \dots$ in a 2-letter alphabet and an infinite sequence of positive integers $z_1, z_2, z_3, \dots$ such that for arbitrary j:*

(1) there is a probabilistic reversible automaton with z_j states that recognizes the language L_j with the probability $\frac{68}{135}$,
(2) any deterministic finite automaton recognizing L_j has at least $(7^{\frac{1}{14}})^{z_j} = (1.149116725\dots)^{z_j}$ states.

The two theorems above are formulated in [3] as assertions about reversible probabilistic automata. For probabilistic automata (reversible or not) it was unknown before the paper [3] whether the gap between the size of probabilistic and deterministic automata can be exponential. It is easy to re-write the proofs in order to prove counterparts of Theorems 1 and 2 for quantum finite automata with pure states. The aim of this paper is to propose a way how to prove a counterpart of these theorems for quantum finite automata with mixed states.

2 Quantum automata with mixed states

Quantum algorithms with mixed states were first considered by D. Aharonov, A. Kitaev, N. Nisan [4]. More detailed description of quantum finite automata with mixed states can be found in A. Ambainis, M. Beaudry, M. Golovkins, A. Ķikusts, M. Mercer, D. Thérien [5]. Since we are interested only in the most simple and the most restricted version of these automata, we consider only so-called Latvian QFA in this paper. These are the quantum finite automata that can be implemented using the Nuclear Magnetic Resonance (NMR) technology. All the other types of quantum finite automata with mixed states are less restrictive.

The automaton is defined by the initial density matrix ρ_0. Every symbol a_i in the input alphabet is associated with a unitary matrix U_i. When the automaton reads the symbol s_i, the current density matrix ρ is transformed into $U_i \rho U_i^\dagger$. When the reading of the input word is finished and the end-marker "\$" is read, the current density matrix ρ is transformed into $U_\$ \rho U_\$^\dagger$ and separate measurements of all states are performed. After that the probabilities of all the accepting states are totaled, and the probabilities of all the rejecting states are totaled.

It is easy to see that quantum finite automata with pure states described by C. Moore and J. Crutchfield [23] but not the quantum finite automata with pure states described by A. Kondacs and J. Watrous [22] can be simulated by Latvian QFA with no increase in the number of states.

3 From quantum automata to permutations

In this section we show how the problem of proving that quantum finite automata with mixed states have a super-exponential advantage over deterministic automata can be reduced to a certain problem about permutation groups.

Definition 1. *The* Hamming distance *or simply* distance $d(r,s)$ *between two n-permutations r and s on the set* S *is the number of elements $x \in \mathrm{S}$ such that $r(x) \neq s(x)$. The* similarity $e(r,s)$ *is the number of $x \in \mathrm{S}$ such that $r(x) = s(x)$. Note that $d(r,s) + e(r,s) = |\mathrm{S}| = n$.*

Theorem 3. *The assertion (1) implies the assertion (2), where:*

(1) there is a fixed constant c and an infinite sequence of distinct integers n such that for each n there is a group G_n of permutations of the set $\{1, 2, \dots, n\}$, the group has $e^{\Omega(n \log n)}$ elements and k generating elements, and the pairwise Hamming distance of permutations is at least $c \cdot n$,

(2) there is an infinite sequence of distinct integers n such that for each n there is a language L_n in a k-letter alphabet that can be recognized with probability $\frac{c}{2}$ by a quantum finite automata with mixed states that has $2n$ states, while any deterministic finite automaton recognizing L_n must have at least $e^{\Omega(n \log n)}$ states.

Proof. For each permutation group G_n we define the language L_n as follows:

The letters of L_n are the k generators of the group G_n and it consists of words $s_1 s_2 s_3 \dots s_m$ such that the product $s_1 \circ s_2 \circ s_3 \circ \dots \circ s_m$ differs from the identity permutation.

We will construct a quantum automaton with mixed states. It has $2n$ states and the initial density matrix ρ_0 is a diagonal block-matrix that consists of n blocks $\widetilde{\rho}_0$:

$$\widetilde{\rho}_0 = \frac{1}{2n} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}. \tag{1}$$

For example, in the case $n = 4$ the density matrix ρ_0 is given in (3).

For each of k generators $g_i \in G_n$ we will construct the corresponding unitary matrix U_i as follows – it is a $2n \times 2n$ permutation matrix, that permutes the elements in the even positions according to permutation g_i, but leaves the odd positions unpermuted.

For example, $g = 3241$ can be expressed as the following permutation matrix that acts on a column vector:

$$g = \begin{pmatrix} 0\,0\,1\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \\ 1\,0\,0\,0 \end{pmatrix} . \tag{2}$$

The initial density matrix ρ_0 for $n = 4$ and the unitary matrix U that corresponds to the permutation matrix (2) of premutation g are as follows:

$$\rho_0 = \frac{1}{8}\begin{pmatrix} 1\,1\,0\,0\,0\,0\,0\,0 \\ 1\,1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,1\,0\,0\,0\,0 \\ 0\,0\,1\,1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,1\,0\,0 \\ 0\,0\,0\,0\,1\,1\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1\,1 \\ 0\,0\,0\,0\,0\,0\,1\,1 \end{pmatrix} , \quad U = \begin{pmatrix} 1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,0\,1\,0 \\ 0\,1\,0\,0\,0\,0\,0\,0 \end{pmatrix} . \tag{3}$$

The unitary matrix $U_\$$ for the end-marker is also a diagonal block-matrix. It consists of n blocks that are the *Hadamard matrices*

$$\widetilde{H} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} . \tag{4}$$

Notice how the Hadamard matrix $\widetilde{H}$ acts on two specific 2×2 density matrices:

$$\text{if } \rho = \frac{1}{2n}\begin{pmatrix} 1\,1 \\ 1\,1 \end{pmatrix} , \text{ then } \widetilde{H}\rho\widetilde{H}^\dagger = \frac{1}{2n}\begin{pmatrix} 2\,0 \\ 0\,0 \end{pmatrix} , \tag{5}$$

$$\text{if } \rho = \frac{1}{2n}\begin{pmatrix} 1\,0 \\ 0\,1 \end{pmatrix} , \text{ then } \widetilde{H}\rho\widetilde{H}^\dagger = \frac{1}{2n}\begin{pmatrix} 1\,0 \\ 0\,1 \end{pmatrix} . \tag{6}$$

For example, when the letter g is read, the unitary matrix U is applied to the density matrix ρ_0 (both are given in equation (3)) and the density matrix $\rho_1 = U\rho_0 U^\dagger$ is obtained. When the end-marker "\$" is read, the density matrix becomes $\rho_\$ = U_\$\rho_1 U_\$^\dagger$. Matrices ρ_1 and $\rho_\$$ are as follows:

$$\rho_1 = \frac{1}{8}\begin{pmatrix} 1\,0\,0\,0\,0\,0\,0\,1 \\ 0\,1\,0\,0\,1\,0\,0\,0 \\ 0\,0\,1\,1\,0\,0\,0\,0 \\ 0\,0\,1\,1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,1\,1\,0 \\ 0\,0\,0\,0\,0\,1\,1\,0 \\ 1\,0\,0\,0\,0\,0\,0\,1 \end{pmatrix} , \quad \rho_\$ = \frac{1}{8}\begin{pmatrix} 1 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 1 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 1 & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 1 & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 1 \end{pmatrix} . \tag{7}$$

Finally, we declare the states in the even positions to be accepting, but the states in the odd positions to be rejecting. Therefore one must sum up the diagonal entries that are in the even positions of the final density matrix to find the probability that a given word is accepted.

In our example the final density matrix $\rho_\$$ is given in (7). It corresponds to the input word "$g\$$", which is accepted with probability $\frac{1}{8}(1+0+1+1) = \frac{3}{8}$ and rejected with probability $\frac{1}{8}(1+2+1+1) = \frac{5}{8}$. Note that the accepting and rejecting probabilities sum up to 1.

It is easy to see, that the words that do not belong to the language L_n are rejected with certainty, because the matrix $U_\$\rho_0 U_\$^\dagger$ has all zeros in the even positions on the main diagonal. However, the words that belong to L_n are accepted with the probability at least $\frac{d}{2n} = \frac{cn}{2n} = \frac{c}{2}$, because all permutations are at least at the distance d from the identity permutation.

It is also easy to see that any deterministic automaton that recognizes the language L_n must have at least $N = |G_n|$ states, where $|G_n|$ is the size of the permutation group G_n. If the number of states is less than N, then there are two distinct words u and v such that the deterministic automaton ends up in the same state no matter which one of the two words it reads. Since G_n is a group, for each word we can find an inverse, that returns the automaton in the initial state (the only rejecting state). Since u and v are different, they have different inverses and $u \circ u^{-1}$ is the identity permutation and must be rejected, but $v \circ u^{-1}$ is not the identity permutation and must be accepted – a contradiction. □

4 Sharply transitive permutation groups

We are interested in permutation groups such that distinct permutations have large Hamming distance (see Sect. 3 for the definition of the *Hamming distance* $d(r,s)$ and the *similarity* $e(r,s)$ of two permutations r and s). It turns out that the notion of Hamming distance is related to the *multiple transitivity* of groups.

Definition 2. *A group G of permutations on the set* S *is called k-*transitive *if for every two k-tuples $(x_1, x_2, \ldots, x_k)$ and $(y_1, y_2, \ldots, y_k)$ of distinct elements of* S*, there is a permutation $p \in G$ such that $p(x_i) = y_i$ for all $i \in \{1, 2, \ldots, k\}$. If there is exactly one such permutation p, then G is called* sharply k-transitive. *Note that a sharply k-transitive group is also sharply $(k-1)$-transitive.*

It seems that it has been noted only recently (see, e.g., [10]) that the sharp k-transitivity imposes a restriction on the Hamming distance. This is given by the following trivial lemma:

Lemma 1. *If G is a sharply k-transitive set of n-permutations, then for any distinct $r, s \in G$:*

$$d(r,s) \geq n - k + 1. \tag{8}$$

Proof. Let us assume that $d(r,s) < n-k+1$ for some $r, s \in G$. It means, the similarity $e(r,s) \geq k$ or both permutations act on some t-tuple ($t \geq k$) in the same way. This is a contradiction, since G is sharply k-transitive. □

Definition 3. *$G(n,d)$ denotes the size of the largest group of n-permutations with the pairwise distance at least d ($d \leq n$).*

An analogue of the *Singleton bound* can be obtained for permutations [11]:

Lemma 2. *The following upper bound holds:*

$$G(n,d) \leq \underbrace{n(n-1)(n-2)\cdots(d+1)d}_{n-d+1 \text{ multipliers}} \tag{9}$$

with equality if and only if there is a sharply $(n-d+1)$-transitive group of permutations.

Proof. We have $d(r,s) \geq d$ and $e(r,s) \leq n-d$ for every distinct $r,s \in G$. It means, if we fix any $(n-d+1)$-tuple x and apply all permutations from G to it, the obtained tuples y must be different. The number of such tuples y can not exceed the right hand side of (9).

If the size of the group G matches the upper bound, then all possible tuples y of $n-d+1$ distinct elements of S can be obtained if all permutations of G are applied to any fixed tuple x. Moreover, for each y there is no more than one such permutation. It means we can send any $(n-d+1)$-tuple x to any y with exactly one permutation thus the group is sharply $(n-d+1)$-transitive.

The other way round – if the group is sharply $(n-d+1)$-transitive, we can send any fixed $(n-d+1)$-tuple x to any y with exactly one permutation. Thus there are at least as many permutations in the group as the right hand side of (9). There are no other permutations, otherwise it would be possible to send the given x to some y with two different permutations, which is a contradiction with the assumption that the group is sharply $(n-d+1)$-transitive. □

In the next several subsections we list the main known results on sharply k-transitive groups (see [6] for the basic facts, [7, 9] for additional information).

4.1 Sharply n-transitive and $(n-1)$-transitive groups

It is clear that the *symmetric group* S_n of all permutations on the set $\{1, 2, \ldots, n\}$ is sharply n-transitive, because there is exactly one permutation that sends any n-tuple to any other n-tuple. However, S_n is also sharply $(n-1)$-transitive, because if the action of the permutation on $n-1$ elements is known, the action on the last element is uniquely determined. From Lemma 1 we obtain that the distance between distinct permutations of S_n is at least 2. It is clear, because distance 1 is not possible for permutations.

The group S_n can be generated by two generators (in cycle notation):

$$g_1 = (12)(3)(4)\ldots(n), \tag{10}$$
$$g_2 = (123\ldots n). \tag{11}$$

The first one corresponds to a transposition of first two elements, but the second one – to a cyclic shift of all elements. The group S_n consists of $n!$ permutations.

4.2 Sharply $(n-2)$-transitive groups

The *signature* or *sign* of a permutation s is defined as the parity of the number of *inversions* in s, i.e., pairs i, j such that $i < j$, but $s(i) > s(j)$. For example, $s = 3241$ has 4 inversions, namely 32, 31, 21, and 41 thus it is an even permutation. It is easy to show that a *transposition* (a permutation that swaps to elements) changes the sign of a permutation to the opposite. In fact the signature "sgn" of a permutation is a group homomorphism from S_n to $\{-1, 1\}$, because for any two permutations r and s we have $\operatorname{sgn}(s \circ r) = \operatorname{sgn} s \cdot \operatorname{sgn} r$. Therefore it is not hard to see that the set of all even permutations of the set $\{1, 2, \ldots, n\}$ forms a group – the *alternating group* A_n.

It is simple to show that A_n is sharply $(n-2)$-transitive – if we know the action of a permutation on $n-2$ elements, the remaining two elements can be either swapped or remain in the same order. One of these cases corresponds to an even permutation, but the other – to odd. From Lemma 1 it follows that the pairwise distance of distinct permutations of A_n is at least 3. This can also be obtained directly – even permutations can not have distance 2, because then they differ only by one transposition and therefore have different signs. Since the number of odd and even permutations is the same, A_n consists of $n!/2$ permutations.

4.3 Sharply 1-transitive groups

An example of a sharply 1-transitive group is the cyclic group C_n that consists of n permutations and is generated by a cyclic shift

$$g_1 = (123 \ldots n). \tag{12}$$

C_n is clearly sharply 1-transitive, because there is exactly one way how to shift any element to any other. The pairwise distance between distinct elements of C_n is exactly n.

4.4 Sharply 2-transitive groups

An infinite sequence of sharply 2-transitive groups can be constructed using an *affine transformation* $y(x) = ax + b$, where $a, b \in \mathbb{F}_n$, $a \neq 0$. Here $\mathbb{F}_n$ denotes the *finite field* of order n, i.e., a set of n elements together with two binary operations – addition and multiplication, such that $(\mathbb{F}_n, +)$ and $(\mathbb{F}_n \backslash \{0\}, *)$ are Abelian groups and both distribute laws hold. Such a field $\mathbb{F}_n$ exists if and only if n is a power of a prime number.

The function $y(x)$ acts on the elements of the field $\mathbb{F}_n$ as a permutation, because $ax_1 + b = ax_2 + b$ implies $x_1 = x_2$. There are in total $n(n-1)$ such permutations and they form a group: if $y_1(x) = a_1 x + b_1$ and $y_2(x) = a_2 x + b_2$, then $(y_1 \circ y_2)(x) = y_1(y_2(x)) = (a_1 a_2)x + (a_1 b_2 + b_1)$ which is also an affine transformation. To prove that the group is sharply 2-transitive, we have to show that there is a unique solution to the following system of two linear equations:

$$\begin{cases} y_1 = ax_1 + b \\ y_2 = ax_2 + b \end{cases} \tag{13}$$

where $x_1 \neq x_2$ and $y_1 \neq y_2$. This solution is

$$a = \frac{y_1 - y_2}{x_1 - x_2}, \quad b = \frac{x_1 y_2 - x_2 y_1}{x_1 - x_2}. \tag{14}$$

Thus the group is sharply 2-transitive. In a similar way one can explicitly show that the pairwise Hamming distance between distinct permutations is at least $n-1$, but it follows from Lemma 1 as well.

4.5 Sharply 3-transitive groups

Let $\mathbb{F}_q$ be a finite field, where q is a power of a prime number.

Definition 4. *The* multiplicative group $\mathbb{F}_q^*$ *of the field* $\mathbb{F}_q$ *is the set of all non-zero elements of* $\mathbb{F}_q$*, i.e.,* $F_q^* = \mathbb{F}_q \backslash \{0\}$.

Definition 5. *The* general linear group $\mathrm{GL}(m, \mathbb{F}_q)$ *is the set of all invertible* $m \times m$ *matrices with elements from the field* $\mathbb{F}_q$*. A matrix is* invertible *if it has a non-zero determinant.*

Definition 6. *The* projective general linear group $\mathrm{PGL}(m, \mathbb{F}_q)$ *is almost the same as* $\mathrm{GL}(m, \mathbb{F}_q)$*, except that matrices* $M, M' \in \mathrm{GL}(m, \mathbb{F}_q)$ *are treated as equal if there is a non-zero scalar* $c \in \mathbb{F}_q^*$ *such that* $M = cM'$*. In other words* $\mathrm{PGL}(m, q) = \mathrm{GL}(m, q)/\mathbb{F}_q^*$.

The matrices from the set $\mathrm{GL}(m, \mathbb{F}_q)$ form a group, because the matrix multiplication is associative, the product of two invertible matrices is also invertible and for each invertible matrix one can find an inverse. This group acts on the set of non-zero m-dimensional column vectors over $\mathbb{F}_q$ as a permutation. The set $\mathrm{PGL}(m, \mathbb{F}_q)$ consists of equivalence classes of matrices and is also a group. It acts on the equivalence classes of non-zero column vectors as a permutation.

The group $\mathrm{PGL}(m, \mathbb{F}_q)$ is sharply 3-transitive, when $m = 2$. To prove this, it is sufficient to show that for any two 3-tuples of vectors $X = (\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3)$ and $Y = (\mathbf{y}^1, \mathbf{y}^2, \mathbf{y}^3)$ from different equivalence classes there is exactly one matrix $M \in \mathrm{PGL}(2, \mathbb{F}_q)$ that sends the tuple X to Y. For vectors $\mathbf{x}^i$ and $\mathbf{y}^i$ this means

$$M \cdot \mathbf{x}^i = c_i \cdot \mathbf{y}^i, \tag{15}$$

where the constant c_i is introduced, because we are dealing with the equivalence classes of vectors. Thus for each $i \in \{1, 2, 3\}$ we have a linear system of equations

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_1^i \\ x_2^i \end{pmatrix} = c_i \begin{pmatrix} y_1^i \\ y_2^i \end{pmatrix}. \tag{16}$$

We have to solve these systems with respect to matrix M, but in addition we have three unknown constants: c_1, c_2, and c_3. In fact, we are allowed to choose one of them and thus specify some particular matrix M from its equivalence class. If $c_3 = 1$, the three systems of linear equations (16) are equivalent to

$$\begin{pmatrix} x_1^1 & x_2^1 & 0 & 0 & y_1^1 & 0 \\ 0 & 0 & x_1^1 & x_2^1 & y_2^1 & 0 \\ x_1^2 & x_2^2 & 0 & 0 & 0 & y_1^2 \\ 0 & 0 & x_1^2 & x_2^2 & 0 & y_2^2 \\ x_1^3 & x_2^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_1^3 & x_2^3 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ -c_1 \\ -c_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ y_1^3 \\ y_2^3 \end{pmatrix}. \tag{17}$$

It has a unique solution if the determinant does not vanish. Using some algebraic manipulations one can show that

$$\begin{vmatrix} x_1^1 & x_2^1 & 0 & 0 & y_1^1 & 0 \\ 0 & 0 & x_1^1 & x_2^1 & y_2^1 & 0 \\ x_1^2 & x_2^2 & 0 & 0 & 0 & y_1^2 \\ 0 & 0 & x_1^2 & x_2^2 & 0 & y_2^2 \\ x_1^3 & x_2^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_1^3 & x_2^3 & 0 & 0 \end{vmatrix} = \begin{vmatrix} x_1^1 & x_1^3 \\ x_2^1 & x_2^3 \end{vmatrix} \cdot \begin{vmatrix} x_1^2 & x_1^3 \\ x_2^2 & x_2^3 \end{vmatrix} \cdot \begin{vmatrix} y_1^1 & y_1^2 \\ y_2^1 & y_2^2 \end{vmatrix} \neq 0. \tag{18}$$

None of the three small determinants vanish, because we were given that the 3-tuples X and Y consist of vectors from different equivalence classes, but such vectors are clearly linearly independent. Therefore the big determinant clearly does not vanish as well and the system (17) has a unique solution.

Let us find the number of equivalence classes of vectors and matrices in $\mathrm{PGL}(2, \mathbb{F}_q)$. The number of 2-dimensional non-zero vectors over $\mathbb{F}_q$ is $q^2 - 1$. There are $q - 1$ non-zero constants and thus there are $q - 1$ vectors in each equivalence class. The number of classes is

$$n = \frac{q^2 - 1}{q - 1} = q + 1. \tag{19}$$

The number of matrices in $\mathrm{PGL}(2, \mathbb{F}_q)$ is

$$|\mathrm{PGL}(2, \mathbb{F}_q)| = \frac{|\mathrm{GL}(2, \mathbb{F}_q)|}{q - 1}, \tag{20}$$

because there are $q - 1$ non-zero constants. The number of matrices in $\mathrm{GL}(2, \mathbb{F}_q)$ is the same as the number of pairs of linearly independent non-zero vectors. The first vector can be chosen in $q^2 - 1$ ways and it determines a set of $q - 1$ linearly dependent vectors and the second vector can be chosen in $(q^2-1)-(q-1) = q^2-q$ ways. Therefore $|\mathrm{GL}(2, \mathbb{F}_q)| = (q^2 - 1)(q^2 - q)$ and

$$|\mathrm{PGL}(2, \mathbb{F}_q)| = (q + 1)q(q - 1) = n(n - 1)(n - 2). \tag{21}$$

The method for obtaining sharply 3-transitive groups given above can be described using a different formalism – the *linear fractional transformation* (also called *Möbius transformation*):

$$y(x) = \frac{ax+b}{cx+d}, \tag{22}$$

where $a, b, c, d \in \mathbb{F}_q$ and $ad - bc \neq 0$ (otherwise $a/c = b/d = \alpha$ and $y(x) = \alpha$). It acts on the set $\mathbb{F}_q \cup \{\infty\}$ as a permutation. The following conventions regarding the element ∞ are used:

$$y\left(-\frac{d}{c}\right) = \infty, \quad y(\infty) = \begin{cases} \infty & \text{if c=0,} \\ \frac{a}{c} & \text{otherwise.} \end{cases} \tag{23}$$

In fact, the element ∞ corresponds to the vector $\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ in the above construction. Note that the inverse of (22) is also a linear fractional transformation:

$$x(y) = \frac{-dy+b}{cy-a}. \tag{24}$$

The same stands for the composition of two linear fractional transformations.

4.6 Sharply 4-transitive and 5-transitive groups

It is known that the *Mathieu group* M_{11} is sharply 4-transitive and therefore the pairwise Hamming distance of distinct elements is at least 8. It consists of $11 \cdot 10 \cdot 9 \cdot 8 = 7920$ elements. It is generated by

$$g_1 = (2,10)(4,11)(5,7)(8,9)(1)(3)(6), \tag{25}$$
$$g_2 = (1,4,3,8)(2,5,6,9)(7)(10)(11). \tag{26}$$

The *Mathieu group* M_{12} is sharply 5-transitive and the pairwise Hamming distance of distinct elements is also at least 8. It has $12 \cdot 11 \cdot 10 \cdot 9 \cdot 8 = 95040$ elements and is generated by [10]:

$$g_1 = (1,2)(3,4)(5,6)(7,8)(9,10)(11,12), \tag{27}$$
$$g_2 = (1,2,3)(4,5,7)(8,9,11)(6)(10)(12). \tag{28}$$

4.7 Summary of sharply k-transitive groups

The Table 1 provides a summary of the sharply k-transitive groups discussed in the Sections 4.1 trough 4.6. These groups match the bound for $G(n,d)$ in Lemma 2 with equality. In Fig. 1 the points of the (n,d)-plane where the maximal value of $G(n,d)$ can be reached are shown. According to Table 1 these points can be divided into 5 infinite classes (indicated with lines) and two *sporadic groups* – the Mathieu groups. It turns out that there are no other groups of maximal size except the Mathieu groups M_{11} and M_{12} between the lines $d \geq 4$ and $d \leq n-3$:

Theorem 4 (see [7, 8, 10]). *A sharply k-transitive group ($k \geq 4$) is isomorphic either to S_n ($n \geq 4$), A_n ($n \geq 6$) or one of the Mathieu groups M_{11} or M_{12}.*

However, the non-existence of maximal groups does not imply that there are no groups with the required properties (see Sect. 6).

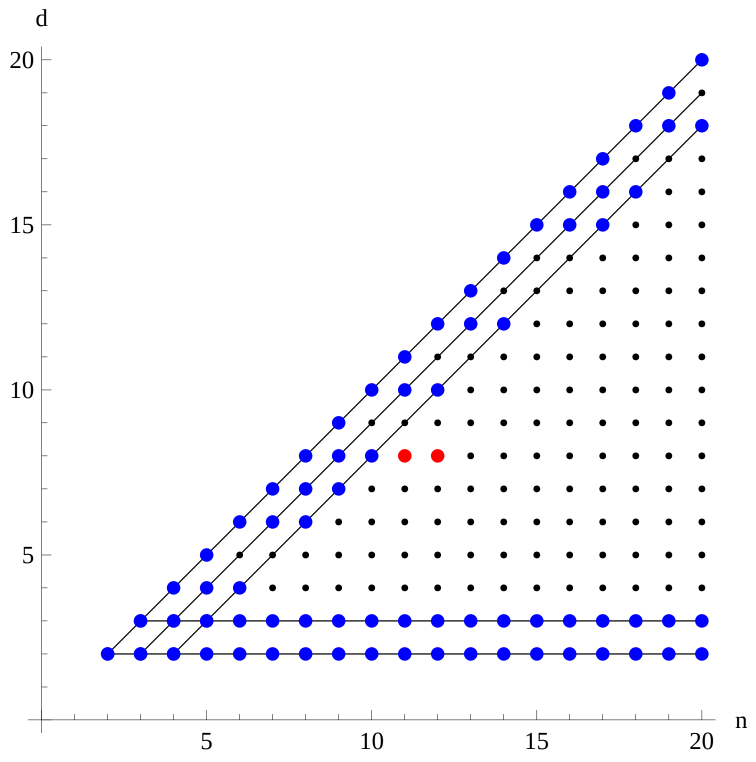


Fig. 1. The maximal permutation groups.

Table 1. The summary of sharply k-transitive groups. The meanings of columns are as follows: d – the pairwise Hamming distance, n – the possible values of the size of the set S (p^m stands for a power of any prime number), $|G_n|$ – the size of the group, G_n – the description of the group.

k	d	n	$\|G_n\|$	G_n	Section
$n, n-1$	2	any	$n!$	S_n	4.1
$n-2$	3	any	$n!/2$	A_n	4.2
5	8	12	95040	M_{12}	4.6
4	8	11	7920	M_{11}	4.6
3	$n-2$	p^m+1	$n(n-1)(n-2)$	$\mathrm{PGL}(2, \mathbb{F}_{n-1})$	4.5
2	$n-1$	p^m	$n(n-1)$	$y(x) = ax + b$	4.4
1	n	any	n	C_n	4.3

5 Permuting polynomials

The affine transformations considered in Sect. 4.4 were actually linear polynomials of x over the field $\mathbb{F}_n$. The linear fractional transformation considered in Sect. 4.5 is also a polynomial, because for any non-zero $a \in \mathbb{F}_q$ we have $a^{-1} \equiv a^{q-2}$ (this is a consequence of the analog of *Fermat's Little Theorem* $a^{p-1} \equiv 1 \mod p$ for finite fields). From this point of view it is interesting to study the *permuting polynomials* over finite fields, because they can give rise to permutation groups with large pairwise Hamming distance. In this section we give some basic results on groups generated by permuting polynomials.

Let us assume that the size n of the set S on which the permutations act is a power of a prime number (otherwise we can augment the set S with additional elements). Then we can put $\mathrm{S} = \mathbb{F}_n$ and express any permutation (actually any function) f on this set as a polynomial (we assume that f acts trivially on the appended elements if $|\mathrm{S}|$ was not a power of a prime number) as follows:

Definition 7. *The* Lagrange interpolating polynomial *of a function f defined on a finite field $\mathbb{F}_n$ as $f(x_1) = y_1$, $f(x_2) = y_2$, ..., $f(x_n) = y_n$ is given by:*

$$P(x) = \sum_{i=1}^{n} P_i(x), \quad \textit{where} \quad P_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} y_i. \tag{29}$$

Note that the division is performed in the field $\mathbb{F}_n$ and the denominator always differs from zero.

It is easy to see that the polynomial $P(x)$ mimics the function $f(x)$, i.e., $P(x) = f(x)$ for all $x \in \mathbb{F}_n$. As an example we will consider the permutation groups over the field $\mathbb{F}_5$.

The symmetric group S_5 is generated by $g_1(x) = x + 1$ and $g_2(x) = x^3$. It consists of all polynomials of the form:

$$ax^3 + bx^2 + 2a^3b^2x + d, \quad a \neq 0, \tag{30}$$

$$cx + d, \quad c \neq 0. \tag{31}$$

The alternating group A_5 is generated by $g_1(x) = x + 1$ and $g_2(x) = 2x^3$. It consists of all polynomials of the form (30) and (31), except that in addition we require that $a \in \{2, 3\}$ (non-squares) and $c \in \{1, 4\}$ (squares) respectively. Affine transformations are generated by $g_1(x) = x + 1$ and $g_2(x) = 2x$ and they are of the form (31). The cyclic group C_5 is generated by $g_1(x) = x + 1$ and is of the form (31) where $c = 1$.

As an example of a permutation group generated by polynomials when $|S|$ is not a power of a prime number, we can mention the case $G(6, 4) = 120$. The corresponding group is generated by $g_1(x) = 6x + 5$ and $g_2(x) = x^4 + 3x + 1$ where both polynomials are modulo 7.

Table 2. Experimentally obtained results for $G(n, d)$. The columns have the following meaning: n – the size of the set S, d – the pairwise Hamming distance, $G(n, d)$ – the size of the group obtained, "Bound" – the upper bound for $G(n, d)$ according to Lemma 2, "Generators" – the two generators of the group.

n	d	$G(n, d)$	Bound	Generators
7	4	168	840	$6, 4, 3, 2, 5, 1, 7$
				$6, 1, 7, 5, 2, 3, 4$
8	5	336	1680	$3, 8, 6, 2, 4, 5, 1, 7$
				$7, 4, 6, 3, 1, 5, 2, 8$
8	4	1344	6720	$2, 6, 8, 4, 5, 7, 1, 3$
				$7, 4, 3, 5, 1, 8, 6, 2$
9	6	1512	3024	$4, 5, 1, 8, 3, 7, 6, 2, 9$
				$3, 4, 8, 5, 7, 1, 6, 9, 2$
9	5	1512	15120	$9, 4, 1, 6, 5, 2, 7, 8, 3$
				$1, 4, 5, 3, 7, 9, 8, 2, 6$
9	4	1512	60480	$7, 2, 8, 3, 5, 6, 9, 4, 1$
				$6, 1, 3, 8, 2, 4, 9, 5, 7$
10	7	720	5040	$3, 9, 5, 7, 4, 8, 10, 6, 1, 2$
				$7, 9, 4, 5, 3, 6, 8, 1, 10, 2$
10	6	1512	30240	$8, 2, 10, 7, 4, 3, 1, 6, 5, 9$
				$1, 2, 8, 5, 10, 6, 3, 7, 9, 4$
10	5	1512	151200	$1, 10, 3, 9, 6, 8, 5, 4, 7, 2$
				$1, 10, 8, 3, 2, 4, 5, 7, 6, 9$
10	4	1920	604800	$5, 1, 4, 8, 9, 7, 6, 10, 2, 3$
				$7, 8, 2, 1, 10, 3, 9, 6, 4, 5$
15	12	2520	32760	$7, 2, 4, 5, 11, 10, 13, 15, 3, 9, 6, 8, 14, 12, 1$
				$9, 15, 11, 6, 4, 2, 10, 13, 7, 12, 8, 1, 14, 3, 5$
16	12	40320	524160	$16, 5, 6, 12, 14, 13, 11, 1, 10, 3, 7, 4, 15, 8, 9, 2$
				$6, 7, 14, 8, 15, 3, 12, 2, 9, 10, 13, 11, 4, 16, 1, 5$

6 Experimental results

We performed computer experiments to find permutation groups with pairwise Hamming distance in the region between $d \geq 4$ and $d \leq n - 3$. The obtained results for $n = 7, 8, 9, 10$ are shown in Table 2. In addition we mention also two large groups for $n = 15$ and $n = 16$.

These groups were obtained by choosing two random permutations g_1 and g_2 and computing their closure with respect to the product of permutations. If at some point the distance between any two distinct obtained permutations became less than some predefined d_{min}, the process was terminated and restarted with another random generators g_1 and g_2. Some of the groups obtained in this way have very interesting properties:

(1) $G(7, 4)$ has $168 = 7 \cdot 6 \cdot 4$ elements and is isomorphic to the automorphism group of the *Fano plane*.
(2) $G(8, 4)$ has $1344 = 8 \cdot 168 = 8 \cdot 7 \cdot 6 \cdot 4$ elements. This group has the property, that the stabilizers of any element form a group that is isomorphic to the automorphism group of the Fano plane. This group also has a property that for any 3-tuples x and y of distinct elements there are exactly 4 permutations that send x to y. It is isomorphic to the automorphism group of the *octonion* multiplication table.
(3) $G(8, 4)$ has $1512 = 9 \cdot 168 = 9 \cdot 8 \cdot 7 \cdot 3$ elements and it has the same stabilizer property, but for each 3-tuples x and y there are exactly 3 permutations that send x to y.
(4) $G(15, 12)$ has $2520 = 15 \cdot 168 = 15 \cdot 14 \cdot 12$ elements and it also has the stabilizer property, but for each 2-tuples x and y there are exactly 12 permutations that send x to y.
(5) $G(16, 12)$ has $40320 = 16 \cdot 15 \cdot 168 = 16 \cdot 15 \cdot 14 \cdot 12$ elements. The stabilizers of any two elements form a group that is isomorphic to the automorphism group of the Fano plane. For any 3-tuples x and y there are exactly 12 permutations that send x to y.

References

1. A.Ambainis, R.Freivalds. 1-way quantum finite automata: strengths, weaknesses and generalizations. *Proc. IEEE FOCS'98*, pp. 332–341, 1998.
2. A.Ambainis. The complexity of probabilistic versus deterministic finite automata. *Lecture Notes in Computer Science,* Springer, v.1178, pp.233–237, 1996.
3. R.Freivalds. Non-constructive methods for finite probabilistic automata. Accepted for *The 11th International Conference Developments in Language Theory DLT-2007*, Turku, Finland, July 3-5, 2007.
4. D.Aharonov, A.Kitaev, N.Nisan. Quantum circuits with mixed states. *Proc. STOC 1998*, pp. 20-30, 1998.
5. A.Ambainis, M.Beaudry, M.Golovkins, A.Ķikusts, M.Mercer, D.Thérien. Algebraic Results on Quantum Automata. *Theory Comput. Syst.*, v. 39(1), pp. 165–188, 2006.
6. A.Wool. Sharply Transitive Permutation Groups, 2006.

7. P.J.Cameron. *Permutation Groups*, Chapter 12, pp. 611–645 in R.L.Graham, M.Grötschel, L.Lovàsz. *Handbook of Combinatorics*, Vol 1, Elsevier Science B.V., The MIT Press, 1995.
8. G.F.Pilz. Near-rings and Near-fields, pp. 463–498 in M.Hazewinkel. *Handbook of Algebra*, Vol 1, Elsevier Science B.V., 1996.
9. K.Itô. *Encyclopedic Dictionary of Mathematics*, 2nd ed., 151 H. *Transitive Permutation Groups*, pp. 591–593, The MIT Press, 1987.
10. R.F.Bailey. Decoding the Mathieu Group M_{12}.
11. P.J.Cameron. Permutation Codes, talk at CGCS, 2007.
12. E.Artin. Beweis des allgemeinen Reziprozitätsgesetzes. *Mat. Sem. Univ. Hamburg*, B.5 , 353–363, 1927.
13. M.Aschbacher. *Finite Group Theory*, (Cambridge Studies in Advanced Mathematics), Cambridge University Press, 2nd edition, 2000.
14. E.Bach, J.Shallit. *Algorithmic Number Theory.*, vol. 1, MIT Press, 1996.
15. A.Cobham. The recognition problem for the set of perfect squares. *Proc. 7th Ann. Symp. Switching and Automata Theory*, pp. 78–87, 1966.
16. Y.L.Ershov. Theory of numberings, *Handbook of computability theory* (E.R.Griffor, ed.), North-Holland, Amsterdam, pp. 473-503, 1999.
17. R.Freivalds. On the growth of the number of states in result of the determinization of probabilistic finite automata. *Avtomatika i Vichislitel'naya Tekhnika*, No. 3, pp. 39–42, 1982. (Russian)
18. N.Z.Gabbasov, T.A.Murtazina. Improving the estimate of Rabin's reduction theorem. *Algorithms and Automata*, Kazan University, pp. 7–10, 1979. (Russian)
19. P.Garret. *The Mathematics of Coding Theory.*, Pearson Prentice Hall, Upper Saddle River, 2004.
20. C.Hooley. On Artin's conjecture. *J.ReineAngew.Math.*, v. 225, pp. 229–220, 1967.
21. D.R.Heath-Brown. Artin's conjecture for primitive roots. *Quart. J. Math. Oxford*, v. 37, pp. 27–38, 1986.
22. A.Kondacs, J.Watrous. On the power of quantum finite state automata. *Proc. IEEE FOCS'97*, pp. 66–75, 1997.
23. C.Moore, J.Crutchfield. Quantum automata and quantum grammars. *Theor. Comput. Sci.*, v. 237(1-2), pp. 275–306, 2000.

First Order Logic and Acceptance Probability of Quantum Finite State Automata

Ilze Dzelme-Bērziņa [⋆]

Institute of Mathematics and Computer Science, University of Latvia, Raiņa bulvāris 29, LV–1459, Riga, Latvia.

ilze.dzelme@gmail.com

Abstract. The connection between formulae of the first order logic and acceptance probability of quantum finite automata has been studied in this paper. It has been proved that there is a subclass of the first order languages that can be recognized by measure-many quantum finite state automata with probability 1. Other properties of the first order languages and acceptance probability of measure-many quantum finite state automata have been studied in the paper.

1 Introduction

The connection between automata theory and logic dates back to the early sixties to the work of Büchi [8] and Elgot [12]. They showed how a logical monadic second-order formula can effectively be transformed into a finite state automaton accepting the language defined by the formula and vice versus - how a finite state automaton can be transformed to a logical monadic second-order formula which specifies the language accepted by the automaton. Later the equivalence between finite state automata and monadic second-order Logics over infinite words and trees were shown in the works of Büchi [9], McNaughton [20], and Rabin [24]. The next important step in the connection between automata theory and logic was Pnueli's work [23], where it was proposed to use Temporal Logic for reasoning about continuously operating concurrent programs. And so in the eighties, temporal Logics and fixed-point Logics took the role of specification languages and more efficient transformations from logic formulae to automata were found. The results of the research were powerful algorithms and software systems for the verification of finite state programs ("model-checking"). The research of the equivalence between automata theory and logic formalism also influenced language theory itself. For example, automata classes were described in the terms of logic.

The logical description of the behavior of computation models also influenced complexity theory. In 1974 Fagin [13] gave a characterisation of nondeterministic polynomial time (NP) as the set of properties expressible in the second order existential logic. Later Immerman [15] and Vardi [26] characterized polynomial

⋆ The research has been supported by the European Social Fund.

time as the set of properties expressed in a first order inductive definition, which is defined by adding a least point operator to the first order logic. And in the similar way polynomial space has also been characterized.

A natural model of classical computing with finite memory is a finite state automaton, likewise a quantum finite state automaton is a natural model of quantum computation. Different notations of quantum finite state automata are used. The two most popular notations of quantum finite state automata are quantum finite state automata introduced by Moore and Crutchfield [21] (measure-once quantum finite state automata) and quantum finite automata introduced by Kondacs and Watrous [18] (measure-many quantum finite state automata). They have a seemingly small difference, in the first definition a quantum finite state automaton performs the measurement only at the end of the computation, but in the second definition a quantum finite state automaton performs the measurement at every step of the computation. Measure-once quantum finite state automata and measure-many quantum finite state automata with isolated cut point recognize only the subset of regular languages. Besides these two models of quantum finite state automata there are also such models of quantum finite state automata as "enhanced" quantum finite state automata [22], Latvian quantum finite state automata [2], 1-way quantum finite state automata with control language [7], quantum finite state automata with mixed states (introduced by D.Aharonov, A.Kitaev and N.Nisan [1]) and quantum finite state automata with quantum and classical states (introduced by A.Ambainis and J.Watrous [5]).

Quantum finite state automata have their strengths and weaknesses in comparison to their classical counterparts. The strength of quantum finite state automata is in the fact that quantum finite automata can be exponentially more effective [4], but the main weakness is caused by necessity that a quantum process has to be reversible, that makes the most of quantum finite state automata notations unable to recognize all regular languages. And for many notations of quantum finite state automata the problem to describe the class of the languages recognizable by the quantum automata is still open and as logic has had a large impact on automata theory, it seams to be useful to look at the languages recognizable by quantum finite state automata in the terms of logic, as well as to look at the properties of quantum finite automata in the terms of logic.

The connection between quantum finite state automata and logic has been studied in [10], where it was shown that measure-once quantum finite state automata does not accept the languages defined by the first order logic except the trivial languages. The connection between the first order logic and measure-many quantum finite automata has been studied in [11] where the forms of the first order formulae have been presented for which it is possible to construct a measure-many quantum finite automaton recognizing the language and the construction of the first order formulae have been shown for which there is no measure-many quantum finite automaton recognizing the language defined by the first order formula.

In the paper we study an acceptance probability of measure-many quantum finite state automata and the first order logic. The language class of the first order languages are presented for which it is possible to construct a measure-many quantum finite state automaton recognizing the language with probability 1.

2 Main notations

2.1 First order logic

Let A be a finite alphabet and let $\omega = a_1a_2...a_n$ be a word over the alphabet A. The corresponding *word model* for the word ω is represented by the relational structure

$$\underline{\omega} = (dom(\omega), S, <, (Q_a)_{a \in A})$$

where $dom(\omega) = \{1, 2, ..., n\}$ is the set of the letters "positions" of ω (the "domain" of ω), S is the successor relation on $dom(\omega)$ with $(i, i+1) \in S$ for all $1 \leq i \leq n$, $<$ is the order relation on $dom(\omega)$, and $Q_a = \{i \in dom(\omega) \mid a_i = a\}$ ("position carries letter a").

We consider word models over the finite alphabet A. The corresponding first order language has variables $x, y, ...$ ranging over positions in the word models, and is built from atomic formulae of the form

$$x = y,\ S(x, y),\ Q_a(x),\ x < y$$

by means of the connectivities $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ and quantifiers $\exists$ and $\forall$. The notation $\varphi(x_1, x_2, ..., x_n)$ indicates that in the formula φ at most the variables $x_1, x_2, ..., x_n$ are free, i.e. they are not in the scope of a quantifier. A *sentence* is a formula with no free variables. If $p_1, p_2, ..., p_n$ are positions from $dom(\omega)$ then $(\underline{\omega}, p_1, p_2, ..., p_n) \models \varphi(x_1, x_2, ..., x_n)$ means that φ is satisfied in the word model $\underline{\omega}$ when $p_1, p_2, ..., p_n$ serve as an interpretation of $x_1, x_2, ..., x_n$. The language defined by the sentence φ is $L(\phi) = \{\omega \in A^* \mid \underline{\omega} \models \varphi\}$. Languages defined by such sentences are the first order languages. For example, the sentence $\forall x(Q_a(x))$ over the alphabet $A = \{a, b\}$ defines the language containing all words that have only letters a. This language is a first order language. The classical equivalence result of the first order logic is results by Schützenberger [25]:

Theorem 1. *For a language $L \in A^*$ the following are equivalent*

1. *L is star-free (the smallest class that satisfies following: all finite languages over A belong to star free languages, if languages L_1, L_2 are star free then so are $L_1 \cdot L_2$, $L_1 \cup L_2$, $L_1 \cap L_2$ and $\bar{L}_1 = A^* \setminus L$).*
2. *L is recognizable by a finite aperiodic monoid - a finite monoid M for which there is and $n \geq 1$ such that $m^{n+1} = m^n$ holds for all $m \in M$.*
3. *L is defined by a first order formula.*

2.2 Quantum finite automata

Definition 1. *A measure-many quantum finite state automaton is defined by a 6-tuple as follows [18]*

$$A = (Q; \Sigma; \delta; q_0; Q_{acc}; Q_{rej})$$

where

1. *Q is a finite set of states,*
2. *Σ is an input alphabet and Γ=$\Sigma \cup \{$ $\sharp$; \$ $\}$ is working alphabet of A, where $\sharp$ and \$ ($\notin \Sigma$) are the left and the right end-markers,*
3. *δ is the transition function $\delta : Q \times \Gamma \times Q \to C_{[0,1]}$, which represents the amplitudes that flows from the state q to the state q' after reading symbol σ,*
4. *$q_0 \in Q$ is the initial state,*
5. *$Q_{acc} \subseteq Q$ and $Q_{rej} \subseteq Q$ are sets of accepting and rejecting states ($Q_{acc} \cap Q_{rej} = \oslash$).*

The states in Q_{acc} and Q_{rej} are halting states and the states in $Q_{non} = Q \setminus (Q_{acc} \cup Q_{rej})$ are non-halting states.

For all states $q_1, q_2, q' \in Q$ and symbols $\sigma \in \Gamma$, the function δ must be unitary, thus the function satisfies the condition

$$\textstyle\sum_{q'} \overline{\delta(q_1, \sigma, q')}\delta(q_2, \sigma, q') = \begin{cases} 1 \ (q_1 = q_2) \\ 0 \ (q_1 \neq q_2) \end{cases} .$$

And it is assumed that an input word starts with the left end-marker and ends with the right end-marker.

The linear superposition of the automaton's A states is represented by a n-dimensional complex unit vector, where $n = |Q|$. The vector is denoted by $|\phi\rangle = \sum_{i=1}^{n} \alpha_i \, |q_i\rangle$, where $\{|q_i\rangle\}$ is the set orthonormal basis vectors corresponding to the states of the automaton A.

The transition function δ is represented by a set of unitary matrices $\{V_\sigma\}_{\sigma \in \Gamma}$, where V_σ is the unitary transition of the automaton A after reading the symbol σ and is defined by $V_\sigma(| \, q\rangle) = \sum_{q' \in Q} \delta(q, \sigma, q') \mid q'\rangle$.

A computation of the automaton A on the input word $\sharp\sigma_1\sigma_2...\sigma_n\$$ proceeds as follows. It starts in the superposition $\mid q_0\rangle$, then a transition corresponding to the current input letter is performed. After every transition the automaton A measures its state with respect to the observable $E_{acc} \oplus E_{rej} \oplus E_{non}$ where $E_{acc} = span\{| \, q\rangle : q \in Q_{acc}\}$, $E_{rej} = span\{| \, q\rangle : q \in Q_{rej}\}$ and $E_{non} = span\{| \, q\rangle : q \in Q_{non}\}$. If the observed state of the automaton A is in E_{acc} subspace, then it accepts the input; if the observed state of A is in E_{rej} subspace, then it rejects the input, otherwise the computation continues. After every measurement the superposition collapses to the measured subspace. A measurement is represented by a diagonal zero-one projection matrices P_{acc}, P_{rej} and P_{non} which project the vector onto E_{acc}, E_{rej} and E_{non}.

Since the automaton A can have a non-zero probability of halting, it is useful to keep a track of the cumulative accepting and rejecting probabilities. Therefore, the state of the automaton A is represented by a triple (ϕ, p_{acc}, p_{rej}), where p_{acc} and p_{rej} are the cumulative probabilities of accepting and rejecting. The transition of A on reading the symbol σ is denoted by $(P_{non} |\phi'\rangle, p_{acc} + \|P_{acc}\phi'\|^2, p_{rej} + \|P_{rej}\phi'\|^2)$, where $\phi' = V_\sigma \phi$.

3 First order logic and quantum finite automata

In this section we look at the connection between accepting probabilities of quantum finite automata and the first order logic. From the fact that intersection of languages recognized by measure-once quantum finite automata and languages defined by the first order logic follows that there are languages recognized by measure many quantum finite automata which cannot be defined by the first order logic. However there are first order languages recognized by measure-many quantum finite automata with probability 1.

Theorem 2. *The first order languages contains languages which can be recognized by a measure-many quantum finite automaton with probability 1.*

Proof. It is possible to define a class L_1, which contains all languages defined by the following rules:

1. $a_i \in \Sigma$, $\{a_1 a_2 ... a_k \Sigma^*\} \in L_1$, $k \in N$
2. if $L_i \in L_1$ and $L_j \in L_1$, then $\overline{L_i} \in L_1$, $L_i \cup L_j \in L_1$ and $L_i \cap L_j \in L_1$.

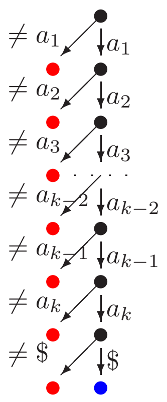
And languages can be defined by the first order formula. The first order formula describing the language can be constructed as follows:

1. $\forall x_1, x_2, ..., x_k(first(x_1) \rightarrow Q_{a_1}(x_1) \wedge S(x_1, x_2) \rightarrow Q_{a_2}(x_2) \wedge S(x_2, x_3) \rightarrow ... \rightarrow Q_{a_{k-1}}(x_{k-1}) \wedge S(x_{k-1}, x_k) \rightarrow Q_{a_k}(x_k)\{\wedge last(x_k)\}^*)$
2. if ϕ_i defines language $L_i \in L_1$ and ϕ_j recognizes the language $L_j \in L_1$, then $\neg\phi_i$ recognizes $\overline{L_i}$, $\phi_i \vee \phi_j$ - $L_i \cup L_j \in L_1$ and $\phi_i \wedge \phi_j$ - $L_i \cap L_j \in L_1$.

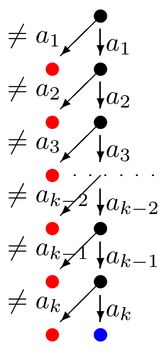
Lemma 1. *The languages in L_1 can be recognized by a measure-many quantum finite automaton with probability 1.*

Proof. Lets look at the language $L_i \in L_1$ to construct the measure-many quantum finite state automaton for the language L_i, we need to represent the language L_i in the tree view. The representation tree is constructed in the following way (there is exactly one edge from the parent node of level k to k+1 with a label a_i $\forall a_i \in \{\Sigma \cup \$\}$) :

- *The representation tree for the language $\{a_1 a_2 ... a_k\}$.* The first level is labeled with 0. The edge from i to i+1 is labeled with a letter a_{i+1} $(i < k)$, the edges from the level k to k+1 are drawn for all symbols in Σ and the children nodes are colored in red. The node of the level k+1 which is connected by the edge from the level k to $k+1$ labeled with the right end-marker is colored in blue. Afterwards the edges are added to parent nodes to make the whole tree, so that there is exactly one edge from the parent node of the level j to j+1, the children nodes (leaves) are marked in red.



Representation tree for $\{a_1 a_2 ... a_k\}$



Representation tree for $\{a_1 a_2 ... a_k \Sigma^*\}$

- *The representation tree for the language* $\{a_1 a_2 ... a_k \Sigma^*\}$. The edge from i to i+1 is labeled with a letter a_{i+1}, the last node is colored in blue. Afterwards the edges are added to parent nodes to make the whole tree, so that there is exactly one edge from the parent node of the level j to j+1. The children nodes (leaves) are marked in red.
- For $\overline{L_i}$ the L_i tree blue nodes are colored in red and the red nodes in blue.
- The tree of $L_i \cup L_j$ is union of the trees for L_i and L_j. If the edge with a label a from the level k to $k+1$ in one of the trees goes to red leaf, then the subtree of the other tree is chosen in the final tree, if the leaf in the one of the trees is blue, then the final tree will have this edge.
- The tree of $L_i \cap L_j$ is intersection of the trees for L_i and L_j. If the edge with a label a from the level k to $k+1$ in one of the trees goes to red leaf, then the final tree will have this edge, if the leaf in the one of the trees is blue, then the subtree of the other tree is chosen in the final tree.

The measure-many quantum finite state automaton accepting the language L_i with probability 1 is the automaton $A = (Q; \Sigma; \delta; q_0; Q_a; Q_r)$, where $Q = \{q_0, q_1, ..., q_n\}$, where n is the count of the nodes. Q_a contains the states which correspond to nodes colored in blue, Q_r contains the states which correspond to nodes colored in red. The transition function are defined by the representation tree. The edge (i, j, a_l) defines the transition $|q_i\rangle = |q_j\rangle$ for letter a_l. The transition function for the left end-marker is unitary transition.

Lets look at the languages that can be recognized by MM-QFA with accepting probability 1 and which cannot be recognized by measure-once quantum finite state automata. Currently we have shown a specific class of languages which can be recognized by MM-QFA with probability 1 and which are first order definable. Naturally, a question arises:

- Are there other first order languages which can be recognized by MM-QFA with probability 1, but are not in the language class L_1?

The answer to which is yes, for example, a language ab^*a can be recognized by MM-QFA with probability 1 and it is first order definable. Another question for further investigation is:

– Is it possible to give a characteristics of the languages which can be recognized by MM-QFA with probability 1, but which cannot be recognized by MO-QFA and are not first order definable? It is clear that such languages exist, for example, $a^{2k}b$ where $K \in Z$.

But now lets look at the other issue - can we present the first order languages which cannot be recognized by a measure-many quantum finite state automaton with probability 1? The first order languages contain such languages which can only be recognized by a measure-many quantum finite automaton with probability less than 1. One of examples is the language a^*b^*.

Theorem 3. *There is no such probability p greater than $\frac{1}{2}$ for which holds following:*

– *any measure-many quantum finite automaton accepts the first order languages (recognized by MM-QFA) at least with probability p.*

Proof. Lets assume that it is possible to provide such probability p. We can express p as $\frac{1}{2} + l$. It is possible to find such n so that $l > \frac{3}{\sqrt{n-1}}$. At the same time it is known [3], that language L_n (L_n is defined as $a_1^*a_2^*a_3^*...a_n^*$) cannot be recognized with probability greater then $\frac{1}{2} + \frac{3}{\sqrt{n-1}}$. We got a contradiction which means that such p does not exist.

References

1. D. Aharonov, A. Kitaev and N. Nisan. *Quantum Circuits with Mixed States.* STOC, 20-30, 1998.
2. A. Ambainis, M. Beaudry, M. Golovkins, A. Kikusts, M. Mercer, D. Thrien. *Algebraic Results on Quantum Automata.* STACS 2004, 93-104, 2004
3. Andris Ambainis, R. F. Bonner, R. Freivalds, A. Kikusts. *Probabilities to Accept Languages by Quantum Finite Automata.* COCOON 1999, 174-183, 1999
4. A. Ambainis and R. Freivalds. *1-way quantum finite automata: strengths, weaknesses and generalizations.* FOCS, 332-341, 1998.
5. A. Ambainis and J. Watrous. *Two-way finite automata with quantum and classical states.* Theoretical Computer Science 287, No.1, 299-311, 2002.
6. F. Baader. *Automata and Logic.* Technische Universität Dresden, 2003.
7. Alberto Bertoni, Carlo Mereghetti, Beatrice Palano. *Quantum Computing: 1-Way Quantum Automata.* Developments in Language Theory 2003: 1-20, 2003.
8. J.R. Büchi. *Weak second-order arithmetic and finite automata.* Z. Math. Logik Grundl. Math 6:66-92, 1960.
9. J.R. Büchi. *On decision method in restricted second-order arithmetic.* Int. Congr. for Logic, Methodology and Philosophy of Science, Stanford Univ. Press, Stanford, 1-11, 1962.
10. I.Dzelme. *Quantum Finite Automata and Logics.* SOFSEM 2006, 246-253, 2006.

11. I. Dzelme-Bērziņa *Formulas of first order logic and quantum finite automata.* QCMC 2006.
12. C.C. Elgot. *Decision problems of finite automata design and related arithmetics.* Trans. Amer. Math Soc. 98:21-52, 1961.
13. R. Fagin *Generalized first-order spectra and polynomial-time recognizable sets.* Complexity of Computation, SIAM-AMS Proceedings 7, 43-73, 1974.
14. J. Gruska. *Quantum Computing.* McGraw-Hill, 439, 1999.
15. N. Immerman. *Relational queries computable in polynomial time.* Information and Control, 68:86-104, 1986.
16. N. Immerman. *Languages that capture complexity classes.* SIAM J. Comput 16:4, 760 - 778, 1987.
17. N. Immerman. *Nondeterministic space is closed under complementation.* SIAM J. Comput 17:5, 953-938, 1988.
18. A. Kondacs, J. Watrous. *On the power of quantum finite state automata.* FOCS, 66-75, 1997.
19. R. McNaugton. *Symbolic logic and automata.* Technical Note, 60-244, 1960.
20. R. McNaughton. *Testing and generating infinite sequences by finite automaton.* Inform. Contr. 9, 521-530, 1966.
21. C. Moore, J. Crutchfield. *Quantum automata and quantum grammars.* Theoretical Computer Science, 237:275-306, 2000.
22. A. Nayak. *Optimal Lower Bounds for Quantum Automata and Random Access Codes.* FOCS, 369-377, 1999.
http://arxiv.org/abs/quant-ph/9904093
23. A. Pnueli. *The temporal logic of programs.* FOCS, 1-14, 1977.
24. M.O. Rabin. *Decidability of second-order theories and automata on infinite trees.* Trans. Amer. Math. Soc. 141:1-35, 1969.
25. M.P. Schüzenberger. *On finite monoids having only trivial subgroups.* Information and Control 8:283-305, 1965.
26. M. Vardi. *Complexity of relational query languages.* STOC, 137-146, 1982.

On a Class of Languages Recognizable by Probabilistic Reversible Decide-and-Halt Automata [⋆]

Marats Golovkins, Maksim Kravtsev, and Vasilijs Kravcevs

Institute of Mathematics and Computer Science, University of Latvia,
Raiņa bulv. 29, Riga, Latvia
marats@latnet.lv, maksims.kravcevs@lu.lv, md80004@lanet.lv

Abstract. We analyze the properties of probabilistic reversible decide-and-halt automata (DH-PRA) and show that there is a strong relationship between DH-PRA and 1-way quantum automata. We show that a general class of regular languages is not recognizable by DH-PRA by proving that two "forbidden" constructions in minimal deterministic automata correspond to languages not recognizable by DH-PRA. The shown class is identical to a class known to be not recognizable by 1-way quantum automata. We also prove that the class of languages recognizable by DH-PRA is not closed under union and other non-trivial boolean operations.

1 Introduction

In this paper we study probabilistic reversible decide-and-halt automata, introduced in [GK 02]. Being entirely classical, the model however has close links to quantum finite automata (QFA). Moreover, with some additional restrictions, DH-PRA may be considered as a marginal special case of Nayak's quantum automata [N 99]. Such marginal, essentially classical, special cases sometimes prove to be extremely useful in the research of the properties of QFA. For example, classical probabilistic reversible automata (C-PRA) are instrumental to prove that Latvian QFA [ABG 04] recognize exactly the regular languages whose syntactic monoids are block groups.

In this paper, we consider bounded-error language recognition by DH-PRA. There are two commonly used models how to interpret word acceptance and hence, language recognition, by quantum automata. In *classical* acceptance model the states are divided into two disjoint sets of accepting and non-accepting states and the automaton accepts a word, if it is in accepting state after having read the last symbol of the word and rejects it otherwise. In *decide-and-halt* acceptance model the states are divided into three disjoint sets of accepting, rejecting

⋆ Research supported by the European Social Fund, contract No. 2004/0001/VPD1/ESF/PIAA/04/NP/3.2.3.1/0001/0001/0063, by the Latvian Council of Science, grant No. 05.1528 and by the University of Latvia research project No. 2006/1-229715.

and non-halting states, and after reading each symbol of the word, the automaton accepts the word if it is in accepting state, rejects if in rejecting state and continues computation otherwise.

Classical one-way QFA (1-QFA) with pure states were introduced by C. Moore and J. P. Crutchfield in [MC 00]. Subsequently, A. Kondacs and J. Watrous introduced "decide-and-halt" 1-QFA with pure states in [KW 97]. Classical 1-QFA with pure states and "decide-and-halt" 1-QFA with pure states are commonly referred in literature as *measure-once* QFA (MO-QFA) and *measure-many* QFA (MM-QFA), respectively. Since then 1-QFA with pure states have been studied a lot. In particular, Kondacs and Watrous showed in [KW 97], that MM-QFA can recognize only a proper subset of regular languages. In [BP 99], Brodsky and Pippenger noted that MO-QFA recognize the same language class as permutation automata ([T 68]). Ambainis, Ķikusts and Valdats determined in [AKV 01] that the class of languages recognized by MM-QFA is not closed under boolean operations, as well as significantly improved the necessary condition of a language to be recognized by MM-QFA, proposed by [BP 99]. Recently in [BMP 03] it is shown that the class of languages recognizable by MM-QFA coincides with a certain subclass of QFA with control language, but still it does not give the description of that class. "Decide-and-halt" 1-QFA with mixed states were introduced by A. Nayak in [N 99] as *enhanced* quantum finite automata. He showed that similar weaknesses apply to this more general model (MM-QFA is a special case of it) as shown for MM-QFA.

Probabilistic reversible automata (PRA) were introduced by M. Golovkins and M. Kravtsev in [GK 02] as probabilistic automata which transition function is determined by doubly stochastic operators. In case of one-way automata that means that a doubly stochastic transition matrix corresponds to each symbol in alphabet. This model is a probabilistic counterpart for Nayak's model of enhanced quantum automata. In Nayak's model, if a result of every observation is a single configuration, not a superposition of configurations, we get a probabilistic automaton which evolution matrices are *doubly* stochastic. So if the transition matrices of a PRA are also unitary stochastic it is in fact a Nayak's quantum automaton.

In this paper we address another type of PRA, i.e., the DH-PRA defined in [GK 02], that behave more like measure-many quantum automata [KW 97], as they halt when entering accepting or rejecting states. We show a general class of regular languages, not recognizable by DH-PRA. This class is identical to a class not recognizable by MM-QFA [AKV 01] (and similar to the class of languages, not recognizable by C-PRA [GK 02]). We also prove that the class of languages recognizable by DH-PRA is not closed under union. That makes more credible our assumption that there exists a strong relationship between DH-PRA and MM-QFA, leading to a conjecture that DH-PRA are at least as powerful as MM-QFA.

The paper is structured in the following way. Section 2 contains definitions and general facts used throughout the paper. In Section 3 we show a class of languages that are not recognizable by DH-PRA. In Section 4 we prove that the class of languages recognizable by DH-PRA is not closed under union. Section 5 is the conclusion.

2 Preliminaries

We refer to Appendix for the definitions of *classical* and *decide-and-halt acceptance* and the general definition of probabilistic reversible automata.

A 1-way probabilistic reversible decide-and-halt automaton (DH-PRA) $A = (Q, \Sigma, q_0, Q_a, Q_r, \delta)$ is specified by a finite set of states Q, a finite input alphabet Σ, an initial state $q_0 \in Q$ and a transition function $\delta : Q \times \Gamma \times Q \longrightarrow \mathbb{R}_{[0,1]}$, where $\Gamma = \Sigma \cup \{\#, \$\}$ is the input tape alphabet of A and $\#$, $\$$ are end-markers not in Σ. For every input symbol $\sigma \in \Gamma$ the transition function is determined by a doubly stochastic $|Q| \times |Q|$ matrix V_σ, where $(V_\sigma)_{i,j} = \delta(q_j, \sigma, q_i)$. The set of accepting states is Q_a and the set of rejecting states is Q_r, where $Q_a \cap Q_r = \emptyset$. Every input word is enclosed into end-marker symbols $\#$ and $\$$. Once an automaton enters any state in $Q_a \cup Q_r$ the computation is halted and the input word is either accepted or rejected. Let $Q_n = Q \setminus (Q_a \cup Q_r)$ the set of nonhalting states. Any state in $Q_a \cup Q_r$ is called halting.

Following notation used in quantum computation, throughout the paper, column vectors will be used in connection with the transition matrices, hence the order of multiplication of transition matrices will be opposite to the order of letters in the input word.

Let us note that any DH-PRA can be transformed into a probabilistic automaton with classical acceptance which recognizes the same language (end-markers remain present). Indeed, for any halting state q let us modify the transition function, so that for any σ in Γ $q \cdot \sigma = q$. The set of final states is Q_a. Let l the number of the previously halting states.

The transition matrices V_{σ_r} remain stochastic, but are not anymore doubly stochastic. However, for each transition matrix V_{σ_r} we can enumerate the states of DH-PRA in the following way:

1. q_1 ... q_{k_r} are the states from which the halting states are not accessible with σ_r;
2. q_{k_r+1} ... q_{n-l} are the non-halting states from which the halting states are accessible with σ_r;
3. q_{n-l+1} ... q_n are the halting states.

Then the structure of the transition matrix $V_{\sigma_r} = (\alpha_{ij})$ is $\begin{pmatrix} DST & 0 & 0 \\ 0 & a_{ij} & 0 \\ 0 & b_{ij} & I \end{pmatrix}$, where

- DST - $k_r \times k_r$ doubly stochastic matrix,
- I - $l \times l$ unit matrix,
- $\forall i,\ k_r + 1 \leq i \leq n - l, \sum\limits_{j=1}^{n} \alpha_{ij} \leq 1$ (as originated from doubly stochastic matrix where the sum of each row is one)

Note that for different letters of the alphabet Σ the numbering of the states $q_1 \dots q_{n-l}$ may be different. Following Markov chains notation, with respect to a transition V_{σ_r}, the states $q_{k_r+1} \dots q_{n-l}$ are called transient and the states $q_1 \dots q_{k_r}$ and $q_{n-l+1} \dots q_n$ are called recurrent. The states $q_{n-l+1} \dots q_n$ are called absorbing.

Let us call a matrix of the type above DH-stochastic matrix (even after the states are renumbered).

Theorem 2.1. *A stochastic matrix S is DH-stochastic, iff exists a permutation P such that $PSP^T = \begin{pmatrix} D & 0 & 0 \\ 0 & A & 0 \\ 0 & B & I \end{pmatrix}$, where D is a $k \times k$ doubly stochastic matrix with $k \geq 0$, A - $l \times l$ matrix with $l \geq 0$, I - $m \times m$ unit matrix with $m > 0$, and the sum of elements in any row in the matrices A and B is less or equal than one.*

Proof. Moved to Appendix.

Certainly, the transformation that corresponds to reading of a sequence of letters also is described by a DH-stochastic matrix.

Lemma 2.2. *For any σ_s, σ_t in Σ, $V_{\sigma_s} \cdot V_{\sigma_t}$ is also a DH-stochastic matrix.*

Proof. Moved to Appendix.

Note that the transient states in $V_{\sigma_s} \cdot V_{\sigma_t}$ may be different from the transient states in V_{σ_s} and in V_{σ_t}.

Before looking at forbidden constructions for DH-PRA we need to consider the behavior of the Markov chain induced by the transition V_{σ_r} in the long run.

Lemma 2.3. *Given a DH-stochastic matrix A, there exists K such that $\lim_{n\to\infty} A^{Kn} = \begin{pmatrix} DST' & 0 & 0 \\ 0 & 0 & 0 \\ 0 & a_{ij} & I \end{pmatrix}$, where DST' is a block diagonal $k_r \times k_r$ matrix such that each block is a doubly stochastic matrix with every element equal to $\frac{1}{k_i}$, where k_i is the size of the block.*

Proof. Moved to Appendix.

Definition 2.4. *By $q \xrightarrow{S} q'$, $S \subset \Sigma^*$, we denote that there is a positive probability to get to a state q' by reading a single word $\xi \in S$, starting in a state q.*

3 On a Class of languages recognizable by 1-way DH-PRA

It is easy to see that the class of languages recognized by C-PRA is a proper subclass of languages recognized by DH-PRA. Indeed, by [FGK 04], we may assume that C-PRA does not use any end-markers. Given a C-PRA $\mathcal{A}$, for any final state q_i add an accepting halting state q_i^h, and for any non-final state g_j, add a rejecting halting state g_j^h. Add the final end-marker \$ with transitions $q_i\$ = q_i^h$, $q_i^h\$ = q_i$, $g_j\$ = g_j^h$, $g_j^h\$ = g_j$. $V_\$$ is doubly stochastic. The addition of end-marker ensures that any input word accepted (rejected) by $\mathcal{A}$ is also accepted (rejected) by the DH-PRA with the same probability.

There exist languages recognized by DH-PRA, and not recognized by C-PRA.

Example 3.1. The language a(a,b)* known not to be recognizable by C-PRA is recognizable by DH-PRA.

In this section we will prove that the regular languages whose minimal deterministic automaton contains certain forbidden constructions cannot be recognized by 1-way DH-PRA. We start by definition of these "forbidden" constructions, that are quite similar to the ones defined for C-PRA.

Definition 3.2. *We say that a regular language is of type* 1 *if the following is true for the minimal deterministic automaton recognizing this language: There exist two states* q_1, q_2, *there exist words* x, y *such that*

1. $q_1 \neq q_2$;
2. $q_1x = q_2$, $q_2x = q_2$;
3. $q_2y = q_1$.

Definition 3.3. *We say that a regular language is of type* 2 *if the following is true for the minimal deterministic automaton recognizing this language: There exist three states* q, q_1, q_2, *there exist words* x, y *such that*

1. $q_1 \neq q_2$;
2. $qx = q_1$, $qy = q_2$;
3. $q_1x = q_1$, $q_1y = q_1$;
4. $q_2x = q_2$, $q_2y = q_2$.

The constructions 1 and 2 are forbidden for C-PRA and define block group languages, see [GK 02] and [ABG 04].

Definition 3.4. *We say that a regular language is of Type* 3 *(Figure 3) if a regular language is of Type* 2 *and additional conditions hold for states* q_1, q_2*: There exist 2 words* z_1 *and* z_2 *such that*

1. *reading* z_1 *when in* q_1 *leads to a final state and reading* z_1 *when in* q_2 *leads to a non-final state;*
2. *reading* z_2 *when in* q_2 *leads to a final state and reading* z_2 *when in* q_1 *leads to a non-final state.*

Theorem 3.5. *If a regular language is of Type* 3 *then it is not recognizable by any DH-PRA.*

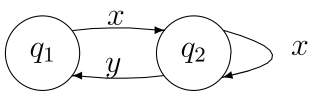


Fig. 1. Type 1 construction

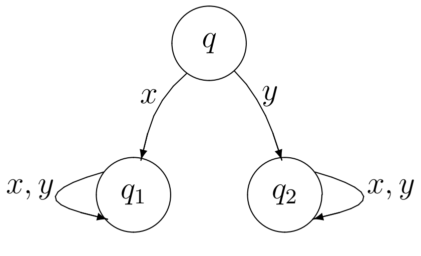


Fig. 2. Type 2 construction

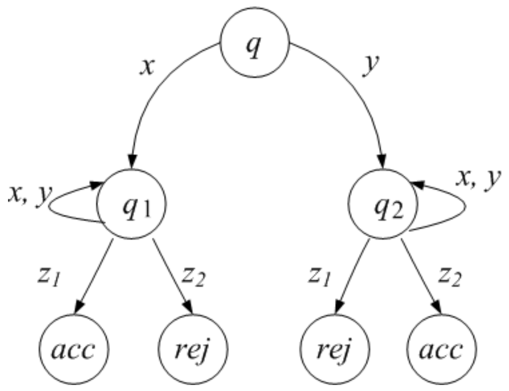


Fig. 3. Type 3 construction

Proof. Assume from the contrary, that A is a DH-PRA automaton which recognizes a language $L \subset \Sigma^*$ of Type 3.

Since L is of Type 3, it is recognized by a minimal deterministic automaton $\mathcal{D}$ with particular three states q, q_1, q_2 such that $q_1 \neq q_2$, $qx = q_1$, $qy = q_2$, $q_1x = q_1$, $q_1y = q_1$, $q_2x = q_2$, $q_2y = q_2$, where $x, y \in \Sigma^*$. Furthermore, there exists $\omega \in \Sigma^*$ such that $q_0\omega = q$, where q_0 is an initial state of $\mathcal{D}$, and there exist words $z_1 \in \Sigma^*$, $z_2 \in \Sigma^*$, such that $q_1z_1 = q^1_{acc}$ and $q_1z_2 = q^1_{rej}$, $q_2z_1 = q^2_{rej}$ and $q_2z_2 = q^2_{acc}$, where q^1_{acc}, q^2_{acc} are final states and q^1_{rej}, q^2_{rej} are non-final states of $\mathcal{D}$.

The transition function of the automaton A is determined by DH-stochastic matrices $V_{\sigma_1}, \ldots, V_{\sigma_n}$. The words from the construction of Type 3 are $x = \sigma_{i_1} \ldots \sigma_{i_k}$ and $y = \sigma_{j_1} \ldots \sigma_{j_s}$. The transitions induced by words x and y are determined by DH-stochastic matrices $X = V_{\sigma_{i_k}} \ldots V_{\sigma_{i_1}}$ and $Y = V_{\sigma_{j_s}} \ldots V_{\sigma_{j_1}}$. Similarly, the transitions induced by words ω, z_1 and z_2 are determined by DH-stochastic matrices W, Z_1 and Z_2.

Let us select 2 words x_1 and x_2 of the form $x_1 = \omega y^K(x^K y^K)^m$ and $x_2 = \omega(x^K y^K)^m$.

We take K to be

- $K > n$, where n is a number of states of the given DH-PRA A;
- K is a multiple of $K_1 * n$ where $(X^{K_1})_{i,i} > 0$ for all non-halting states of A recurrent with respect to X;
- K is a multiple of $K_2 * n$ where $(Y^{K_2})_{i,i} > 0$ for all non-halting states in A recurrent with respect to Y.

We can select such K_1 and K_2 by Corollary A.24. Recurrent states in X and Y in general could be different, by the selection of $K > n$ any transient state in Y^K is also transient in $Y^K X^K$. Indeed, q_i transient with respect to Y^K means some halting state is accessible in 1 step from q_i due to selection of $K > n$. So q_i is either

a) transient with respect to X^K and then some transient state is accessible in 1 step with X^K, thus q_i is transient with respect to $Y^K X^K$, or

b) q_i is recurrent with respect to X^K and then $q_i \xrightarrow{X^K} q_i$, but then q_i transient with respect to $Y^K X^K$. It easy to see that for any transient state q of any DH-stochastic matrix A some absorbing state will be accessible by A^K, $K \geq n$, in 1 step. (As $q \xrightarrow{x^K} q'$ where q' is absorbing state, and $q' \xrightarrow{(xy)^K} q'$.)

There could be some states recurrent for Y^K that are transient states for $Y^K X^K$. However, if q_i and q_j are states recurrent for Y^K and are accessible from each other with Y^K, and q_i is transient for $Y^K X^K$ then q_j is transient for $Y^K X^K$ as well. That holds as we selected K to satisfy the conditions of Lemma A.26 for both X and Y. Indeed, assume from the contrary that q_j is recurrent for $Y^K X^K$. If
a) q_j is transient with respect to X^K then some halting state is accessed in 1 step with respect to X^K and thus q_j is transient with respect to $Y^K X^K$;

b) q_j is recurrent with respect to X^K, then $q_j \xrightarrow{X^K} q_j$ and as $q_j \xrightarrow{Y^K} q_i$ and then again q_j is transient with respect to $Y^K X^K$.

So we get that $\lim\limits_{m\to\infty} (Y^K X^K)^m$ converges to some matrix J of the form described in Lemma 2.3. Consider JY^K. With respect to non-halting recurrent states of $(Y^K X^K)$ the corresponding submatrix of Y^K is a block diagonal matrix of the same block ordering and size as J (although it is possible that some of the blocks consist of smaller blocks). For the transient and halting states there is the same position and size of identity matrix, all the rows corresponding to transient states in J remain 0 rows, but the rows corresponding to halting states are changed (see also Figure 4).

That means that after reading $x_1 = \omega y^K (x^K y^K)^m$ and $x_2 = \omega (x^K y^K)^m$ from the initial state we will get arbitrary close probability distributions for non-halting states, but possibly different probability distributions for the halting states. Let ρ_{a1} be accepting probability and ρ_{r1} rejecting probability after

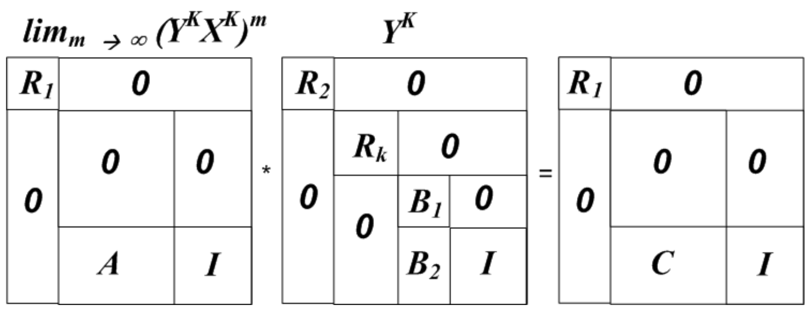


Fig. 4. The structure of the matrices Y^K and $\lim\limits_{m\to\infty} (Y^K X^K)^m$ and $\lim\limits_{m\to\infty} (Y^K X^K)^m \times Y^K$

reading x_1. (So at that moment the automaton remains in a non-halting state with probability $1 - \rho_{a1} - \rho_{r1}$). Let ρ_{a2} be accepting probability and ρ_{r2} rejecting probability after reading x_2. Consider reading z_1 after x_1 that needs to be rejected and x_2 that needs to be accepted. As distributions on non-halting states before reading z_1 are arbitrary close, and reading any word cannot significantly increase their difference, at that moment the word $x_1 z_1 \notin L$ is accepted with probability $\rho_{a1} + c_1$, and the word $x_2 z_1 \in L$ is accepted with probability $\rho_{a2} + c_2$, where c_1 and c_2 are arbitrary close nonnegative values. Due to the assumption that L is recognized with bounded error, we get $\rho_{a1} < \rho_{a2}$. On the other hand, consider reading z_2 after x_1 that needs to be accepted and x_2 that needs to be rejected. In the similar fashion, we get that $\rho_{a2} < \rho_{a1}$. This is a contradiction. □

Theorem 3.6. *If a regular language is of Type* 1*, it is not recognizable by any DH-PRA.*

Proof. The proof is moved to Appendix. It is similar to the proof of Theorem 3.5. Behavior on words $x_1 = \omega(x^K(xy)^K)^m$ and $x_2 = \omega(x^K(xy)^K)^m x^K$ is considered.

4 Closure properties

In this section we prove that the class of languages recognizable by DH-PRA automata is not closed by the union. In [AKV 01] there is proposed a language which is union of languages recognizable by QFA-DH which is not recognizable by QFA-DH, we basically follow their proof.

Theorem 4.1. *There are two languages L_2 and L_3 which are recognizable by DH-PRA, but the union of them $L_1 = L_2 \cup L_3$ is not recognizable by DH-PRA.*

Proof. Let L_1 be the language consisting of all words that start with any number of letters a and after the first letter b (if there is one) there is an odd number of letters a. Its minimal automaton G_1 is shown in Fig. 5.

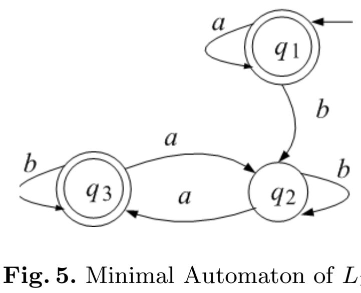


Fig. 5. Minimal Automaton of L_1

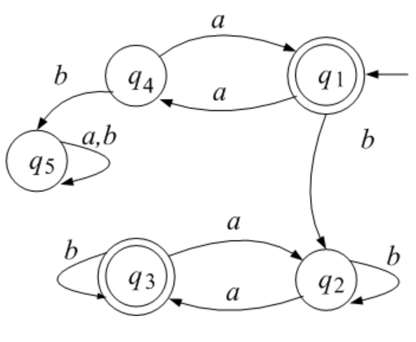


Fig. 6. Minimal Automaton of L_2 "even"

This language satisfies the conditions of Theorem 3.5 (g_1, g_2 and g_3 of Theorem 3.5 are just g_1, g_2 and g_3 of G_1. x, y, z_1, z_2 are b, aba, ab and b.) Hence it cannot be recognized by a DH-PRA. Consider two other languages L_2 and L_3 defined as follows. L_2 consists of all the words which start with an even number of letters a and after the first letter b (if there is one) there is an odd number of letters a. L_3 consists of all the words which start with an odd number of letters a and after the first letter b (if there is one) there is an odd number of letters a. It is easy to see that $L_1 = L_2 \cup L_3$. The minimal automaton for L_2 is shown on Fig. 6 and for L_3 the difference is that initial state is g_4.

We construct two DH-PRA automata K_2 and K_3 which recognize languages G_2 and G_3. The automaton K_2 consists of 12 states: g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9, g_{10}, g_{11} and g_{12}, where $Q_{non} = \{g_1, g_2, g_3, g_4, g_{12}\}$, $Q_{rej} = \{g_5, g_6, g_7, g_8\}$ and $Q_{acc} = \{g_9, g_{10}, g_{11}\}$. The initial state of K_2 is g_{12}. The transition matrices $V_\#$, V_a, V_b and $V_\$$ are defined as follows:

$$V_\# = \begin{pmatrix} \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{2}{3} \\ 0 & \frac{2}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \frac{2}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, V_a = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

$$V_b = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, V_\$ = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

1. After reading the left end-marker # K_2 with probability $\frac{2}{3}$ is in the state g_1 and with probability $\frac{1}{3}$ is in the state g_2. G_2 is in the initial state g_1.
2. After reading even number of letters a K_2 with probability $\frac{2}{3}$ is in the state g_1 and with probability $\frac{1}{3}$ is in the state g_2.
3. After reading odd number of letters a K_2 with probability $\frac{2}{3}$ is in the state g_4 and with probability $\frac{1}{3}$ is in the state g_3.

4. If after reading an odd number of the letter a K_2 receives the letter b or right end-marker then it rejects input with probability at least $\frac{2}{3}$ (from the state g_4 by reading b or right end-marker K_2 goes to rejecting state)
5. If after reading even number of letters a K_2 receives right end-marker then it accepts the input with probability $\frac{2}{3}$
6. If after reading even number of letters a K_2 receives letter b then with probability $\frac{1}{3}$ K_2 passes to accepting state, with probability $\frac{1}{3}$ K_2 passes to rejecting state, and probability $\frac{1}{3}$ K_2 passes to the non-final state g_2
7. By reading the letter a automaton K_2 passes from g_2 to g_3 or back. By reading the letter b automaton K_2 passes from g_2 to g_2 and from g_3 to g_3, so receiving right end-marker in the state g_3 the input is accepted with total probability $\frac{2}{3}$ and receiving right end-marker in the state g_2 the input is rejected with total probability $\frac{2}{3}$.
   This shows that K_2 accepts the language L_2 with probability $\frac{2}{3}$. Similarly we construct K_3 that accepts L_3 with probability $\frac{2}{3}$. Thus we have shown that there are two languages L_2 and L_3 which are recognizable by DH-PRA with probability $\frac{2}{3}$, but the union of them $L_1 = L_2 \cup L_3$ is not recognizable by DH-PRA.

As $L_2 \bigcap L_3 = \emptyset$ then also $L_1 = L_2 \triangle L_3$. So the class of languages recognizable by DH-PRA is not closed under symmetric complement. As this class is closed under complement and not closed under symmetric complement, it easily follows that the class of languages recognizable by DH-PRA is not closed under any binary boolean operation where both arguments are significant.

5 Conclusion

In this paper we continued the research of probabilistic reversible automata (PRA) that were introduced by M. Golovkins and M. Kravtsev in [GK 02]. The first type of PRA - automata with classical acceptance (C-PRA) was completely described in [GK 02] and [ABG 04]. In those papers, it was shown a general class of regular languages, not recognizable by C-PRA, and it was proved that all the other regular languages are recognizable by C-PRA. In this paper we initiated similar research for another type of PRA, i.e., DH-PRA merely defined in [GK 02], that behave more the way measure many quantum automata [KW 97] do, as they halt when entering accepting and rejecting states.

We can see that although DH-PRA can recognize some languages not recognizable by C-PRA the "forbidden constructions" for both of them are very similar. At the same time we see that forbidden constructions for DH-PRA are actually identical to some of those [AKV 01] known for quantum 1-way automata. As we know from [ABG 04] the class of languages recognizable by C-PRA and QFA-C is all the regular languages except the forbidden constructions exposed in [GK 02]. As languages recognizable by C-PRA are also recognizable by DH-PRA, there is one open problem which still remains: whether the set of languages that are of Type 2 and not of Type 3 is recognizable by DH-PRA. The same holds for the MM-QFA (by results of [AKV 01] and [ABG 04]). This fact and similar closure properties give us an opportunity to speculate that possibly DH-PRA and MM-QFA actually recognize the same class of languages, or, which is even more credible, that DH-PRA is at least as powerful as 1-way quantum automata.

References

[ABG 04] A. Ambainis, M. Beaudry, M. Golovkins, A. Ķikusts, M. Mercer, D. Thérien. Algebraic Results on Quantum Automata. *STACS 2004, Lecture Notes in Computer Science*, 2996:93–104, 2004.

[AKV 01] A. Ambainis, A. Ķikusts, M. Valdats. On the Class of Languages Recognizable by 1-Way Quantum Finite Automata. *STACS 2001, Lecture Notes in Computer Science*, 2010:75–86, 2001.

[BMP 03] Alberto Bertoni, Carlo Mereghetti, and Beatrice Palano. Quantum Computing: 1-Way Quantum Automata, *DLT 2003, Lecture Notes in Computer Science*, Vol. 2710, pp. 120, 2003.

[BP 99] A. Brodsky, N. Pippenger. Characterizations of 1-Way Quantum Finite Automata. http://arxiv.org/abs/quant-ph/9903014

[FGK 04] R. Freivalds, M. Golovkins, A. Ķikusts. On the Properties of Probabilistic Reversible Automata. *SOFSEM 2004, Proceedings Vol. 2*, pp. 75-84, MatFyzPress, Prague, 2004. Also see the Appendix of http://arxiv.org/abs/cs/0209018.

[GK 02] M. Golovkins, M. Kravtsev. Probabilistic Reversible Automata and Quantum Automata. *COCOON, Lecture Notes in Computer Science*, 2378:574–583, 2002.

[KW 97] A. Kondacs, J. Watrous. On The Power of Quantum Finite State Automata. *Proc. 38th FOCS*, pages 66–75, 1997.

[MC 00] C. Moore, J. P. Crutchfield. Quantum automata and quantum grammars. *Theoretical Computer Science*, 2000, Vol. 237(1-2), pp. 275-306. http://arxiv.org/abs/quant-ph/9707031

[N 99] A. Nayak. Optimal Lower Bounds for Quantum Automata and Random Access Codes. *Proc. 40th FOCS*, pages 369–377, 1999.

[R 63] M. O. Rabin. Probabilistic Automata. *Information and Control*, 1963, Vol. 6(3), pp. 230-245.

[T 68] G. Thierrin. Permutation Automata. *Mathematical Systems Theory*, Vol. 2(1), pp. 83-90.

A Appendix

In the following definitions computation step corresponds to reading a single input character.

A.1 Definition of acceptance

Definition A.1. "Decide-and-halt" acceptance. *Consider an automaton with the set of configurations partitioned into non-halting configurations and halting configurations, where halting configurations are further classified as accepting configurations and rejecting configurations. We say that an automaton accepts (rejects) an input in a decide-and-halt manner, if the following conditions hold:*

- *the computation is halted as soon as the automaton enters a halting configuration;*
- *if the automaton enters an accepting configuration, the input is accepted;*
- *if the automaton enters a rejecting configuration, the input is rejected.*

We refer to the decide-and-halt automata as DH-automata. In case of real-time automata, we may use the following definition.

Definition A.2. Classical acceptance. *Consider an automaton with the set of configurations partitioned into accepting configurations and rejecting configurations. We say that an automaton accepts (rejects) an input classically, if the following conditions hold:*

- *the computation is halted as soon as the number of computation steps is equal to the length of input;*
- *if the automaton has entered an accepting configuration when halted, the input is accepted;*
- *if the automaton has entered a rejecting configuration when halted, the input is rejected.*

We refer to the classical acceptance automata as classical automata or C-automata.

Having defined word acceptance, we define language recognition in an equivalent way as in [R 63].

By $p_{x,A}$ we denote the probability that an input x is accepted by an automaton A.

Furthermore, we denote $P_L = \{p_{x,A} \mid x \in L\}$, $\overline{P_L} = \{p_{x,A} \mid x \notin L\}$, $p_1 = \sup \overline{P_L}$, $p_2 = \inf P_L$.

Definition A.3. *We say that an automaton A recognizes a language L with interval (p_1, p_2), if $p_1 \leq p_2$ and $P_L \cap \overline{P_L} = \emptyset$.*

Definition A.4. *We say that an automaton A recognizes a language L with bounded error and interval (p_1, p_2), if $p_1 < p_2$.*

We consider only bounded error language recognition.

A.2 Probabilistic reversible automaton

Definition A.5. *A probabilistic automaton is called* reversible *if its evolution is described by a doubly stochastic matrix, using canonical basis.*

Definition A.6. *1-way probabilistic reversible automaton (PRA)* $A = (Q, \Sigma, q_0, \delta)$ *is specified by a finite set of states* Q*, a finite input alphabet* Σ*, an initial state* $q_0 \in Q$*, and a transition function*

$$\delta : Q \times \Gamma \times Q \longrightarrow \mathbb{R}_{[0,1]},$$

where $\Gamma = \Sigma \cup \{\$, \#\}$ *is the input tape alphabet of* A *and* $\$$*,* $\#$ *are end-markers not in* Σ*. Furthermore, transition function satisfies the following requirements:*

$$\forall (q_1, \sigma_1) \in Q \times \Gamma \ \sum_{q \in Q} \delta(q_1, \sigma_1, q) = 1 \tag{1}$$

$$\forall (q_1, \sigma_1) \in Q \times \Gamma \ \sum_{q \in Q} \delta(q, \sigma_1, q_1) = 1 \tag{2}$$

For every input symbol $\sigma \in \Gamma$, the transition function may be determined by a $|Q| \times |Q|$ matrix V_σ, where $(V_\sigma)_{i,j} = \delta(q_j, \sigma, q_i)$.

Lemma A.7. *The probabilistic automaton is reversible iff all matrices* V_σ *are doubly stochastic.*

Proof. Trivial.

A.3 Markov chains

We recall definitions and theorems from the theory of finite Markov chains.

Definition A.8. *A state* q_j *is accessible from* q_i *(denoted* $q_i \to q_j$*) if there is a positive probability to get from* q_i *to* q_j *in 1 or more steps.*

Note that some authors consider zero steps are valid for this definition, that means $q_i \to q_i$ for any i.

Definition A.9. *States* q_i *and* q_j *communicate (denoted* $q_i \leftrightarrow q_j$*) if* $q_i \to q_j$ *and* $q_j \to q_i$*.*

For accessibility or communication in one step we will put the corresponding matrix above the symbol. Example: $q_i \xrightarrow{A} q_j$ means there is a positive probability to get from q_i to q_j by performing transformation A. Or the same, $A_{j,i} > 0$.

Definition A.10. *A state* q *is called recurrent if* $\forall i \ q \to q_i \Rightarrow q_i \to q$*. Otherwise the state is called transient.*

There are several different definitions for transient states proven to be equivalent to the above, important for us is

Definition A.11. *A state q_i is called transient iff* $\sum\limits_{n\to\infty} (A^n)_{i,i} < \infty$.

Definition A.12. *A state q is called absorbing if there is a zero probability of exiting from this state.*

Definition A.13. *A Markov chain without transient states is called irreducible if for all q_i, q_j $q_i \leftrightarrow q_j$. Otherwise the chain without transient states is called reducible.*

Definition A.14. *The period of a recurrent state $q_i \in Q$ of a Markov chain with a matrix A is defined as $d(q_i) = \gcd\{n > 0 \mid (A^n)_{i,i} > 0\}$.*

Definition A.15. *A recurrent state q_i is called aperiodic if $d(q_i) = 1$. Otherwise the recurrent state is called periodic.*

Definition A.16. *A Markov chain without transient states is called aperiodic if all of its states are aperiodic. Otherwise the chain without transient states is called periodic.*

Definition A.17. *Markov chain is called absorbing iff it contains at least one absorbing state, and for any non-absorbing state q_i there is an absorbing state that is accessible from q_i. Thus the states of absorbing Markov Chain can be numbered so that transition matrix A has a form*

$$\begin{pmatrix} A & O \\ B & I \end{pmatrix},$$

where I - unit matrix, O - all zero matrix.

Definition A.18. *A probability distribution X of a Markov chain with a matrix A is called stationary, if $AX = X$.*

We recall the following theorem from the theory of finite Markov chains:

Theorem A.19. *If a Markov chain with a matrix A is irreducible and aperiodic, then*
a) it has a unique stationary distribution Z;
b) $\lim\limits_{n\to\infty} A^n = (Z, \ldots, Z)$;
c) $\forall X \lim\limits_{n\to\infty} A^n X = Z$.

We recall the following fact regarding transient states of a Markov chain:

Theorem A.20. *Given a Markov chain with a matrix A and a transient state q_i, for matrix A^n, for any j, if $n \to \infty$, $a^n_{ij} \to 0$.*

A.4 Doubly Stochastic Markov chain

We recall the following definitions and facts from [GK 02].

Definition A.21. *A Markov chain is called doubly stochastic, if its transition matrix is a doubly stochastic matrix.*

Corollary A.22. *If a doubly stochastic Markov chain with an $m \times m$ matrix A is irreducible and aperiodic,*

a) $\lim\limits_{n\to\infty} A^n = \begin{pmatrix} \frac{1}{m} \cdots \frac{1}{m} \\ \cdots\cdots\cdots \\ \frac{1}{m} \cdots \frac{1}{m} \end{pmatrix}$;

b) $\forall X \ \lim\limits_{n\to\infty} A^n X = \begin{pmatrix} \frac{1}{m} \\ \cdots \\ \frac{1}{m} \end{pmatrix}$.

Proof. By Theorem A.19.

Lemma A.23. *If M is a doubly stochastic Markov chain with a matrix A, then $\forall q \ q \to q$.*

Corollary A.24. *Suppose A is a doubly stochastic matrix. Then exists $k > 0$, such that $\forall i \ (A^k)_{i,i} > 0$.*

Lemma A.25. *If M is a doubly stochastic Markov chain with a matrix A, then $\forall q_a, q_b \ A_{b,a} > 0 \Rightarrow q_b \to q_a$.*

A.5 Working lemmas for Doubly stochastic Markov chains

These simple facts are used in the proofs of the paper.

Lemma A.26. *Suppose A is a doubly stochastic matrix and $k > 0$, such that $\forall i \ (A^k)_{i,i} > 0$. Then there exist $m > 0$ such that for all pairs q_i, q_j, if $q_i \to q_j$ for A^k, then $q_i \to q_j$ in one step for A^{km}.*

Proof. Assume $q_i \to q_j$ in x steps. Since by Lemma A.24, if $q_i \to q_i$ in one step, then $q_i \to q_j$ in $x+1$ step as well. For any pair of states q_i, q_j, where $q_i \to q_j$ for A^k, q_j is accessible in less than n steps, where n is a number of rows in A (i.e., the number of elements in the underlying Markov chain). Thus $m = n$ gives the necessary constant.

Lemma A.27. *Accessibility is a class property for states of doubly stochastic Markov chains.*

Proof.
- reflexive - $\forall i \ q_i \to q_i$ by Lemma A.23;
- symmetric - If $q_i \to q_j$ then $q_j \to q_i$ by Lemma A.25;
- transitive - If $q_i \to q_j$ and $q_j \to q_k$ then $q_i \to q_k$.

A.6 DH-stochastic matrices

In order to prove Theorem 2.1, we first formulate the following lemma:

Lemma A.28. *Let A a $k \times m$ matrix such that $m < k$, where the sum of elements in any column is one, and the sum of elements in any row is less or equal than one. Then exists a $k \times (k-m)$ matrix B, such that $(A\ B)$ is doubly stochastic.*

Proof. Let s_i the sum of elements of the i-th row of A. Let $B = \begin{pmatrix} \frac{1-s_1}{k-m} & \cdots & \frac{1-s_1}{k-m} \\ \dots & \dots & \dots \\ \frac{1-s_k}{k-m} & \cdots & \frac{1-s_k}{k-m} \end{pmatrix}$.

Now the sum of elements in any column of B is $\frac{k-\sum_{i=1}^{k} s_i}{k-m} = 1$. Hence $(A\ B)$ is stochastic. The sum of the i-th row of $(A\ B)$ is $s_i + (k-m)\frac{1-s_i}{k-m} = 1$. Hence $(A\ B)$ is doubly stochastic. □

Proof of Theorem 2.1. To prove the theorem, it is sufficient to show that any matrix of the form $\begin{pmatrix} D & 0 & 0 \\ 0 & A & 0 \\ 0 & B & I \end{pmatrix}$, specified in the theorem, can be obtained from a doubly stochastic matrix. This indeed holds, since by Lemma A.28, the matrix $\begin{pmatrix} D & 0 \\ 0 & A \\ 0 & B \end{pmatrix}$ can be complemented with new columns to obtain a doubly stochastic matrix. □

Proof of Lemma 2.2. Follows from matrix manipulation. To show that for the states from which the halting states are not accessible with $\sigma_t \sigma_s$ the matrix is doubly stochastic, observe that no sum in the row can exceed one and also cannot be less than one as otherwise summing by rows and columns would give different results. □

Proof of Lemma 2.3. Follows if taken K such that $A_{i,i}^{K} > 0$ for all recurrent states (possible by Lemma A.23). Rows filled entirely by zeros correspond to transient states. As non-halting recurrent states in A^K form doubly stochastic matrix then they can be split into equivalence classes with respect to communication property (see Lemma A.27) and each block diagonal submatrix corresponds to states in one equivalence classes. The values in these submatrixes are determined by Corollary A.22. □

A.7 Proof of Theorem 3.6

Proof. Assume from the contrary, that A is a DH-PRA automaton which recognizes a language $L \subset \Sigma^*$ of Type 1.

Since L is of Type 1, it is recognized by a deterministic automaton $\mathcal{D}$ which has two states q_1, q_2 such that $q_1 \neq q_2$, $q_1 x = q_2$, $q_2 y = q_1$, $q_2 x = q_2$ where

$x, y \in \Sigma^*$. Furthermore, exists $\omega \in \Sigma^*$ such that $q_0\omega = q_1$, where q_0 is an initial state of $\mathcal{D}$, and exists a word $z \in \Sigma^*$, such that $q_1 z = q_{acc}$ if and only if $q_2 z = q_{rej}$, where q_{acc} is an accepting state and q_{rej} is a rejecting state of $\mathcal{D}$.

The transition function of the automaton A is determined by DH-stochastic matrices $V_{\sigma_1}, \ldots, V_{\sigma_n}$. The words $x = \sigma_{i_1} \ldots \sigma_{i_k}$ and $y = \sigma_{j_1} \ldots \sigma_{j_s}$, the transitions induced by words x and y are determined by DH-stochastic matrices $X = V_{\sigma_{i_k}} \ldots V_{\sigma_{i_1}}$ and $Y = V_{\sigma_{j_s}} \ldots V_{\sigma_{j_1}}$. Similarly, the transitions induced by word ω is determined by DH-stochastic matrix W.

Let us select 2 words x_1 and x_2 of the form $x_1 = \omega(x^K(xy)^K)^m$ and $x_2 = \omega(x^K(xy)^K)^m x^K$.

We will show that for any ε we can select K and m such that $|p_{x_1} - p_{x_2}| < \varepsilon$. Then as $x_1 z \in L$ and $x_2 z \notin L$ we get a contradiction.

We take K to be

- $K > n$, where n is a number of states of the given DH-PRA A;
- K is a multiple of $K_1 * n$ such that $(X^{K_1})_{i,i} > 0$ for all non-halting states of A recurrent in respect to X;
- K is a multiple of $K_2 * n$ such that $((YX)^{K_2})_{i,i} > 0$ for all non halting states of A recurrent in respect to YX.

We can select such K_1 and K_2 by Corollary A.24. We should note however that recurrent states in X and YX in general could be different! Given $K > n$ we get that any transient state for X^K is also transient state for $(YX)^K X^K$. As for any transient state q of any DH stochastic matrix A some absorbing state will be accessible by A^K $K \geq n$ in 1 step, and if q' is absorbing state that $q \xrightarrow{x^K} q'$, and $q' \xrightarrow{(xy)^K} q'$.

But there could be some states recurrent for X^K that are transient states for $(YX)^K X^K$. However if q_i and q_j are states recurrent for X^K and $q_i \leftrightarrow q_j$ for X^K and q_i is transient for $(YX)^K X^K$ then q_j is transient for $(YX)^K X^K$ as well.

q_i transient in respect to $(YX)^K X^K$ means there is some sequence of letters starting with x^K that leads to the absorbing state from q_i but not from q_j. But that is contradiction as for any $q\prime$ $q_i \xrightarrow{X^K} q'$ will also hold $q_j \xrightarrow{X^K} q'$ as we selected K to satisfy conditions of Lemma A.26.

So for $\lim\limits_{m \to \infty}$ we get $((YX)^K X^K)^m$ converges to some matrix J of the form described in Lemma 2.3. $X^K J = J$ follows from matrix multiplication rules: X^K in respect to non-halting recurrent states of $((YX)^K X^K)$ is a block diagonal matrix of the same block ordering and size (although it is possible that some of blocks consist of smaller blocks), but for transient and halting states there is the same position and size of identity matrix and all the rows corresponding to transient states in J are 0 rows (see also Figure 7).

That means that after reading $x_1 = \omega(x^K(xy)^K)^m$ and $x_2 = \omega(x^K(xy)^K)^m x^K$ we will get arbitrary close probability distributions that gives us required contradiction. Or formally,

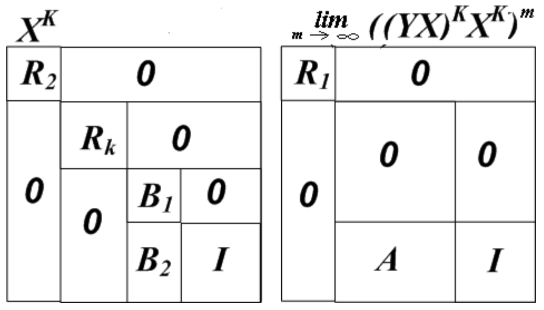


Fig. 7. The structure of the matrices X^K and $\lim_{m\to\infty}((YX)^K X^K)^m$

$\lim_{m\to\infty} ZX^K((YX)^K X^K)^m W = \lim_{m\to\infty} Z((YX)^K X^K)^m W = ZJW$. So

$$\forall\varepsilon > 0\ \exists m\ \left\|\Big(Z(X^K((YX)^K X^K)^m W - Z((YX)^K X^K)^m W\Big)Q_0\right\| < \varepsilon. \qquad (3)$$

As we can select z such that $\omega(x^k(xy)^k)^m x^K z \in L$ and $\omega(x^k(xy)^k))^m z \notin L$, that requires existence of $\varepsilon > 0$, such that

$$\forall m\ \left\|\Big(ZX^K((YX)^K X^K)^m W - Z((YX)^K X^K)^m W\Big)Q_0\right\| > \varepsilon. \qquad (4)$$

□

Nondeterministic quantum query with minimal complexity

Lelde Lāce[⋆]

Institute of Mathematics and Computer Science University of Latvia
Raiņa bulvaris 29, LV-1459, Riga, Latvia

Abstract. We study nondeterministic quantum algorithms [4] for Boolean functions f. Such algorithms have positive acceptance probability on input x iff f(x) = 1. We construct some nondeterministic quantum query algorithms with complexity 1 for Boolean functions with 2, 4 and 2n variables and study some properties of these functions.

1 INTRODUCTION

Recently it has become clear that a quantum computer could, in principle, solve certain problems faster than a conventional computer. A quantum computer is a device, which takes full advantage of quantum mechanical superposition and interference.

Boolean decision trees model is the most simple model to compute Boolean functions. In this model the primitive operation made by an algorithm is evaluating an input Boolean variable. The cost of a (deterministic) algorithm is the number of variables it evaluates on a worst case input. It is easy to find the deterministic complexity of all explicit Boolean functions (for most functions it is equal to the number of variables).

The *black-box* model of computation arises when one is given a black-box containing an N-tuple of Boolean variables $X = (x_0, x_1, ..., x_{N-1}.)$. The box is equipped to output x_i on input i. We wish to determine some property of X, accessing the x_i only through the black box. Such a black-box access is called a *query*. A property of X is any Boolean function that depends on X, i.e. a property is function $f : \{0,1\}^N \to \{0,1\}$. We want to compute such properties using as few queries as possible.

In classical computing, nondeterministic computation has a prominent place in many different models and for many good reasons. For example, in Turing machine complexity, the study of nondeterminism leads naturally to the class of NP-complete problems, which contains some of the most important and practically relevant computer science problems- as well as some of the hardest theoretical open questions.

In [4] R. de Wolf define nondeterministic quantum algorithm: a nondeterministic quantum algorithm for f is defined to be a quantum algorithm that outputs 1 with positive probability if $f(x) = 1$ and that always outputs 0 if $f(x) = 0$.

⋆ Research supported by The European Social Fund (ESF)

We construct some nondeterministic quantum query algorithms with complexity 1 for Boolean functions with 2, 4 and 2n variables and study some properties of these functions.

2 DEFINITIONS

2.1 Quantum computing

We introduce the basic model of quantum computing. For more details, see textbooks by Gruska [2] and Nielsen and Chuang [3].

Quantum states: We consider finite dimensional quantum systems. An n-dimensional pure state is a vector $|\psi\rangle \in C^n$ of norm 1. Let $|1\rangle, |2\rangle, \dots, |n-1\rangle$ be an orthonormal basis for C^n. Then, any state can be expressed as $|\psi\rangle = \sum_{i=0}^{n-1} a_i|i\rangle$ for some $a_0 \in C, a_1 \in C, \dots, a_{n-1} \in C$. Since the norm of $|\psi\rangle$ is 1, $|a_i|^2 = 1$. We call the states $|1\rangle, |2\rangle, \dots, |n-1\rangle$ *basic states*. Any state of the form $\sum_{i=0}^{n-1} a_i|i\rangle$ is called a *superposition* of $|1\rangle, |2\rangle, \dots, |n-1\rangle$. The coefficient a_i is called *amplitude* of $|i\rangle$.

A quantum system can undergo two basic operations: an unitary evolution and a measurement.

Unitary evolution: A *unitary transformation* U is a linear transformation on C^k that preserves the l_2 norm (i.e., maps vectors of unit norm to vectors of unit norm). If, before applying U, the system was in a state $|\psi\rangle$, then the state after the transformation is $U|\psi\rangle$.

Measurements: In this survey, we just use the simplest case of quantum measurement. It is the full measurement in the computation basis. Performing this measurement on a state $|\psi\rangle = a_1|0\rangle + \dots a_k|k\rangle$ gives the outcome i with probability $|a_i|^2$. The measurement changes the state of the system to $|i\rangle$. Notice that the measurement destroys the original state $|\psi\rangle$ and repeating the measurement gives the same i with probability 1 (because the state after the first measurement is $|i\rangle$. More general classes of measurements are general von Neumann and POVM measurements [3].

2.2 Query model

In the query model, the input $x_1, \dots, x_N$ is contained in a black box and can be accessed by queries to the black box. In each query, we give i to the black box and the black box outputs x_i. The goal is to solve the problem with the minimum number of queries.

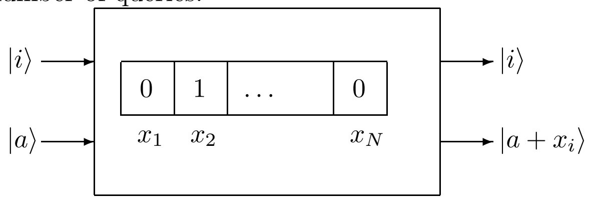


Fig. 1 *Quantum black box.*

There are two ways how to define the query box in the quantum model. The first is the extension of the classical query (Figure 1). It has two inputs: i, consisting of [logN] bits and b consisting of 1 bit. If the input to the query box is a basic state $|i\rangle|b\rangle$, the output is $|i\rangle|b \oplus x_i\rangle$. If the input is a superposition $\sum_{i,b} a_{i,b}|i\rangle|b\rangle$, the output is $\sum_{i,b} a_{i,b}|i\rangle|b \oplus x_i\rangle$. Notice that this definition applies both to case when x_i are binary and to the case when they are k-valued. In the k-valued case, we just make b to consist of $\lceil log_2 k \rceil$ bits and take $b \oplus x_i$ to be bitwise XOR of b and x_i.

In the second form of quantum query (which only applies to problems with $\{0, 1\}$-valued x_i), the black box has just one input i. If the input is a state $\sum_i a_i|i\rangle$, the output is $\sum_i (-1)^{x_i} a_i|i\rangle$. While this form is less intuitive, it is very convenient for the use in quantum algorithms, including Grover's search algorithm [1]. A query of second type can be simulated by a query of first type [1].

A quantum query algorithm with T queries is just a sequence of unitary transformations

$$U_0 \to O \to U_1 \to O \to \ldots \to U_{T-1} \to O \to U_T$$

on some finite- dimensional space C^k. $U_0, U_1, \ldots, U_T$ can be any unitary transformations that do not depend on the bits $x_1, \ldots, x_N$ inside the black box. O are query transformations that consist of applying the query box to the first logN+1 bits of the state. That is, we represent basic states of C^k as $|i, b, z\rangle$. Then, O maps $|i, b, z\rangle$ to $|i, b \oplus x_i, z\rangle$. We use O_x to denote the query transformation corresponding to an input $x = (x_1, \ldots x_N)$.

The computation starts with state $|0\rangle$. Then, we apply $U_0, O_x, \ldots, O_x, U_T$ and measure the final state. The result of the computation is the rightmost bit of the state obtained by the measurement (or several bits if we are considering a problem where the answer has more than 2 values).

The quantum algorithm computes a function $f(x_1, \ldots, x_N)$ if, for every $x = (x_1, \ldots, x_N)$ for which f is defined, the probability that the rightmost bit of $U_T O_x U_{T-1} \ldots O_x U_0|0\rangle$ equals $f(x_1, \ldots, x_N)$ is at least $1 - \epsilon < \frac{1}{2}$.

3 MAIN RESULTS

3.1 Boolean functions with 2 variables

We have only four nontrivial Boolean functions with 2 variables.- OR, AND, PARITY and ¬ PARITY. There is well known exact algorithm for Boolean function PARITY in figure 2. Each exact quantum algorithm satisfies conditions of nondeterministic quantum. From this follows that we have nondeterministic quantum algorithm computing PARITY with one query. Function ¬ PARITY we can compute by using the same algorithm we only need to switch function output values.

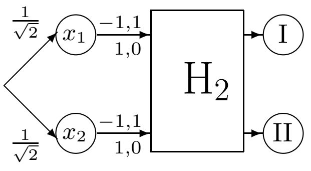


Fig. 2 *Algorithm for function PARTY .*

If we look at OR function - we do not have exact quantum algorithm for OR function with one query. But we know that $NQ(f) = ndeg(f)$. For example, $p(x_1, x_2) = x_1 - x_2$ is a degree-1 nondeterministic polynomial for PARITY with two variables: it assumes value 0 on x-weights 0 and 2 and +/-1 on weight 1. But $p(x_1, x_2) = x_1 + x_2$ is a degree-1 nondeterministic polynomial for OR : it assumes value 0 on x-weights 0 and 1 and 2 on weight 1 and 2. There is nondeterministic quantum algorithm for OR in figure 3 . Output I has value 1 other outputs -0.

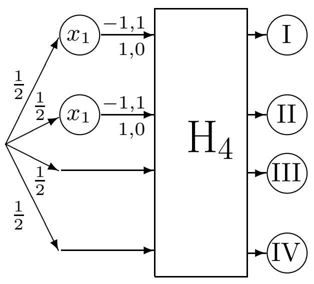


Fig. 3 *Algorithm for function OR .*

It is easy to see that nondeterministic query algorithms are not symmetric. If function value is 0 we have stronger condition compared to function value 1. We suppose that function has more 0 values, then nondeterministic complexity of this function is bigger. $p(x_1, x_2) = x_1 * x_2$ is a degree-2 nondeterministic polynomial for AND. In this case $ndeg(f) = def(f)$.

3.2 Boolean functions with 4 variables

We can use previous ideas (PARITY and ¬ PARITY) to make some nondeterministic quantum algorithms with four variables.

Theorem 1. *If there are nondeterministic quantum query algorithms for functions $F_1(x_1, ...x_k)$ and $F_2(x_1, ...x_m)$ with complexities q_1 and q_2 then nondeterministic quantum query algorithm exists for function $F(x_1, ...x_{k+m}) = F_1(x_1, ... x_k)$ OR $F_2(x_{k+1}, ...x_{k+m})$ with complexity $q = max(q_1, q_2)$.*

Proof. There is nondeterministic quantum query algorithm for function F shown in figure 4. This algorithm gives value of function 0 if $F_1(x_1, ...x_k)$ and $F_2(x_1, ... x_m)$ have values 0. If one of values or both are 1 then function value is 1. In both algorithms A1 and A2 values of output states remain.

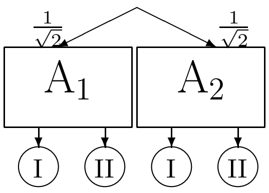


Fig. 4 *Algorithm for function F .*

If we use nondeterministic quantum query algorithms for functions PARITY and ¬PARITY we can make 6 different functions with four variables. Zero values of those functions are shown in table 1.

Table 1. Zero values of functions $D_1, \ldots D_6$

D_1	D_2	D_3	D_4	D_5	D_6
$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 0	0 1 0 1
0 0 1 1	1 0 0 1	0 1 0 1	0 0 1 0	1 0 0 0	1 0 0 1
1 1 0 0	0 1 1 0	1 0 1 0	1 1 0 1	0 1 1 1	0 1 1 0
1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 0	1 0 1 1	1 0 1 0

The corresponding nondeterministic polynomials from these functions:

$$\begin{array}{ll} D_1 & x_1 - x_2 + 2x_3 - 2x_4 \\ D_2 & x_1 + 2x_2 - 2x_3 - x_4 \\ D_3 & x_1 + 2x_2 - x_3 - 2x_4 \\ D_4 & 2x_1 - 2x_2 + x_3 + x_4 - 1 \\ D_5 & x_1 + x_2 + 2x_3 - 2x_4 - 1 \\ D_6 & x_1 + x_2 + 2x_3 + 2x_4 - 3 \end{array}$$

It is easy to see that polynomials for functions D_1, D_2 and D_3 are possible to obtain one from another by changing variables. Functions D_4 and D_5 have the same property. Function D_2 has value 0 if input vector is symmetric.

$$D_1(x_1, x_2, x_3, x_4) = D_2(x_2, x_3, x_4, x_1) = D_3(x_1, x_3, x_2, x_4)$$
$$D_4(x_1, x_2, x_3, x_4) = D_5(x_3, x_4, x_1, x_2)$$

For all i $D_i(x_1, x_2, x_3, x_4) = D_i(1 - x_1, 1 - x_2, 1 - x_3, 1 - x_4)$. This property is deduced from the fact that all functions are based on functions PARITY and ¬PARITY . In figure 5 is shown another algorithm which computes some Boolean function. Values of functions depend on values of output states.

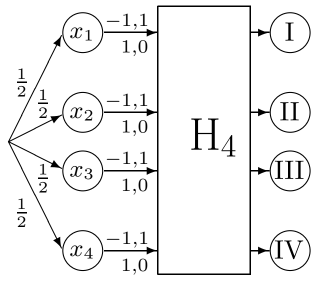


Fig. 5 *Quantum query algorithm .*

Each output state of this algorithm can recognize two input vectors $(x_1x_2x_3 x_4)$ and $(1-x_11-x_21-x_31-x_4)$. For example output state I recognizes (0 0 0 0) and (1 1 1 1) but III - (0 0 1 1) and (1 1 0 0). States II, III and IV are mutually symmetric they can be obtained one from another by changing variables. If we choose two output states then we can recognize four input vectors. Maximum number of output states with the same value is three. If we put one value to all output states we obtain constant function.

There are zero values of functions $E_1, \dots E_6$ shown in table 2. Nondeterministic quantum algorithms of these functions are obtained by assigning values of zero to two output states of previous algorithm .

Table 2. Zero values of functions $E_1, \dots E_6$

E_1(I,II)	E_2(I,III)	E_3(I,IV)	E_4(II,III)	E_5(II,IV)	E_6(II,IV)
$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$
1 1 1 1	1 1 1 1	1 1 1 1	0 1 0 1	0 1 0 1	0 0 1 1
0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0	1 0 1 0	1 1 0 0
0 1 0 1	0 0 1 1	0 1 1 0	0 0 1 1	0 1 1 0	0 1 1 0
1 0 1 0	1 1 0 0	1 0 0 1	1 1 0 0	1 0 0 1	1 0 0 1

It is easy to see that $E_1 = D_3$, $E_2 = D_1$, $E_3 = D_2$ and $E_5 = D_6$ and $E_4(x_1, x_2, x_3, x_4) = E_5(x_1, x_4, x_2, x_3) = E_6(x_1, x_2, x_4, x_3)$.

The corresponding degree-1 nondeterministic polynomials from these functions are:

$$\begin{array}{ll} E_4 & x_1 + 2x_2 + 2x_3 + x_4 - 3 \\ E5 & x_1 + x_2 + 2x_3 + 2x_4 - 3 \\ E6 & x_1 + 2x_2 + x_3 + 2x_4 - 3 \end{array}$$

Zero values of functions $F_1, \dots F_4$ are shown in table 3. Nondeterministic quantum algorithms of these functions are obtained by assigning values of zero to three output states of previous algorithm.

Table 3. Zero values of functions $F_1, \dots F_4$

F_1(I,I,IIII)	F_2(I,II,IV)	F_3(I,II,IV)	F_4(II,III,IV)
$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$
1 1 1 1	1 1 1 1	1 1 1 1	0 1 0 1
0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0
0 1 0 1	0 1 0 1	0 0 1 1	0 0 1 1
1 0 1 0	1 0 1 0	1 1 0 0	1 1 0 0
0 0 1 1	0 1 1 0	0 1 1 0	0 1 1 0
1 1 0 0	1 0 0 1	1 0 0 1	1 0 0 1

$F_1(x_1, x_2, x_3, x_4) = F_2(x_1, x_4, x_3, x_2) = F_3(x_1, x_2, x_4, x_3)$.

The corresponding degree-1 nondeterministic polynomials from these functions are:

F_1	$x_1 - x_2 - x_3 + x_4$
F_2	$x_1 + x_2 - x_3 - x_4$
F_3	$x_1 - x_2 + x_3 - x_4$
F_4	$x_1 + x_2 + x_3 + x_4 - 2$

Theorem 2. *Nondeterministic quantum query algorithm with one query is not able to recognize nonconstant function with $F(x) = F(y) = 0$ and $||x| - |y|| = 1$.*

Proof. If we can recognize function with nondeterministic one query algorithm then nondeterministic degree-1 polynomial exists for this function. Assume that x =(0,0,0,0) and y= (1,0,0,0), (0,1,0,0), (0,0,1,0) and (0,0,0,1). Polynomial of function can be written as $c_0 + c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4$. From F(0,0,0,0)=0 follows $c_0 = 0$. From F(y)=0 follows $c_1 = 1, c_2 = 0, c_3 = 0$ and $c_4 = 0$. We obtain constant function.

All previous examples have maximum six zero input vectors. Question arises if this is a maximum possible value. From theorem 2 follows that two potential groups of input values are:

I.(0000) (1100) (1010) (1001) (0110) (0101) (0011) (1111)

II. (1000) (0100) (0010) (0001) (1110) (1101) (1011) (0111)

Function polynomial is $c_0 + c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4$.

I. $c_0 = 0$ from first input vector. $c_1 + c_2 = 0$ (2), $c_1 + c_3 = 0$ (3), $c_1 + c_4 = 0$ (4), $c_2 + c_3 = 0$ (5), $c_2 + c_4 = 0$ (6), $c_3 + c_4 = 0$ (7), $c_1 + c_2 + c_3 + c_4 = 0$ (8)

From 2,3,4 follows $c_2 = c_3 = c_4 = -c_1$
From 5 and 6 follows $c_2 = c_3 = c_4 = 0$ and $c_1 = 0$
And we get constant function.

II. $c_0+c_1 = 0$ (1) $c_0+c_2 = 0$ (2) $c_0+c_3 = 0$ (3) $c_0+c_4 = 0$ (4) $c_0+c_1+c_2+c_3 = 0$ (5) $c_0 + c_1 + c_2 + c_4 = 0$ (6) $c_0 + c_1 + c_3 + c_4 = 0$ (7) $c_0 + c_2 + c_3 + c_4 = 0$ (8)
From 1,2,3, and 4 follows $c_1 = c_2 = c_3 = c_4 = -c_0$
5 and 6 are correct only if $c_0 = c_1 = c_2 = c_3 = c_4 = 0$.
We again get constant function.

3.3 Boolean functions with 2n variables

In this paragraph we show some ideas how to make nondeterministic quantum query algorithm with one query.

If we use algorithms of PARITY un $\neg$ PARITY functions we can get 2^n different nondeterministic quantum query algorithms. Each of these algorithms have 2^{2n} input vectors but only 2^n have function value 0.

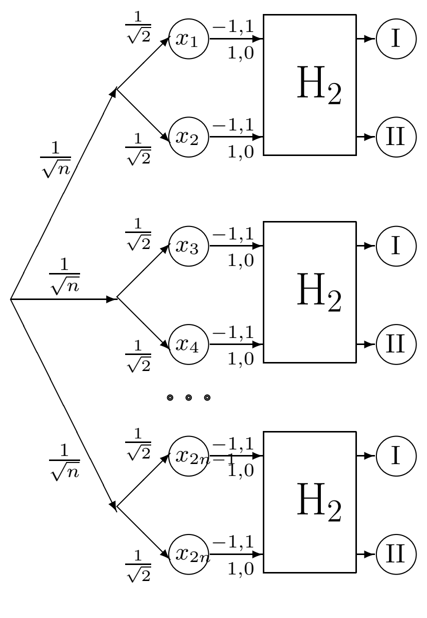


Fig. 6 *Quantum query algorithm with 2n variables .*

One of these algorithms gives zero values on input vectors ((00)*(11)*)*. We can get this algorithm (Figure 6) if we put output 0 to all first outputs. Nondeterministic polynomial is $\sum_{i=1}^{n} i(x_{2i-1} - x_{2i})$

We can make nondeterministic query algorithm which gives value zero if input vector is symmetric (Figure 7). Nondeterministic polynomial of this function is $\sum_{i=1}^{n} i(x_i - x_{2n-i+1})$. We put zero outputs to all first outputs.

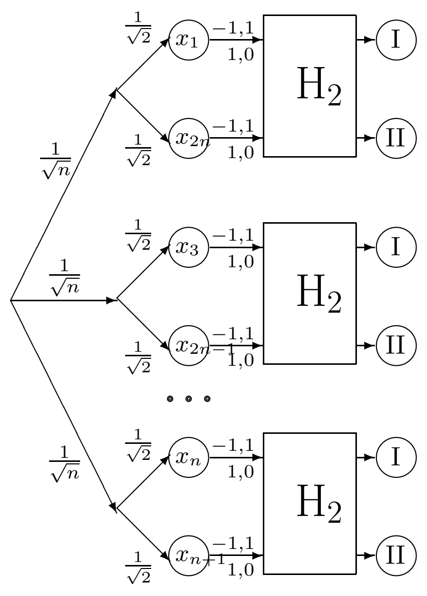


Fig. 7 *Quantum query algorithm for symmetric function .*

If we have some nondeterministic algorithms with $k_1, k_2 < 2n$ variables and $k_1 + k_2 = 2n$ then we can make nondeterministic quantum query algorithm with 2n variables by using theorem T1. It is possible to merge not only two algorithms but more than two.

Simplest degree-1 nondeterministic polynomial is $x_1 + x_2 + x_3 ....$ If $c_0 = 0$ then function value zero is only on input (000) and this is function OR. We can make nondeterministic quantum query algorithm which computes function OR with 2n variables similar with two variables OR algorithm (Section 3.1).

If we choose $c_0 = -n$ $(-n + \sum_{i=1}^{2n} x_i)$ then we obtain function P which give zero value if input vector have n 1's and n 0's ($|x| = n$). We have C_{2n}^{n} such input vectors.

Theorem 3. *There is nondeterministic quantum query algorithm with one query for function P.*

Proof. Nondeterministic quantum query algorithm is shown in figure 8. If function value is 0 then algorithm has n positive and n negative equivalent amplitudes after query. It is easy to see that in this case we can obtain in output I amplitude 0. Only output I has value 1, other outputs have values 0. Algorithm gives an answer 0 with probability 1 if function value is 0. If function value is 1 then algorithm have $n + k$ positive and $n - k$ negative amplitudes ($k \neq 0$) after query. In this case output I always has nonzero amplitude.

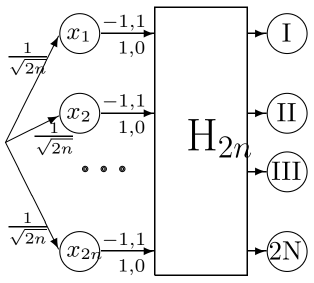


Fig. 8 *Quantum query algorithm with 2n variables .*

Theorem 4. *There is a nondeterministic quantum query algorithm with one query for function with 2n variables and nondeterministic polynomial* $-k + \sum_{i=1}^{2n} x_i$ *,* $0 \le k \le 2n$*.*

Proof. If $c_0 = -k$ then given function has value 0 if input vector has k 1's ($|x| = k$). We add to algorithm 2n states and after then we use Hadamard matrix H_{4n}. We make initial attribute distribution so that added states simulate input vector with k 0's and 2n-k 1's. Constructed algorithm has to give function value 0 if $|y| = 2n$ and this problem reduces to recognizing function P (Theorem 3)

Theorem 5. *There is a nondeterministic quantum query algorithm with one query for function with nondeterministic polynomial* $\sum_{i=1}^{2n} x_{2i-1} - x_{2i}$

Proof. We make this algorithm the same as in Theorem 3 except we put output value 1 to state II and values 0 to other states. Function gives value 0 only if input vector have k 1's in even positions and k 1's in odd positions. Second row in Hadamard matrix H_{2n} consists of one by one $\frac{1}{\sqrt{2n}}$ and $-\frac{1}{\sqrt{2n}}$. If function value is 0 then algorithm has amplitude 0 in II state. In other cases in state II algorithm has nonzero amplitude.

If we analyze all other possible degree-1 polynomials we can see that they are much more than number of polynomials considered. But we can not make nondeterministic quantum query algorithms for most of them so elegant.

References

1. L. Grover. A fast quantum mechanical algorithm for database search. Proceedings of the 28th ACM symposium on Theory of Computing, pp 212-219, 1996.
2. J. Gruska. Quantum Computing. McGraw-Hill, 1999.
3. M. Nielsen, I. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, 2000.
4. R.de Wolf. Nondeterministic Quantum Query and Quantum Communication Complexities. SIAM Journal on Computing, 32(3):681-699, 2003.

Quantum Query Algorithms for Certain Problems and Generalization of Algorithm Designing Techniques

Alina Dubrovska, Taisia Mischenko-Slatenkova, and Alexander Rivosh[⋆]

Institute of Mathematics and Computer Science University of Latvia
Raiņa bulvāris 29, Riga, LV-1459, Latvia
{Alina.Dubrovska, TaisiaMischenko, Alexander.Rivosh}@gmail.com

Abstract. Quantum algorithms can be analyzed in a query model to compute Boolean functions where input is given in a black box and the aim is to compute function value for arbitrary input using as few queries as possible. We concentrate on quantum query algorithm designing tasks in this paper. The main aim of the research was to find new efficient algorithms and develop general algorithm designing techniques. We present several exact quantum query algorithms for certain problems that are better than classical counterparts. Next, we introduce algorithm transformation methods that allow significant enlarging of exactly computable functions sets. Finally, we propose quantum algorithm designing methods. Given algorithms for the set of sub-functions, our methods use them to design a more complex one, based on algorithms described before. Methods are applicable for input algorithms with specific properties and preserve acceptable error probability and number of queries.

Kewords. Quantum computing, quantum query algorithms, complexity theory, Boolean functions, quantum algorithm design.

1 Introduction

Let $f(x_1, x_2, \dots, x_n) : \{0,1\}^n \to \{0,1\}$ be a Boolean function. We have studied the query model, where a black box contains the input $(x_1, x_2, \dots, x_n)$ and can be accessed by questioning x_i values. The goal here is to compute the value of the function. The complexity of a query algorithm is measured by number of questions it asks. The classical version of this model is known as *decision trees* [1]. Quantum query algorithms can solve certain problems faster than classical algorithms. The best-known exact quantum algorithm was designed for PARITY function with $n/2$ questions vs. n questions required by classical algorithm [2, 3].

The problem of quantum algorithm construction is not that easy. Although there is a large amount of lower and upper bound estimations of quantum algorithm complexity [2, 6, 7], examples of non-trivial and original quantum query

⋆ Research supported by the European Social Fund.

algorithms are very few. Moreover, there is no special technique described to build a quantum algorithm for a certain function with complexity defined in advance.

Boolean functions are widely adopted in real life processes, that is the reason why our capacity to build a quantum algorithm for an arbitrary function appears to be extremely important. While working on common techniques, we are trying to collect examples of efficient quantum algorithms to build up a base for powerful computation using the advantages of the quantum computer.

2 Notation and Definitions

Let $f(x_1, x_2 \ldots, x_n) : \{0,1\}^n \to \{0,1\}$ be a Boolean function. We use $\oplus$ to denote XOR operation (exclusive OR). We use $\bar{f}$ for the function $1-f$. We use abbreviation QQA for "quantum query algorithm" as well.

2.1 Quantum computing

We apply the basic model of quantum computing. For more details see textbooks by Gruska [4] and Nielsen and Chuang [5]. An n-dimensional quantum pure state is a vector $|\psi\rangle \in \mathbb{C}^n$ of norm 1. Let $|0\rangle$, $|1\rangle$, ..., $|n-1\rangle$ be an orthonormal basis for $\mathbb{C}^n$. Then, any state can be expressed as $|\psi\rangle = \sum_{i=0}^{n-1} a_i |i\rangle$ for some $a_i \in \mathbb{C}$. Since the norm of $|\psi\rangle$ is 1, we have $\sum_{i=0}^{n-1} |a_i|^2 = 1$. States $|0\rangle$, $|1\rangle$, ..., $|n-1\rangle$ are called *basic states*. Any state of the form $\sum_{i=0}^{n-1} a_i |i\rangle$ is called a *superposition* of $|0\rangle$, $|1\rangle$, ..., $|n-1\rangle$. The coefficient a_i is called an *amplitude* of $|i\rangle$. The state of a system can be changed using *unitary transformations*. Unitary transformation U is a linear transformation on $\mathbb{C}^n$ that maps vector of unit norm to vectors of unit norm.

There is the simplest case of quantum measurement used in our model. It is the full measurement in the computation basis. Performing this measurement on the state $|\psi\rangle = a_1 |0\rangle + \ldots + a_k |k\rangle$ gives the outcome i with probability $|a_i|^2$. The measurement changes the state of the system to $|i\rangle$ and destroys the original state $|\psi\rangle$.

2.2 Query model

Query model is probably the simplest model for computing Boolean functions. In this model, a black box contains the input $(x_1, x_2, \ldots, x_n)$ and can be accessed by questioning x_i values. Query algorithm must be able to determine the value of a function correctly for arbitrary input contained in a black box. The complexity of the algorithm is measured by the number of queries to the black box which it uses. The classical version of this model is known as *decision trees*. For details, see the survey by Buhrman and de Wolf [1].

We consider computing Boolean functions in the quantum query model. For more details, see the survey by Ambainis [6, 8] and textbooks by Gruska [4] and de Wolf [2]. A quantum computation with T queries is a sequence of unitary transformations:

$$U_0 \rightarrow Q_0 \rightarrow U_1 \rightarrow Q_1 \rightarrow \ldots \rightarrow U_{T-1} \rightarrow Q_{T-1} \rightarrow U_t,$$

where U_i's can be arbitrary unitary transformations that do not depend on the input bits x_1, x_2, ..., x_n; Q_i's are query transformations. Computation starts in the state $|\mathbf{0}\rangle$. Then we apply U_0, Q_0, ..., Q_{T-1}, U_T and measure the final state.

We use the following definition of query transformation: if input is a state $|\psi\rangle = \sum_i a_i |i\rangle$, then the output is $|\phi\rangle = \sum_i (-1)^{x_k} a_i |i\rangle$, where we can arbitrarily choose a variable assignment x_k for each amplitude a_i.

Each amplitude of the final quantum state corresponds to the algorithm output. We assign a value of a function to each output. The result of running algorithm on input X is j. Its probability equals the sum of squares of all the amplitudes, which corresponds to outputs with value j.

A very convenient way of quantum query algorithm representation is graphical picture and we will use this style describing designed quantum query algorithms.

2.3 Query Algorithm Complexity

The complexity of a query algorithm is based on the number of questions it uses to determine the value of a function on worst-case input.

The *deterministic complexity* of a function f, denoted by $D(f)$, is the maximum number of questions that must be asked on any input by a deterministic algorithm for f [1].

The sensitivity of f on input $(x_1, x_2, \ldots, x_n)$ is the number of variables x_i with the following property: $f(x_1, \ldots, x_i, \ldots, x_n) \neq f(x_1, \ldots, 1 - x_i, \ldots, x_n)$. The sensitivity of f is the maximum sensitivity of all possible inputs. It has been proved that $D(f) \geq s(f)$ [1].

A quantum query algorithm *computes f exactly* if the output equals $f(x)$ with a probability 1, for all $x\{0,1\}^n$. Complexity is denoted by $Q_E(f)$ [1].

A quantum query algorithm *computes f with bounded-error* if the output equals $f(x)$ with probability $p \geq 1/2$, for all $x \in \{0,1\}^n$. Complexity is denoted by $Q_P(f)$ [1].

3 Exact Quantum Query Algorithms for Certain Problems

In this section we present exact QQAs for several certain problems. We would like to emphasize that the best known separation between classical deterministic and exact quantum algorithm complexity is n vs. $\lceil n/2 \rceil$ for PARITY function [2, 3]. In our algorithms we do not exceed this limit and do not show the new best record, but our algorithms obtain exactly the same complexity gap.

3.1 3-variable function with 2 queries

In this section we present quantum query algorithm for 3-variable Boolean function that saves one query comparing to the best possible classical deterministic algorithm.

Problem 3.1. Check if all input variable values are equal.

Possible real life application is, for example, automated voting system, where statement is automatically approved only if all participants voted for acceptance/rejection equally. We provide solution for 3-party voting routine. We reduce Problem 3.1 to computing the following Boolean function defined by the logical formula: $EQUALITY_3(X) = \neg(x_1 \oplus x_2) \wedge \neg(x_2 \oplus x_3)$.

Deterministic complexity: $D(EQUALITY_3) = 3$, by sensitivity on any accepting input.

Algorithm 1. Exact quantum query algorithm for $EQUALITY_3$ is presented in Fig. 1. Each horizontal line corresponds to the amplitude of the basic state. Computation starts with amplitude distribution $\langle \mathbf{0} \mid = (1, 0, 0, 0)$. Three large rectangles correspond to the 4×4 unitary matrices (U_0, U_1, U_2). Two vertical layers of circles specify the queried variable order for each query (Q_0, Q_1). Finally, four small squares at the end of each horizontal line define the assigned function value for each output.

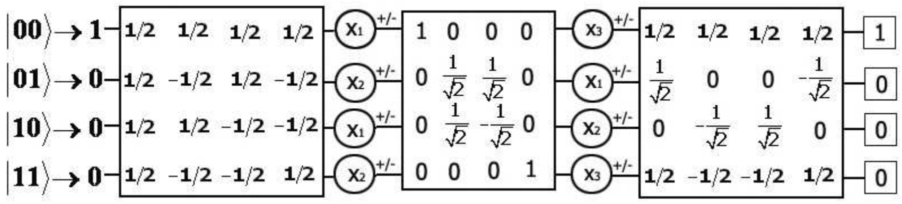


Fig. 1. Exact quantum query algorithm for $EQUALITY_3$

We show the computation process for accepting input $X = 111$:

$$\begin{aligned}\langle \psi \mid &= (1/2, 1/2, 1/2, 1/2)Q_0U_1Q_1U_2 = (-1/2, -1/2, -1/2, -1/2)U_1Q_1U_2 \\ &= (-1/2, -1/\sqrt{2}, 0, -1/2)Q_1U_2 = (1/2, 1/\sqrt{2}, 0, 1/2)U_2 = (1, 0, 0, 0) \\ &\Rightarrow [ACCEPT]\end{aligned}$$

3.2 4-variable function with 2 queries

In this section we present our solution for the well-known computational problem of comparing two binary strings.

Problem 3.2. Check if two binary strings are equal.

We present an algorithm for strings of length 2. We reduce Problem 3.2 to computing the Boolean function of 4 variables. The first two variables represent the first string, and the second two variables correspond to the second string. Boolean function can be represented by formula: $STRING\_EQ_4(X) = \neg((x_1 \oplus x_3) \vee (x_2 \oplus x_4))$.

Deterministic complexity: $D(STRING\_EQ_4) = 4$, by sensitivity on accepting input.

Algorithm 2. Exact quantum query algorithm for $STRING\_EQ_4$ is presented in Fig. 2.

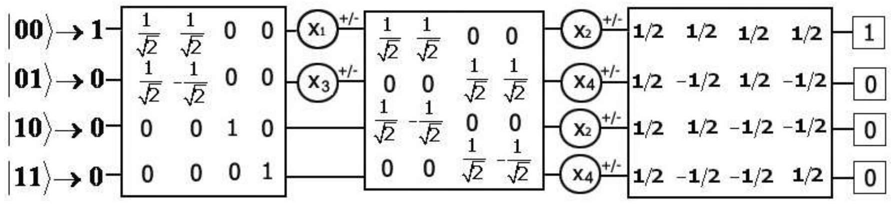


Fig. 2. Exact quantum query algorithm for $STRING\_EQ_4$

3.3 $2n$-variable functions with n queries

In this section we present a set of exact QQAs, which perform the largest advantage of quantum query complexity over deterministic one that is known for today.

Problem 3.3. Let us consider a function T_4 of 4 variables defined by a truth-table:

x_1	x_2	y_1	y_2	$T_4(XY)$	x_1	x_2	y_1	y_2	$T_4(XY)$
0	0	0	0	1	1	0	0	1	1
0	0	1	1	1	1	0	1	0	1
0	1	0	1	1	1	1	0	0	1
0	1	1	0	1	1	1	1	1	1

otherwise $T_4(XY) = 0$

Deterministic complexity: $D(T_4) = 4$, by sensitivity on any accepting input.

Algorithm 3. Exact QQA for T_4 with 2 queries is presented in Figure 3.

It appears that the idea of this algorithm can be used for a bigger number of variables. Let us define a function T_6 of 6 variables:

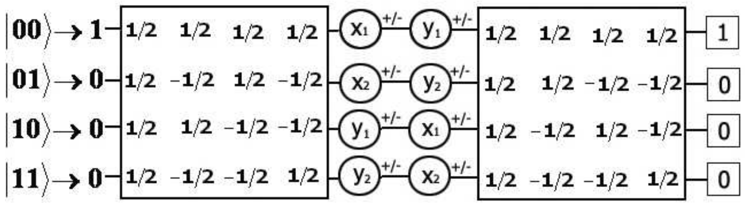


Fig. 3. Exact quantum query algorithm for T_4

XY	T_6	XY	T_6	XY	T_6	XY	T_6
000000	1	010010	1	100001	1	110011	1
000101	1	010111	1	100100	1	110110	1
001001	1	011011	1	101000	1	111010	1
001100	1	011110	1	101101	1	111111	1

otherwise $T_6(XY) = 0$

T_6 is computable by algorithm similar to Algorithm 3, which is specified by following sequence: $U_0 \to Q_1Q_2Q_3 \to U_1$, where $Q_1 = (\alpha_1 = x_1, \alpha_2 = x_2, \alpha_2 = y_1, \alpha_4 = y_2)$, $Q_2 = (\alpha_1 = x_2, \alpha_2 = x_3, \alpha_2 = y_2, \alpha_4 = y_3)$, $Q_3 = (\alpha_1 = y_1, \alpha_2 = y_3, \alpha_2 = x_1, \alpha_4 = x_3)$.

It is possible to generalize this idea.

Theorem 3.4. *For Boolean function T_{2n} of $2n$ variables, an exact quantum query algorithm does exist, that computes this function with n queries.*

Proof. The idea of the algorithm remains the same as in the case of T_4 and T_6, we use a chain of transformations $U_0 \to Q \to U_1$, where Q is a sequence of queries. We visualize Q in the form of a matrix, where i-th query is represented by the i-th column of matrix Q. Generalizing Q for T_{2n}: use Q_{even} for even number n, Q_{odd} for odd n.

$$Q_{\text{even}} = \begin{pmatrix} x_1 & \dots x_{\frac{n}{2}} & y_1 & \dots y_{\frac{n}{2}} \\ x_{\frac{n}{2}+1} & \dots & x_n & y_{\frac{n}{2}+1} & \dots & y_n \\ y_1 & \dots y_{\frac{n}{2}} & x_1 & \dots x_{\frac{n}{2}} \\ y_{\frac{n}{2}+1} & \dots & y_n & x_{\frac{n}{2}+1} & \dots & x_n \end{pmatrix}$$

$$Q_{\text{odd}} = \begin{pmatrix} x_1 & \dots x_{\lfloor \frac{n}{2} \rfloor+1} & y_1 & \dots y_{\lfloor \frac{n}{2} \rfloor} \\ x_{\lfloor \frac{n}{2} \rfloor+1} & \dots & x_n & y_{\lfloor \frac{n}{2} \rfloor+2} & \dots & y_n \\ y_1 & \dots y_{\lfloor \frac{n}{2} \rfloor+1} & x_1 & \dots x_{\lfloor \frac{n}{2} \rfloor} \\ y_{\lfloor \frac{n}{2} \rfloor+1} & \dots & y_n & x_{\lfloor \frac{n}{2} \rfloor+2} & \dots & x_n \end{pmatrix}$$

The fact is that the exact quantum algorithm computing the function T_{2n} can be changed to compute $T_{2(n+1)}$ by transformation of query sequence Q. Hence, we have an unlimited set of exact quantum algorithms computing corresponding functions T_{2n} with n queries only, while deterministic algorithm can do it with $2n$ queries.

4 Algorithm Transformation Methods

In this section we introduce quantum query algorithm transformation methods that can be useful for enlarging a set of exactly computable Boolean functions. Each method on input receives exact QQA, processes it as defined, and as the result, a slightly different exact algorithm that computes another function is obtained.

4.1 Output value assignment inversion

The first method is the simplest one. All we need to do with original algorithm is to change assigned function value for each output to the opposite.

First Transformation Method - Output value assignment inversion
Input. An arbitrary exact QQA that computes $f(X)$.
Transformation actions.
• For each algorithm output change assigned value of function to opposite. If original assignment was $QM = (\alpha_1 \equiv k_1, \ldots, \alpha_m \equiv k_m)$, where $k_i \in \{0,1\}$, then it is transformed to $QM' = (\alpha_1 \equiv \overline{k}_1, \ldots, \alpha_m \equiv \overline{k}_m)$, where $\overline{k}_i = 1 - k_i$.
Output. An exact QQA that computes $\overline{f}(X)$.

Box 1. Description of the First Transformation Method.

4.2 Output value assignment permutation

Describing the next method we will limit ourselves to using only exact QQA with specific properties as an input for transformation method.

Property 1. An exact QQA satisfies *Property 1* IFF on any input system state before a measurement is such that for exactly one amplitude α_i holds true that $|\alpha_i|^2 = 1$. For other amplitudes holds true that $|\alpha_j|^2 = 0$, for $\forall j \neq i$.

Algorithm 1, Algorithm 2 and Algorithm 3 from section 3 satisfy Property 1.

Second Transformation Method
- Output value assignment permutation

Input.

- An exact QQA satisfying *Property 1* that computes $f(X)$.
- Permutation σ of the set *OutputValues*$=\{k_1, k_2, \ldots, k_m\}$.

Transformation actions.

- Permute function values assigned to outputs in order specified by σ.
  If original assignment was $QM = (\alpha_1 \equiv k_1, \ldots, \alpha_m = k_m)$,
  where $k_i \in \{0,1\}$,
  then it is transformed to $QM' = (\alpha_1 \equiv \sigma(k_1), \ldots, \alpha_m \equiv \sigma(k_m))$.
  where $\overline{k}_i = 1 - k_i$.

Output. An exact QQA for some function $g(X)$.

Box 2. Description of the Second Transformation Method.

Proof of correctness. Application of the method does not break the exactness of QQA, because the essence of *Property 1* is that before the measurement we always obtain non-zero amplitude in exactly one output. Since function value is clearly specified for each output we would always observe specific value with probability 1 for any input.

The structure of the new function $g(X)$ strictly depends on internal properties of the original algorithm. To explicitly define new function there is a need to inspect original algorithm behavior on each input and construct a truth table for new output value assignment.

4.3 Query variable permutation

Let σ be a permutation of the set $\{1, 2, \ldots, n\}$, where elements correspond to variable numbers. By saying that the function $g(X)$ is obtained by permutation of $f(X)$ variables we mean the following: $g(X) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$. In our Third Transformation Method we expand the idea of variable permutation to QQA algorithm definition.

Third Transformation Method - Query variable permutation

Input.

- An arbitrary exact QQA that computes $f_n(X)$
- Permutation σ of the set *VarNum*$=\{0, 1, \ldots, n\}$.

Transformation actions.

- Apply permutation of variable numbers σ to all query transformations.
  If original i-th query was defined as $QQ_i = (\alpha_1 \equiv k_1, \ldots, \alpha_m \equiv k_m)$,
  Then it is transformed to $QQ'_i = (\alpha_i = \sigma(k_1), \ldots, \alpha_m = \sigma(k_m))$,
  $k_i \in \{1, \ldots, n\}$.

Output. An exact QQA for computing a function
$g(X) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$.

Box 2. Description of the Third Transformation Method.

5 Algorithm Designing Methods

In this section we will present several quantum query algorithm designing methods. Each method requires explicitly specified exact QQAs on input, and as a result a bounded-error QQA for more complex function is constructed. Our methods maintain quantum query complexity for complex function in comparison to increased deterministic complexity, thus enlarging the gap between classical and quantum complexities of the algorithm.

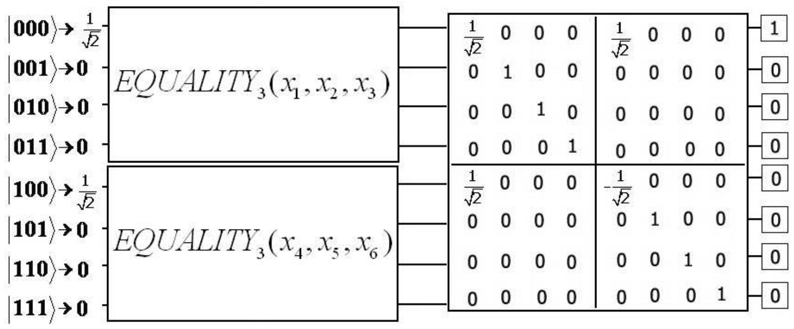
5.1 Obtaining a gap $D(f) = 6$ vs. $Q_{3/4}(f) = 2$

We consider composite Boolean function, where two instances of $EQUALITY_3$ (section 3.1) are joined with logical AND operation:

$$EQUALITY_3^{\wedge 2}(x_1, \ldots, x_6) = (\neg(x_1 \oplus x_2) \wedge \neg(x_2 \oplus x_3)) \wedge (\neg(x_4 \oplus x_5) \wedge \neg(x_5 \oplus x_6))$$

Deterministic complexity. $D(EQUALITY_3^{\wedge 2}) = 6$, by sensitivity on $X = 111111$.

Our approach in designing an algorithm for $EQUALITY_3^{\wedge 2}$ is to employ quantum parallelism and superposition principle. We execute algorithm pattern defined by original algorithm for $EQUALITY_3$ in parallel for both blocks of variables of $EQUALITY_3^{\wedge 2}$. Finally we apply additional quantum gate to correlate amplitude distribution. Algorithm flow is depicted explicitly in Figure 4.



$|000\rangle \to \frac{1}{\sqrt{2}}$
$|001\rangle \to 0$
$|010\rangle \to 0$
$|011\rangle \to 0$
$|100\rangle \to \frac{1}{\sqrt{2}}$
$|101\rangle \to 0$
$|110\rangle \to 0$
$|111\rangle \to 0$
$EQUALITY_3(x_1, x_2, x_3)$
$EQUALITY_3(x_4, x_5, x_6)$
$\begin{array}{cccc|cccc} \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline \frac{1}{\sqrt{2}} & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$
1
0
0
0
0
0
0
0

Fig. 4. Bounded-error QQA for $EQUALITY_3^{\wedge 2}$

Quantum complexity. Algorithm presented in Figure 4 computes $EQUALITY_3^{\wedge 2}$ using 2 queries with correct answer probability $p = 3/4$: $Q_{3/4}(EQUALITY_3^{\wedge 2}) = 2$.

5.2 First Designing Method

In this section we will generalize approach used in the previous section. To be able to use generalized version of method we will limit ourselves to examining only exact QQA with specific properties.

Property 2+ We say that the exact QQA satisfies *Property2+* IFF there is exactly one accepting basic state and on any input for its amplitude $\alpha \in \mathbb{C}$ only two values are possible before the final measurement: either $\alpha = 0$ or $\alpha = 1$.

Algorithm 1 presented in section 3.1 satisfies Property 2+.

Property 2- We say that the exact QQA satisfies *Property2-* IFF there is exactly one accepting basic state and on any input for its amplitude $\alpha \in \mathbb{C}$ only two values are possible before the final measurement: either $\alpha = 0$ or $\alpha = -1$.

Lemma 5.1. *It is possible to transform algorithm that satisfies* Property2- *to algorithm satisfying* Property2+ *by applying additional unitary transformation.*

Proof. Let us assume that we have QQA satisfying *Property2-* and k is the number of accepting output. To transform algorithm to satisfy *Property2+* apply the following quantum gate:

$$U = (u_{ij}) = \begin{cases} 0, & \text{if} i \neq j \\ 1, & \text{if} i = j \neq k \\ -1, & \text{if} i = j = k \end{cases}$$

First Designing Method

Input.

- Two exact QQAs A1 and A2 satisfying *Property2+* that compute correspondingly Boolean functions $f_1(X_1)$ and $f_2(X_2)$.

Transformation actions.

1. If A1 and A2 utilize quantum systems of different size, then extend the smallest one with auxiliary space to obtain equal number of amplitudes. We denote the dimension of obtained Hilbert spaces with m.
2. For new algorithm utilize a quantum system with $2m$ amplitudes.
3. Combine unitary transformations and queries of A1 and A2 in the following way: $U_i = \begin{pmatrix} U_i^1 & O \\ O & U_i^2 \end{pmatrix}$, where O's are $m \times m$ zero-matrices, U_i^1 and U_i^2 are either unitary transformations or query transformations of A_1 and $A2$.
4. Start computation from the state $\langle \psi \mid = (1/\sqrt{2}, 0, \ldots, 0, 1/\sqrt{2}, 0, \ldots, 0)$.
5. Before the final measurement apply additional unitary gate. Let's denote the positions of accepting outputs of A1 and A2 by acc_1 and acc_2. Then the final gate is defined as follows:
$$U = (u_{ij}) = \begin{cases} 1, & \text{if } (i = j)\,\&\,(i \neq acc_1)\,\&\,(i \neq (m + acc_2)) \\ 1/\sqrt{2}, & \text{if } (i = j = acc_1)\text{ OR }(i = j = (m + acc_2)) \\ 1/\sqrt{2}, & \text{if } (i = acc_1)\,\&\,(j = (m + acc_2)) \\ & \text{OR }(i = (m + ac_2))\,\&(j = acc_1) \\ 0, & \text{otherwise} \end{cases}$$
6. Define as accepting output exactly one basic state $|acc_1\rangle$.

Output. A bounded-error QQA A computing a function $F(X) = f_1(X) \wedge f_2(X_2)$ with probability $p = 3/4$ and complexity $Q_{3/4}(A) = \max(Q_E(A_1), Q_E(A_2))$.

Box 4. Description of the First Designing Method.

5.3 Obtaining a gap $D(f) = 8$ vs. $Q_{3/4}(f) = 2$

We will try to apply approach described in the First Designing Method to a function $STRING\_EQ_4$ (section 3.2), ignoring the fact that algorithm does not satisfy required property. The resulting algorithm *Algorithm 4* has exactly the same structure as the one presented in Figure 4, the difference is that now we execute exact QQA for $STRING\_EQ_4$ in parallel instead of EQUALITY$_3$.

Quantum complexity. Bounded-error QQA *Algorithm 4* is computing Boolean function defined as:
$$STRING\_\text{EQ}_2^{||} = \begin{cases} 1, \text{if} X \in \{00000000, 00001111, 11110000, 11111111\} \\ 1, \text{if} X \in \{01010101, 01011010, 10101010, 10100101\} \\ 0, \text{otherwise} \end{cases} \text{and}$$
complexity $Q_{3/4}(\text{Algorithm4})$ is 2.

Deterministic complexity. $D(STRING\_\mathrm{EQ}_2^{||}) = 8$, by sensitivity on accepting input.

5.4 Second Designing Method

Let us define the next property of exact QQA that extends previous Property2x.

Property 3. We say that the exact QQA satisfies *Property3* IFF there is exactly one accepting basic state and after processing any input its amplitude before the measurement is $\alpha \in \{-1, 0, 1\}$.

Algorithm 2 (section 3.2) and Algorithm 3 (section 3.3) satisfy Property 3.

We denote a set of accepting inputs for Boolean function F by Acc_F. While discussing exact QQA satisfying Property3 we define the following sets:

$$Acc_F^+ = \{X \in Acc_F \mid \text{accepting output amplitude before measurement is } +1\}$$

$$Acc_F^- = \{X \in Acc_F \mid \text{accepting output amplitude before measurement is } -1\}$$

Second Designing Method
Input. • Two exact QQAs A1 and A2 satisfying *Property3* that compute correspondingly Boolean functions $f_1(X_1)$ and $f_2(X_2)$. **Transformation actions.** • Perform steps 1–6 described in the First Designing Method (Box 4). **Output.** A bounded-error QQA A computing a function $F(X)$ defined below with correct answer probability $p = 3/4$ and complexity $Q_{3/4}(A) = \max(Q_E(A_1), Q_E(A_2))$. $F(X) = \begin{cases} 1, \text{if } X \in (Acc_{f1}^+ \times Acc_{f2}^+) \cup (Acc_{f1}^- \times Acc_{f2}^-) \\ 0, \text{otherwise} \end{cases}$

Box 5. Description of the Second Designing Method.

5.5 Obtaining a gap $D(f) \geq 9$ vs. $Q_{9/16}(f) = 2$

Let us try to increase the effect gained by employing quantum parallelism. The next idea is to execute 4 instances of algorithm in parallel, adjusting algorithm parameters in appropriate way. We will take function EQUALITY$_3$ from section 3.1 as a pattern. Designed *Algorithm 5* and additional gates are presented in Figure 5 below.

Additional quantum gates (empty matrix cells correspond to "0") U' and U'' are as in Figures 6 and 7, respectively.

After examination of algorithm computational flow and calculation of probabilities we obtained result that is formulated in the next statement. Quantum complexity. Bounded-error QQA *Algorithm5* is computing function defined as:

$$F(x_1, \ldots, x_{12}) = 1$$
$$\iff \left(\begin{array}{l} \text{Not less than 3 functions from: } EQUALITY(x_1, x_2, x_3) \\ EQUALITY(x_4, x_5, x_6),\ EQUALITY(x_7, x_8, x_9) \\ EQUALITY(x_{10}, x_{11}, x_{12}) \text{ give value "1".)} \end{array} \right)$$

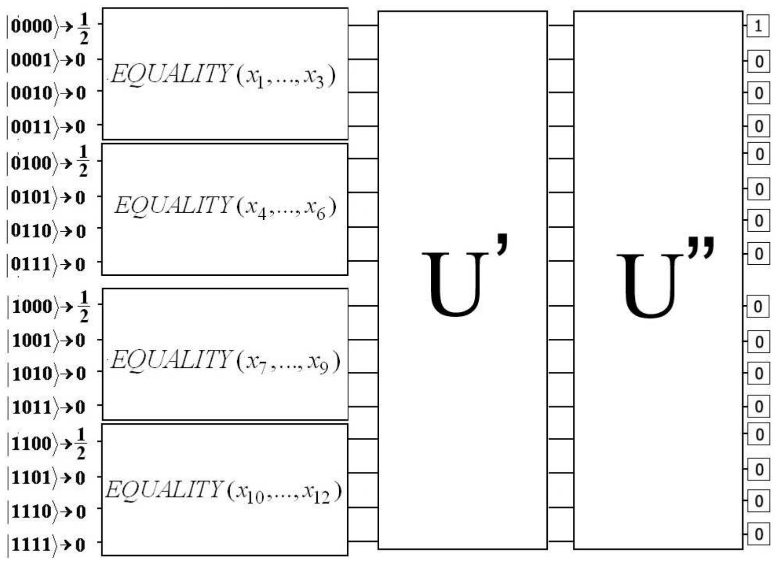


Fig. 5. Quantum query algorithm Algorithm 5.

$$
\begin{pmatrix}
\frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & & & 0 & & & & 0 & & & & & & & 0 \\
0 & & 1 & & 0 & & & & 0 & & & & & & & 0 \\
0 & & & 1 & 0 & & & & 0 & & & & & & & 0 \\
\frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} & & & & 0 & & & & & & & 0 \\
0 & & & & & 1 & & & 0 & & & & & & & 0 \\
0 & & & & & & 1 & & 0 & & & & & & & 0 \\
0 & & & & & & & 1 & 0 & & & & & & & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\
0 & & & & & & & & 0 & 1 & & & 0 & & & 0 \\
0 & & & & & & & & 0 & & 1 & & 0 & & & 0 \\
0 & & & & & & & & 0 & & & 1 & 0 & & & 0 \\
0 & & & & & & & & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} & & & 0 \\
0 & & & & & & & & 0 & & & & & 1 & & 0 \\
0 & & & & & & & & 0 & & & & & & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

Fig. 6. Matrix U'

$$\begin{pmatrix}
\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & & & & & & & 0 & & & & & & & 0 \\
0 & & 1 & & & & & & 0 & & & & & & & 0 \\
0 & & & 1 & & & & & 0 & & & & & & & 0 \\
0 & & & & 1 & & & & 0 & & & & & & & 0 \\
0 & & & & & 1 & & & 0 & & & & & & & 0 \\
0 & & & & & & 1 & & 0 & & & & & & & 0 \\
0 & & & & & & & 1 & 0 & & & & & & & 0 \\
\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & & & & & & & & 0 & 1 & & & & & & 0 \\
0 & & & & & & & & 0 & & 1 & & & & & 0 \\
0 & & & & & & & & 0 & & & 1 & & & & 0 \\
0 & & & & & & & & 0 & & & & 1 & & & 0 \\
0 & & & & & & & & 0 & & & & & 1 & & 0 \\
0 & & & & & & & & 0 & & & & & & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Fig. 7. Matrix U''

and complexity is $Q_{9/16}(\text{Algorithm5}) = 2$.

Deterministic complexity. This time we did not achieve maximal possible gap. From the definition of function F we find that sensitivity is $s(F) = 9$. Thus in this case we can only register a gap $D(f) \geq 9$ vs. $Q_{9/16}(f) = 2$.

5.6 Third Designing Method

Our last method is generalization of approach demonstrated in the previous section. We leave detailed description of transformation actions for an interested reader as an exercise.

Third Designing Method

Input.

- Four exact QQAs A1, A2, A3, A4 satisfying *Property2+* that compute correspondingly Boolean functions $f_1(X_1)$, $f_2(X_2)$, $f_3(X_3)$, $f_4(X_4)$.

Transformation actions.

- Combine approach described in section 5.5 with the First Designing Method and adjust according to the structure of input exact QQAs.

Output. A bounded-error QQA A computing a function $F(X)$ with probability $p = 9/16$ and complexity $Q_{9/16} = \max(Q_E(A_1), Q_E(A_2), Q_E(A_3), Q_E(A_4))$, and:
$F(X) = 1 \iff (f_1(X_1) + f_2(X_2) + f_3(X_3) + f_4(X_4) \geq 3)$

Box 6. Description of the Third Designing Method.

We would like to note, that it is technically possible to apply approach described in the Third Designing Method to exact QQAs satisfying Property 3. Definition of a computable function will be more complex, but the most important is that we can design a lot of different algorithms without increasing number of queries in such a way.

6 Results of Applying Methods

We have applied transformation and designing methods to two basic exact QQAs described in Section 3. Totally we obtained 32 exact QQAs and 512 QQAs with bounded error. Each algorithm computes different Boolean function and uses only 2 queries. Results are summarized in Table 1, where n is number of variables of computable function.

	Exact QQAs			**QQAs with Bounded error**			
	Total	Property 2x	Property 3	n=6	n=7	n=8	n=12
n=3	**8**	4	4	**16**	**96**		**256**
n=4	**24**		12			**144**	
Total:	**32 algorithms**			**512 algorithms**			

Table 1. Results of transformations and designing methods application.

The important point is that invention of each brand-new exact QQA with required properties will significantly increase a set of efficiently computable functions at once.

7 Conclusion

We describe exact quantum query algorithms for several problems in the present paper. Moreover, all our algorithms have the largest possible gap between quantum and deterministic complexities known for today. Next, we propose techniques that allow transformation of an existing quantum query algorithm for a certain Boolean function so that the resulting algorithm computes a function with other logical structure. Finally, we suggest approaches that allow building bounded-error quantum query algorithms for complex functions based on the already known exact algorithms. Combination of those three aspects allows us to construct large sets of efficient quantum algorithms for various Boolean functions.

Further work in that direction could be to invent new efficient quantum algorithms that exceed already known separation from classical algorithms. Another important direction is improvement of general algorithm designing techniques.

References

1. H. Buhrman and R. de Wolf: “Complexity Measures and Decision Tree Complexity: A Survey”. Theoretical Computer Science, v. 288(1): 21–43 (2002).
2. R. de Wolf: “Quantum Computing and Communication Complexity”. University of Amsterdam (2001).
3. R. Cleve, A. Ekert, C. Macchiavello, et al. “Quantum Algorithms Revisited”. Proceedings of the Royal Society, London, A454 (1998).
4. J. Gruska: “Quantum Computing”. McGraw-Hill (1999).
5. M. Nielsen, I. Chuang: “Quantum Computation and Quantum Information”. Cambridge University Press (2000).
6. A. Ambainis: “Quantum query algorithms and lower bounds (survey article)”. Proceedings of FOTFS III, to appear.
7. A. Ambainis and R. de Wolf: “Average-case quantum query complexity”. Journal of Physics A 34, pp 6741–6754 (2001).
8. A. Ambainis. “Polynomial degree vs. quantum query complexity”. Journal of Computer and System Sciences 72, pp. 220–238 (2006).

TUCS

Workshop on

Reachability Problems

7-8 July 2007, Turku, Finland

Editors:

Vesa Halava
Igor Potapov

Preface

The Workshop on *Reachability Problems in Computational Models* is a satellite event of the Developments in Language Theory Conference (DLT'07) that was held in Turku, Finland on July 7-8, 2007. The workshop is specifically aimed at gathering together scholars from diverse disciplines and backgrounds interested in reachability problems that appear in

- Algebraic structures
- Computational models
- Hybrid systems
- Verification

The workshop is mainly focused on reachability problems in different computational models and systems. Topics of interest include (but are not limited to): Reachability analysis in counter (timed, cellular, communicating) automata; Petri-Nets; computational aspects of semigroups, groups and rings; verification and reachability analysis for infinite state systems, rewriting systems, dynamical and hybrid systems; predictability in iterative maps and new computational paradigms, frontiers between decidable and undecidable problems.

The workshop was supported by the University of Turku. It was jointly organized by University of Turku and University of Liverpool. The high quality of the volume was achieved through the hard work of the Program Committee, consisting of Vincent Blondel, Olivier Bournez, Christian Choffrut, Javier Esparza, Vesa Halava, Oscar Ibarra and Igor Potapov, with the help of external referees, among them Paul Bell, Eero Lehtonen and Alexei Lisitsa.

The Programme Committee, selected six papers to be presented as regular contributions. In addition the program of the workshop included four invited talks given by

- Vincent Blondel (Louvain, Belgium)
- Christian Choffrut (Paris, France)
- Javier Esparza (Munich, Germany)
- Oscar Ibarra (Santa Barbara, USA).

June 2007

Vesa Halava
Igor Potapov

Table of Contents

Products of Matrices and Reachability of their Optimal Rate of Growth

Vincent Blondel

Department of Mathematical Engineering
Faculty of Applied Sciences
Catholic University of Louvain
Louvain-la-Neuve, Belgium

Abstract. In this talk I will survey several aspects of an apparently simple (but so far unsolved) question: under what conditions on the matrices A and B do all infinite products of the type ABBABAAAB... converge to zero? Even for matrices with rational entries, there is no known algorithm for this problem and it is so far unknown if the problem is algorithmically decidable.
The maximal asymptotic growth rate that can be obtained by forming long products of matrices was first defined by Rota and Strang in the 1960s and is known as the ”joint spectral radius” of the matrices. Convergence to zero of all infinite products is equivalent to the requirement that the joint spectral radius be less than one. In the last decade the joint spectral radius has appeared a number of application contexts, including hybrid systems, multi-agent networks, wavelets, capacity of codes, and sensor networks.
The joint spectral radius is notoriously difficult to compute even when constraints are imposed on the number of matrices, on their size, or on their entries. I will briefly describe a number of NP-hard, undecidable and other depressing negative results and will then move to more positive aspects. In particular, I will describe an algorithm that computes the joint spectral with arbitrary high accuracy and that is polynomial in the size of the matrices once the desired accuracy is fixed and I will present recent results for the computation of the capacity of codes and for sensor networks.
During the talk I will mention a problem that has attracted much attention and that is still unsolved. The problem is this: assume that we want to obtain the largest possible asymptotic rate of growth of long products of matrices; can this optimal rate always be obtained by a periodic product? This is known not to be true for matrices with real entries but the case of rational (or even binary) entries is unsolved. This last question relates to a number of situations where one is asked whether or not optimality can be achieved with a periodic strategy.
This is joint work with a number of co-authors. Some of my co-authored publications that are relevant to this subject are [1–7].

References

1. Vincent D. Blondel, Raphael Jungers, Vladimir Protasov, On the complexity of computing the capacity of codes that avoid forbidden difference patterns. IEEE Transactions on Information Theory, 52:11, pp. 5122-5127, 2006.

2. Vincent D. Blondel, Yurii Nesterov, Computationally efficient approximations of the joint spectral radius. SIAM Journal of Matrix Analysis, 27:1, pp. 256-272, 2005.
3. Vincent D. Blondel, Vincent Canterini, Undecidable problems for probabilistic automata of fixed dimension, Theory of Computing systems, 36, pp. 231-245, 2003.
4. Vincent D. Blondel, J. Theys and A. A. Vladimirov, An elementary counterexample to the finiteness conjecture, SIAM Journal on Matrix Analysis, 24:4, pp. 963-970, 2003.
5. Vincent D. Blondel, John N. Tsitsiklis, The boundedness of all products of a pair of matrices is undecidable, Systems and Control Letters, 41:2, pp. 135-140, 2000.
6. John Tsitsiklis, Vincent Blondel, The Lyapunov exponent and joint spectral radius of pairs of matrices are hard – when not impossible – to compute and to approximate, Mathematics of Control, Signals, and Systems, 10, pp. 31-40, 1997.
7. Vincent Blondel, John Tsitsiklis, When is a pair of matrices mortal?, Information Processing Letters, 63, pp. 283-286, 1997.

Weakening Presburger Arithmetic

Christian Choffrut

Laboratoire LIAFA, Université de Paris 7
2, pl. Jussieu, 75251, Paris Cedex 05
`cc@liafa.jussieu.fr`,
`http://www.liafa.jussieu.fr/~cc`

Abstract. We consider logics on $\mathbb{Z}$ and $\mathbb{N}$ which are weaker than Presburger Arithmetic and we settle the following decision problem: given a k-ary relation on $\mathbb{Z}$ and $\mathbb{N}$ which is first order definable in Presburger Arithmetic, is it definable in these weaker logics? These logics, intuitively, are obtained in two different ways. First by introducing modulo and threshold counting predicates on the difference of two variables and second by depriving $\mathbb{Z}$ from the ordering.

1 Background and definitions

Presburger arithmetic is the first order theory of integers $\mathbb{Z}$ with the operation of addition and the usual ordering, though the same term might also refer to the first order theory of nonnegative integers with the addition and the equality. Whichever arithmetic is meant should be clear from the context. The validity of a closed formula in this structure is proved decidable via quantifier elimination in the extension including all predicates $x = b \mod a$ with $a \in \mathbb{N}$ and $0 \leq b < a$, see e.g., [8, Chap. III.4]. Though long underestimated, this result is nowadays one of the main tools available in model-checking and program verification and the number of papers refereeing to it exceed several hundreds. Here we are concerned with definability in various, strictly weaker substructures on the same domain or possibly on the domain of nonnegative integers. The general question is as follows: given an arbitrary Presburger formula with n free variables defining an n-ary relation, is it first order definable in the weaker structure? E., g., the binary relation in $\mathbb{Z}$ defined by the formula $\phi(x, y) = ((x > 0) \vee (x < 0)) \wedge ((y > 0) \vee (y < 0))$ is definable without the ordering but the relation $\phi(x, y) = x < y$ is not.

The substructures we investigate are summarized in table 1. The integers a, b, c satisfy the conditions $0 \leq b < a$ and $c \in \mathbb{Z}$ or $c \in \mathbb{N}$, depending on which structure it refers to. The subscripts used should suggest the general idea of *modulo* and *threshold* counting. Observe that the first structure has no predicate (except equality) while the remaining structures have no operations.

Some of these structures were studied earlier from different points of view. In [5] the author studies the complexity of the validity of a closed expression in $\mathcal{Z}_{\text{thresh}}$ and of the quantifier elimination. A precise estimate is given and compared to the similar problems in full Presburger arithmetic. The same questions are solved when substituting the additive group of rationals $\mathbb{Q}$ for the group of

structure	domain	functions	predicates
$\mathcal{Z}$	$\mathbb{Z}$	$x+y$	$x<y$
$\mathcal{N}_p$	$\mathbb{N}$	$x+y$	$=$
$\mathcal{Z}^W$	$\mathbb{Z}$	$x+y$	$=$
$\mathcal{Z}_{\text{thresh + mod}}$	$\mathbb{Z}$	none	$(x \geq c)_{c\in\mathbb{Z}}, (x \pm y \geq c)_{c\in\mathbb{Z}}, (x = b \mod a)_{0\leq b<a}$
$\mathcal{N}_{\text{thresh + mod}}$	$\mathbb{N}$	none	$(x \geq c)_{c\in\mathbb{N}}, (x - y \geq c)_{c\in\mathbb{N}}, (x = b \mod a)_{0\leq b<a}$
$\mathcal{Z}_{\text{mod}}$	$\mathbb{Z}$	none	$x \pm y \geq 0, (x = b \mod a)_{0\leq b<a}$
$\mathcal{N}_{\text{mod}}$	$\mathbb{N}$	none	$x - y \geq 0, (x = b \mod a)_{0\leq b<a}$
$\mathcal{Z}_{\text{thresh}}$	$\mathbb{Z}$	none	$(x \geq c)_{c\in\mathbb{Z}}, (x \pm y \geq c)_{c\in\mathbb{Z}}$
$\mathcal{N}_{\text{thresh}}$	$\mathbb{N}$	none	$(x \geq c)_{c\in\mathbb{N}}, (x - y \geq c)_{c\in\mathbb{N}}$

Table 1. The different structures studied in this paper.

integers. In [6] the structure $\mathcal{N}_{\text{thresh + mod}}$ is not obtained as a reduction of the Presburger arithmetics but rather as an extension of the first order logic of the successor. A characterization is given but no decidability issue is tackled.

Our main result is the following, cf. [2] and [1].

Theorem *Given a relation over $\mathbb{Z}$ (resp. $\mathbb{N}$) which is first order definable in Presburger Arithmetic, for each one of the structures of table 1 whose domain is $\mathbb{Z}$ (resp. $\mathbb{N}$), it is recursively decidable whether or not this relation is first order definable in this structure.*

2 The ingredients of the proof

In this section we intend to give an idea of the ingredients for the proof of the main result. We concentrate on the two structures $\mathcal{Z}^W$ and $\mathcal{Z}_{\text{thresh + mod}}$. In other words we are given a relation in $\mathbb{Z}^k$ defined by a $\mathcal{Z}$-formula with k free variables and we want to decide whether or not it is $\mathcal{Z}^W$- (resp. $\mathcal{Z}_{\text{thresh + mod}}$-) definable. The proof for the remaining structures does not require essentially different arguments.

The following is the basic result on first order definable subsets in Presburger arithmetic, see [4, 3, 7].

Theorem 1. *Given a subset $X \subseteq \mathbb{Z}^k$ (resp. $X \subseteq \mathbb{N}^k$) the following conditions are equivalent*

1. *X is a finite union of $\mathbb{Z}$-linear (resp. $\mathbb{N}$-linear) subsets, i.e, of subsets of the form $x_0+\mathbb{N}x_1+\cdots+\mathbb{N}x_n$ for some $n \geq 0$ and some $x_0, x_1, \ldots x_n \in \mathbb{Z}^k$ (resp. $\mathbb{N}^k$)*
2. *X is a finite union of $\mathbb{Z}$-simple (resp. $\mathbb{N}$-simple) subsets, i.e, of subsets of the form $x_0 + \mathbb{N}x_1 + \cdots + \mathbb{N}x_n$ for some $n \geq 0$ and some $x_0, x_1, \ldots x_n \in \mathbb{Z}^k$ (resp. $\mathbb{N}^k$) which are linearly independent as $\mathbb{Q}$-vectors.*
3. *X is first order definable in the structure $\langle \mathbb{Z}, =, <, +, 0, 1\rangle$ (resp. $\langle \mathbb{N}, =, <, +, 0, 1\rangle$)*

Furthermore, there exists a procedure which converts one form into another

2.1 The case $\mathcal{Z}_{\mathbf{thresh+mod}}$

The $\mathcal{Z}_{\text{thresh}+\text{mod}}$-definable relations have a simple decomposition in terms of norm one vectors. Here is a precise definition. We consider the set $\{0, 1, -1\}^k \cap (\mathbb{Z}^k - \{0\}^k)$ of (L_∞-) norm one vectors and we provide it with a partial ordering by writing $e \geq f$ if the following condition holds: if the i-th component of f is nonzero, then f and e have the same i-th component. These vectors are not linearly independent, however we have the disjoint union

$$\mathbb{Z}^k = \bigcup_E \left(\sum_{e \in E} (\mathbb{N} - \{0\}) e \right) \tag{1}$$

where E ranges over all strictly decreasing sequences of norm one vectors and where by convention the empty sequence represents the null vector.

The decomposition mentioned above involves the family of recognizable relations in $\mathbb{N}^k$ which is an important subfamily of the Presburger definable relations. We recall its definition.

Definition 1. *A subset $X \subseteq \mathbb{N}^k$ is* recognizable *if it is a finite union of direct products such as*

$$X_1 \times \cdots \times X_k$$

where for $i = 1, \ldots, k$, X_i is a finite union of arithmetic sequences, i.e., sequences of the form $\{a + kp \mid k \in \mathbb{N}\}$ for some $a, b \in \mathbb{N}$.

We are now able to state the main characterization of the family of $\mathcal{Z}_{\text{thresh}+\text{mod}}$-definable relations on which the decidability result is based: indeed, recognizability in $\mathcal{Z}$ is decidable as proved in [4, Corollary 4.5].

Theorem 2. *A relation $X \subseteq \mathbb{Z}^k$ is definable in $\mathcal{Z}_{\text{thresh}+\text{mod}}$ if and only if it is a finite union of relations of the form*

$$X_1 e_1 + \cdots + X_p e_p$$

where $0 < p \leq k$, $e_1 > \cdots > e_p$ is a strictly decreasing sequence of norm one vector and $X_1 \times \cdots \times X_p$ is recognizable.

2.2 The case $\mathcal{Z}^{\boldsymbol{W}}$

We proceed top-down. We are given a subset in $\mathbb{Z}^k$ defined by a $\mathcal{Z}$-formula with k free variables and we transform it into a finite union of simple sets as asserted in Theorem 1. Call the integer n appearing in the expression of the simple set its *dimension* and observe that n is at most equal to k. By applying a simple geometric transformation of the space, we show that we can always assume that this union possesses at least one element of the dimension of the space, i.e., without loss of generality equal to k. The crux of the proof is the following result.

Theorem 3. *Let $X \subseteq \mathbb{Z}^k$ be a $\mathcal{Z}$-definable set of the form $X = T \cup \bigcup_{i=1}^{m} Y_i$ where $Y_i = a^{(i)} + \sum_{j=1}^{k} \mathbb{N} b_j^{(i)}$ for some linearly independent vectors $b_j^{(i)}$ (i.e. it is a simple set of dimension k) and T is a finite union of $\mathbb{N}$-simple sets of dimension less than k. Then X is $\mathcal{Z}^W$-definable if, and only if, it can be decomposed as $S \cup (P \setminus R)$, where:*

1. *$P = \bigcup_{1 \leq i \leq m} (a^{(i)} + \sum_{j=1}^{k} \mathbb{Z} b_j^{(j)})$;*
2. *R and S are $\mathcal{Z}^W$-definable sets which are included in a finite union of $\mathbb{Z}$-simple sets of dimension less than k;*
3. *$R \subseteq P$ and $S \cap P = \emptyset$.*

Indeed, given X, we can compute P, test equality *1* and obtain $S = X \setminus P$ and $R = P \setminus X$. The result applies recursively to R and S.

References

1. C. Choffrut. Deciding whether a relation defined in Presburger logic can be defined in weaker logics. submitted.
2. C. Choffrut and A. Frigeri. Definable sets in weak Presburger arithmetic. submitted.
3. S. Eilenberg and M.-P. Schützenbeger. Rational sets in commutative monoids. *Journal of Algebra*, 13(2):173–191, 1969.
4. S. Ginsburg and E.H. Spanier. Bounded regular sets. In *Proc. of the Amer. Math. Soc. 17*, pages 1043–1049, 1966.
5. M. Koubarakis. The complexity of query evaluation in indefinite temporal constraint databases. *Theor. Comput. Sci.*, 171(1-2):25–60, 1997.
6. P. Péladeau. Logically defined subsets of $\mathbb{N}^k$. *Theoret. Comput. Sci.*, 93:169–193, 1992.
7. J. Sakarovitch. *Eléments de théorie des automates.* Vuibert Informatique, 2003.
8. C. Smoryński. *Logical Number Theory I: An Introduction.* Springer Verlag, 1991.

Symbolic Reachability in Boolean Programs

Javier Esparza

Institut fuer Informatik (I7)
Technische Universitaet Muenchen
Germany

Abstract. One of the problems faced by Software Model Checking is the complexity of the control-flow mechanisms of modern programming languages. The combination of recursion and thread creation makes languages Turing-powerful even in the absence of data. In particular, the reachability of a program point is already undecidable for programs without variables!

In this talk I model control-flow primitives using term rewriting systems, and survey some results on their associated reachability and symbolic reachability problems. The symbolic reachability problem (informally stated) consists of computing a useful finite representation of the set of forward or backward reachable states.

On Counter Machines, Diophantine Equations, and Reachability Problems [⋆]

Oscar H. Ibarra[1], Zhe Dang[2], and Linmin Yang[2]

[1]Department of Computer Science
University of California
Santa Barbara, CA 93106, USA

[2]School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164, USA

Abstract. We give a brief survey of results that use "reversal-bounded counters" in studying reachability problems for various classes of transition systems. We also discuss the connection between the decidability of reachability in counter machines and the solvability of certain quadratic Diophantine equations. Finally, we report on some preliminary results concerning the emptiness problem for "stateless" multihead two-way (resp., one-way) nondeterministic finite automata.

1 Introduction

The verification problem is to decide whether a machine (representing a system) satisfies a desired property. Seeking algorithmic solutions to the problem is very important in many areas of computer science, since the machine can be interpreted as a protocol, a chip, a software design, etc. Even in emerging and nonclassical (the so called unconventional or natural) models of computing such as membrane and DNA computing systems, verification problems are still an important topic to pursue; e.g., to predict whether a cell system modeled as a P system [22] will evolve into a desired configuration.

There is a natural connection between automata theory and verification problems. Automata theory tries to answer questions concerning machine models, languages, and their decidability properties. An automaton defines an (abstracted) transition system, and the language accepted by the automaton defines a set of expected sequence of events (an input symbol is interpreted as an event) for the transition system. In this way, the automaton is a natural abstraction of a system, and the language represents the system's requirement (the property that the system is intended to satisfy). Therefore, in principle, automata theory is useful in asserting whether a particular class of transition systems is decidable for a certain class of verification queries like reachability.

[⋆] The work of Oscar H. Ibarra was supported in part by NSF Grants CCF-0430945 and CCF-0524136. The work of Zhe Dang and Linmin Yang was supported in part by NSF Grant CCF-0430531.

Integer variables are the most common unbounded storage devices in modeling infinite state transition systems (such as the size of the sliding window in the Transmission Control Protocol (TCP), the accumulated delay for an event in a discrete timed automaton [1], etc.). Such variables, in automata theory, are further modeled as counters, each of which can store a nonnegative integer number and can be incremented/decremented by one and tested against zero. Since two-counter machines (i.e., Minsky machines) are Turing-complete, it is important to study under what restrictions a multicounter machine has decidable reachability. One such restriction is to require that a counter be reversal-bounded (i.e., the number of alternations between nondecreasing mode and nonincreasing mode and vice-versa is bounded by a fixed integer independent of the computation) [12].

In this paper, we provide a survey on some of the results that use reversal-bounded counters in studying reachability problems for various classes of transition systems. It is known that a reversal-bounded multicounter machine with a one-way input tape accepts a semilinear language, even when the machine is augmented with a pushdown stack [12]. This result is already useful in showing that the reachability relation of nondeterministic pushdown machine with reversal-bounded counters is also semilinear and hence, a certain class of verification queries like reachability and safety are decidable. This result has also been used in verification problems for systems that apparently possess non-reversal-bounded counters (e.g., reachability in timed automata [5, 4]). When a reversal-bounded multicounter machine is equipped with a two-way input tape, however, it turns out that such a machine does not have a decidable emptiness problem even when it is deterministic and over a bounded language (i.e., the inputs are in the form $a_1^{i_1} \dots a_n^{i_n}$ for some fixed n and distinct symbols $a_1, \dots, a_n$, where $i_1, \dots, i_n$ are nonnegative integers) [12]. The case when the machine has only one reversal-bounded counter is interesting. More precisely, we use 2DCM(1) (resp. 2NCM(1)) to denote a deterministic (resp. nondeterministic) two-way finite automaton augmented with one reversal-bounded counter. It is known that the emptiness problem for 2DCM(1) over a bounded language is decidable [10]. This result was later generalized to 2DCM(1) (not necessarily over a bounded language) [15]. For the nondeterministic counterpart, it has been shown that the emptiness problem for 2NCM(1) over a bounded language is decidable [6]. However, in general, the emptiness problem for 2NCM(1) is still an open problem.

A surprisingly complex case is when a two-way finite automaton is augmented with monotonic (i.e., nondecreasing) counters. Notice that a monotonic counter is essentially a reversal-bounded counter making 0 reversal. Such a machine is called 2FACM [13]. We are interested in the following reachability problem: Given a 2FACM M, a state q, and a Presburger relation E over counter values, is there $(i_1, \dots, i_n)$ such that M, when started in its initial state on the left end of the input $a_1^{i_1} \dots a_n^{i_n}$ with all counters initially zero, reaches some configuration where the state is q and the counter values satisfy E? One can show that when M always halts, the problem is decidable. However, when M does not always halt, the problem can be reduced to the solvability of a special class of systems of quadratic Diophantine equations. In turn, reversal-bounded arguments can be used to establish new classes of solvable Diophantine equation systems [14, 24].

Biologically inspired computing models like P systems [22] are often stateless. This is because it is difficult and even unrealistic to maintain a global state for a massively parallel group of objects. The most common form of stateless counter machines are probably the Vector Addition Systems (VASs), which are well-studied. Indeed, VASs have been shown intimately related to certain classes of P systems [16]. Clearly, it is of interest to study other models of stateless automata. In this paper, we mention briefly some of our preliminary results concerning stateless multihead two-way (resp., one-way) nondeterministic finite automata.

2 Counter Machines

Let k be a nonnegative integer. A k-counter machine is a two-way nondeterministic finite automaton with input end markers (two-way NFA) augmented with k counters, each of which can be incremented by 1, decremented by 1, and tested for zero. We assume, w.l.o.g., that each counter can only store a nonnegative integer, since the sign can be stored in the states. If r is a nonnegative integer, let 2NCM(k,r) denote the class of k-counter machines where each counter is r *reversal-bounded*; i.e., it makes at most r alternations between nondecreasing and nonincreasing modes in any computation; e.g., a counter whose values change according to the pattern 0 1 1 2 3 4 $\underline{4\ 3}$ 2 1 $\underline{0\ 1}$ $\underline{1\ 0}$ is 3-reversal, where the reversals are underlined. For convenience, we sometimes refer to a machine in the class as a 2NCM(k,r).

A 2NCM(k,r) is *finite-crossing* [9, 12] if there is a positive integer d such that in any computation, the input head crosses the boundary between any two adjacent cells of the input no more than d times. Note that a 1-crossing 2NCM(k,r) is a one-way nondeterministic finite automaton augmented with k r-reversal counters. 2NCM(k) will denote the union of 2NCM(k,r), $r = 1, 2, \ldots$. For deterministic machines, we use 'D' in place of 'N'. If M is a machine, $L(M)$ denotes the language it accepts. A language is *bounded* if it is a subset of $a_1^* a_2^* \ldots a_n^*$ for some fixed n and distinct symbols symbols $a_1, a_2, \ldots, a_n$.

In the following, we summarize the important results concerning reversal-bounded counter machines.

Theorem 1. *There is a* fixed r *such that the emptiness problem for 2DCM(2,r) over bounded languages is undecidable [12].*

Theorem 2. *The emptiness problem is decidable for the following classes:*
(a) 2DCM(1) [15].
(b) 2NCM(1) over bounded languages [6].
(c) 2NCM(k) over a unary alphabet, for every k [15].
(d) finite-crossing 2NCM(k), for every k [9, 12].
(e) NPCM = nondeterministic one-way pushdown automata with reversal-bounded counters.

Let Y be a finite set of variables over integers. For all integers a_y, with $y \in Y$, b and c (with $b > 0$), $\sum_{y \in Y} a_y y < c$ is an *atomic linear relation* on Y and $\sum_{y \in Y} a_y y \equiv_b c$ is a *linear congruence* on Y. A *linear relation* on Y is a Boolean combination (using

$\neg$ and $\wedge$) of atomic linear relations on Y. A *Presburger formula* on Y is the Boolean combination of atomic linear relations on Y and linear congruences on Y. A set P of tuples of nonnegative integers is *Presburger-definable* or a *Presburger relation* if there exists a Presburger formula $\mathcal{F}$ on Y such that P is exactly the set of the solutions for Y that make $\mathcal{F}$ true. It is well known that Presburger formulas are closed under quantification.

Let N be the set of nonnegative integers and n be a positive integer. A subset S of N^n is a *linear set* if there exist vectors $v_0, v_1, ..., v_t$ in N^n such that $S = \{v \mid v = v_0 + a_1 v_1 + \ldots + a_t v_t, \forall 1 \leq i \leq t, a_i \in \mathrm{N}\}$. S is a *semilinear set* if it is a finite union of linear sets. It is known that S is a semilinear set if and only if S is Presburger-definable [8].

Let Σ be an alphabet consisting of n symbols $a_1, ..., a_n$. For each string (word) w in Σ^*, we define the *Parikh map* of w, denoted by $p(w)$, as follows:

$$p(w) = (i_1, ..., i_n), \text{ where } i_j \text{ is the number of occurrences of } a_j \text{ in } w.$$

If L is a subset of Σ^*, the *Parikh map* of L is defined by $p(L) = \{p(w) \mid w \in L\}$. L is a *semilinear language* if its Parikh map $p(L)$ is a semilinear set.

Theorem 3. *$p(L(M))$ is an effectively computable semilinear set when:*
(a) M is a finite-crossing 2NCM(k), for every k [9, 12].
(b) M is an NPCM [12].

Theorem 3 generalizes to the following: Let $\mathcal{M}$ be any class of machines such $p(L(M))$ is an effectively computable semilinear set for every machine M in $\mathcal{M}$. Then $p(L(M'))$ is an effectively computable semilinear set for every machine M' in $\mathcal{M}$ augmented with reversal-bounded counters. The result is not true for machines that are not finite-crossing. For example, a 2DCM(1,1) can recognize the language $\{0^i 1^j \mid i \text{ divides } j\}$, which is not semilinear.

The first part of the next theorem was shown in [12]. The second part is easily verified.

Theorem 4. *Let S be a subset of N^n and $L = \{a_1^{i_1} \ldots a_n^{i_n} \mid (i_1, \ldots, i_n) \in S\}$. If S is Presburger-definable (i.e., semilinear), then:*
(a) L can be accepted by a 1-crossing 2DCM(k) for some k.
(b) L can be accepted by a 2DCM(1).

3 Reachability and Safety

The results of the previous section can be used to analyze verification problems (such as reachability and safety) in infinite-state transition systems that can be modeled by multicounter machines. Decidability of reachability is of importance in the areas of model checking, verification, and testing [7, 3, 20]. In these areas, a machine is used as a system specification rather than a language recognizer, the interest being more in the behaviors that the machine generates.

In what follows, let M be a nondeterministic finite automaton (*with no input tape*) with a pushdown stack and k reversal-bounded counters. Let Γ be the pushdown alphabet and $\{1, 2, \ldots, s\}$ be the state set for some s . As usual, each counter can be incremented by integer constants $(+, -, 0)$ and can be tested $(<, >, =)$ to integer constants. Let $(j, v_1, ..., v_k, w)$ denote the configuration of M when it is in state j, counter i has value v_i for $1 \leq i \leq k$, and the pushdown stack contains word $w \in \Gamma^*$ (with the right-most bit of w being the top of the stack). Thus, the set of all possible configurations is a subset of $\mathrm{N}^{k+1} \times \Gamma^*$. We use the symbols $\alpha, \beta, \ldots$ to denote configurations. If α is a configuration, α^r will denote the configuration where the pushdown word w in α is written in reverse, i.e, w^r in α^r.

Given M, let $R(M) = \{(\alpha, \beta) \mid \alpha$ can reach β in 0 or more moves $\}$. $R(M)$ is called the *binary reachability* set of M. For a set S of configurations, define *post*$^*_M(S)$ to be the set of all successors of configurations in S; i.e., *post*$^*_M(S) = \{\alpha \mid \alpha$ can be reached from some configuration in S in 0 or more moves $\}$. Similarly, define *pre*$^*_M(S) = \{\alpha \mid \alpha$ can reach some configuration in S in 0 or more moves $\}$. *post*$^*_M(S)$ and *pre*$^*_M(S)$ are called the *forward* and *backward reachability* of M with respect to S, respectively.

Note that configuration $\alpha = (j, v_1, ..., v_k, w)$ in $\mathrm{N}^{k+1} \times \Gamma^*$ can be represented as a string $E(\alpha) = j\%v_1 \ldots \%v_k\%w$, where $j, v_1, ..., v_k$ are represented in unary (separated by a distinguished symbol %). Using this encoding, we can define the language $L_M = \{E(\alpha)\#E(\beta^r) \mid (\alpha, \beta) \in R(M)\}$ (# is a new symbol). Then we have:

Theorem 5. *L_M can be accepted by an NPCM (i.e. by a nondeterministic one-way pushdown automaton with reversal-bounded counters). It follows that the binary reachability set is semilinear.*

We say that a set $S \subseteq N^{k+1} \times \Gamma^*$ is NCM-recognizable if $E(S) = \{E(\alpha) \mid \alpha$ in $S\}$ can be accepted by an NCM. The problem of *safety* is of importance in the area of verification. The following theorem can be shown:

Theorem 6. *It is decidable to determine for a given nondeterministic finite automaton (with no input tape) with a pushdown stack and reversal-bounded counters M and two given sets of NCM-recognizable configurations S and T, whether every configuration in S can only reach configurations in T. Thus, safety is decidable.*

4 Diophantine Equations and Counter Machines

It is well-known that it is undecidable to determine, given an arbitrary multi-variable polynomial with integers coefficients, whether it has a solution over the nonnegative integers [18]. Some interesting nontrivial special cases have been shown to be decidable (e.g., [17]).

In this section we look at a class of systems of Diophantine equations for which the existence of solutions was shown to be decidable in [14] The decidability can be used to investigate certain verification problems concerning counter machines. We note that a similar class of equations was independently studied in [2].

4.1 A Solvable Class

For $1 \leq i \leq k$, let R_i denote $p_i(y)F_i + G_i$, where $p_i(y)$ is a polynomial in y with integer coefficients (e.g., $8y^5 - 3y^2 + 7y - 6$), and F_i, G_i are linear polynomials in $x_1, ..., x_n$ with integer coefficients (e.g., $-10x_1 + 5x_2 + 7x_4 - 3x_7 + 9$). Let $P(z_1, ..., z_k)$ be a Presburger relation over the nonnegative integers, i.e., definable by a Presburger formula.

Theorem 7. *The following problem is decidable:*

> **Given:** $R_1, ..., R_k$ *and a Presburger relation* P.
> **Question:** *Are there nonnegative integer values for* $y, x_1, ..., x_n$ *such that for these values,* $(R_1, ..., R_k)$ *satisfies* P*?*

Note that p_i, F_i, G_i can be a constant (in particular, 0 or 1). Hence, R_i can be just p_i, F_i, or G_i. In particular, R_i can be y or one of the x-variables.

The proof of the above theorem uses the following known results concerning reversal-bounded multicounter machines:

1. Let M be nondeterministic two-way finite automaton over a *unary* input (with left and right end markers) augmented with reversal-bounded counters. Thus the inputs to such a machine is of the form $\$o^y\$$, where y is a nonnegative integer, and \$ is the end marker. (A counter is reversal-bounded if at each step, it can be incremented/decremented by 1 or left unchanged and tested for zero, but can only reverse its mode from nondecreasing to nonincreasing and vice-versa at most a fixed number of times, independent of the input.) Call this machine 2NCM.

   It is known that the emptiness problem for 2NCMs (does the machine accept some unary input?) is decidable [15]. Note that, in contrast, it follows from Minsky's result [19], that the emptiness problem problem is undecidable for deterministic two-way finite automata over unary input (with end markers) augmented with one unrestricted counter.

2. Let G be a nondeterministic finite automaton with r nondecreasing counters (thus at each step, each counter can only be incremented by 0 or 1). G starts with all counters zero. If G halts and accepts, then we say that the values $(n_1, ..., n_r)$ of the counters when it halts are generated by G. G is called a monotonic counter generator.

   Let $P \subseteq N^r$. Then P is a Presburger relation or, equivalently, a semilinear set if and only if we can (effectively) construct a monotonic counter generator G such that the set of tuples generated by G is $\{(n_1, ..., n_r) \mid P(n_1, ..., n_r)$ is satisfied $\}$ [23].

We sketch the proof of above theorem, as given in [14].

Proof. Let G be the monotonic counter generator for the Presburger relation $P(z_1, ..., z_k)$. We show how to construct a 2NCM M to accept the unary language $\{o^y \mid y$ is a nonnegative integer, and there exist $x_1, ..., x_n$ such that for these values, $(R_1, ..., R_k)$ satisfies $P\}$. M operates as follows:

1. Given o^y, M first nondeterministically guesses $x_1, ..., x_n$ and stores them in k counters.
2. With o^y on the input and $(x_1, ..., x_n)$ on the counters, and using (many) auxiliary reversal-bounded counters, M computes $R_i = p_i(y)F_i + G_i$ for $i = 1, ..., k$.
3. Then M checks that $(R_1, ..., R_k)$ satisfies P, by simulating the generator G. (Note that in the simulation of the counters of G, an increment is simulated by a decrement.)

Items 1 and 3 above are obvious. That item 2 can be implemented by M using reversal-bounded counters follows from the the following observations:

1. Clearly, each $F_i(x_1, ..., x_n)$ and each $G_i(x_1, ..., x_n)$ can be computed using reversal-bounded counters.
2. $y^s x$ for any s and x can be computed using reversal-bounded counters as follows:

Suppose x is in counter 1 and o^y is on the input tape, M makes left-to-right (right-to-left) sweeps on the input while incrementing counter 2 on every move of the input head. At the end of every sweep, M decrements counter 1 by 1. When counter 1 becomes zero, counter 2 has value yx. Now that we have yx in counter 2, we can use the same idea to produce y^2x in counter 1, etc. Note that for a given s, $y^s x$ can be computed with counters 1 and 2 being reversal-bounded (each making no more than $s/2$ reversals).
Hence, the system has a solution if and only if the unary language accepted by M is not empty, which is decidable (since M is reversal-bounded).

4.2 Applications to Reachability Problems in Counter Machines

The question of the decidability of the class of Diophantine equations studied in Section 1 arose when we were studying some verification problems concerning counter machines in [13]. The decidability of one such property turned out to be equivalent to the decidability of the class in Section 1, where the $p_i(y)$'s are linear polynomials in y.

Consider a class of machines M as follows. M is a *deterministic* two-way finite automaton augmented with k monotonic counters $C_1, \ldots, C_k$. The two-way input, w (which is provided with left and right end markers), comes from a bounded language, i.e., $w = a_1^{i_1} \ldots a_n^{i_n}$ for some fixed n and distinct symbols $a_1, \ldots, a_n$, and $i_1, \ldots, i_n$ are nonnegative integers. The counters are initially zero and can be incremented by 0 or 1 at each step, but cannot be decremented. They do not participate in the dynamic of the machine. We shall simply call M a 2FAMC. We do not assume that the machine halts on all inputs.

Note that the set of tuples of nonnegative integers "generated" by a 2FACM (at a specified state) need not be semilinear (Presburger) in general. For example, consider a 2FACM with two monotonic counters C_1 and C_2. On unary input x of length n, M initially stores n in C_1. Then M makes left-to-right and right-to-left sweeps of the input, adding n to C_2 after every left-to-right sweep. M iterates this process without halting. Let s be the state of M just after a left-to-right sweep. Then the set of tuples of values of the counters when it is in state s is $Q_s = \{(n, kn) \mid n \geq 0, k > 0\}$, which is not semilinear.

We are interested in the problem of deciding, given a 2FAMC M and a Presburger relation E, whether the set of tuples generated by M satisfies E. The decidability of this question was left unresolved in [13], even for simple Presburger relations.

An atomic equality relation on the counters is a relation of the form $C_i = C_j, i \neq j$. An equality relation E is a conjunction of atomic equality relations. An example of E is $(C_1 = C_3 \ \wedge \ C_1 = C_4 \ \wedge \ C_2 = C_3)$. We say that M satisfies E at state q if there is some input $w = a_1^{i_1} \dots \ a_n^{i_n}$ such that M on input w, enters some configuration where the state is q and the counter values satisfy the relation E. For convenience, when q is understood, we simply say M satisfies E.

Since M does not necessarily halt, a configuration that satisfies E can be an intermediate configuration of a possibly infinite computation. Note also that M can satisfy E many times during the computation. We are interested in the following:

Reachability Problem Under Equality Relation:

Given: A 2FAMC M and an equality relation E.
Question: Does M satisfy E?

An obvious generalization of the above problem is when E is an arbitrary Presburger relation (note that an equality relation is a special form of a Presburger relation).

It turns out that the above problem (where E is an equality relation) is equivalent to the solvability of the following:

Problem DE:

Given: A system S consisting of the following $(k + m)$ equations:
$A_i = B_i, \ i = 1, \dots, k$
$yF_i = G_i, \ i = 1, \dots, m$

where A_i, B_i, F_i, G_i are linear polynomials in nonnegative integer variables $x_1, \dots, x_n$ with integer coefficients. Hence these polynomials are of the form $a_0 + a_1x_1 + \dots + a_nx_n$, where each a_i is an integer (positive, negative, or zero).

Question: Does S have a solution, i.e., there are nonnegative integers $y, x_1, \dots, x_n$ satisfying S?

Note that the system S above is a special case of the system we considered in Section 1. Hence Problem DE is decidable. The following was shown in [13]:

Theorem 8. *The Reachability Problem under equality relation is equivalent to Problem DE in the following sense:*

1. *Given a system S, we can effectively construct a 2FAMC M and an equality relation E such that S has a solution if and only if M satisfies E.*
2. *Given a 2FAMC and an equality relation E, we can effectively construct a finite number of systems S of equations of the form $A_i = B_i$ or of the form $yF_i = G_i$ (where A_i, B_i, F_i, G_i are linear polynomials in nonnegative integer variables $x_1, \dots, x_n$ with integer coefficients) such that M satisfies E if and only if one of the S's has a solution in the nonnegative integers $y, x_1, \dots, x_n$.*

The following corollary follows from Theorems 7 and 8:

Corollary 1. *The Reachability Problem for 2FMACs under equality relation is decidable.*

Reachability Problem Under Arbitrary Presburger Relation

Corollary 1 only proved the Reachability Problem when E is an equality relation. In this section, we look at the general case when the relation E is an arbitrary Presburger relation $E(c_1, ..., c_k)$ over the counter values $c_1, ..., c_k$, and show that the reachability problem is still decidable, resolving a more general open problem that was also posed in [13].

The idea is as follows. In [13], it was shown that the value of counter c_i $(1 \leq i \leq k)$ at any time can effectively be represented by equations of the form:
$c_i = A_i + yB_i + C_i$
where y is a nonnegative integer variable, and A_i, B_i, C_i are nonnegative linear polynomials in some nonnegative integer variables $x_1, ..., x_n$. (We can combine A_i and C_i into a single linear polynomial, but we wanted to first show the representation above to be consistent with the formulation in [13].) Then, the value of counter $c_i = yB_i + (A_i + C_i)$. It then follows from Theorem 7 that we can decide whether for this system of equations there exist nonnegative integers $y, x_1, ..., x_n$ such that $(c_1, ..., c_k)$ satisfies the Presburger relation E. Thus, we have:

Theorem 9. *The Reachability Problem for 2FAMCs under arbitrary Presburger relation is decidable.*

4.3 Variations

Now consider the following system of equations, S, which we call a width-2 system:

$$A_i = B_i, \;\; i = 1, \ldots, k$$
$$y_i F_i = G_i \; \wedge \; y_i H_i = I_i, \;\; i = 1, \ldots, m$$

where $A_i, B_i, F_i, G_i, H_i, I_i$ are linear polynomials in nonnegative variables $x_1, \ldots, x_n$. Note that each y_i is involved in exactly two equations. We say that S has a solution if there are nonnegative integers $y_1, \ldots, y_m, x_1, \ldots, x_n$ satisfying S. It was shown in [13], using an intricate reduction to Hilbert's Tenth Problem, that it is undecidable to determine, given a width-2 system S, whether it has a (nonnegative integer) solution in $y_1, ..., y_m, x_1, ..., x_n$.

A *semi-equality relation* E is a set of equality relations. Thus, $E = \{E_1, \ldots, E_m\}$, where each E_i is an equality relation (i.e., a conjunction of atomic equality relations). Note that when $m = 1$, E is simply an equality relation. E is of *width* k if each E_i is a conjunction of at most k atomic equality relations. Given M, a semi-equality relation $E = \{E_1, \ldots, E_m\}$, and states $q_1, \ldots, q_m$ (not necessarily distinct), M satisfies E if there is a tuple $(i_1, \ldots, i_n)$ such that M on input $(i_1, \ldots, i_n)$ has a computation where the counter values satisfy each E_i at state q_i during the computation (not necessarily at the same time, and not necessarily in the given order). To avoid writing so many subscripts, when $\{q_1, \ldots, q_m\}$ is understood, we simply say M satisfies E. The connection of width-2 systems to 2FAMCs is given by the following result shown in [13]

Theorem 10. *Given a width-2 system S, we can effectively construct a 2FAMC M and a width 2 semi-equality relation E such that S has a solution if and only if M satisfies E. Conversely, given a 2FAMC M and a width 2 semi-equality relation E, we can effectively construct a finite number of width 2 systems S such that M satisfies E if and only if one of the S's has a solution.*

Next, consider the following simpler case, called width-1 system, S:

$$A_i = B_i, \;\; i = 1, \dots, k$$
$$y_i F_i = G_i, \;\; i = 1, \dots, m$$

where, again, A_i, B_i, F_i, G_i are linear polynomials in $x_1, \dots, x_n$. Thus, each y_i is involved in only one equation. It known [13] that it is decidable to determine, given a width-1 system, whether it has a solution.

Theorem 11. *Given a width-1 system S, we can effectively construct a 2FAMC M and a width 1 semi-equality relation E such that S has a solution if and only if M satisfies E. Conversely, given a 2FAMC M and a width 1 semi-equality relation E, we can effectively construct a finite number of width-1 systems S such that M satisfies E if and only if one of the S's has a solution.*

5 Nondeterministic 2FAMC

In this section, we consider the reachability problem when the 2FAMC is nondeterministic. The results we discuss below were shown in [13]. We also reproduce the proofs as they appeared in [13]. We begin with the following theorem.

Theorem 12.
1. *There is a fixed k such that it is undecidable to determine, given a nondeterministic 2FAMC M with k monotonic counters and an equality relation E, whether M satisfies E. The result also holds for the case when M always halts.*
2. *It is decidable to determine, given a nondeterministic 2FAMC M over a unary input alphabet (but no restriction on the number of monotonic counters) and a Presburger relation E, whether M satisfies E.*

Proof. We first prove Part 1. From Theorem 1, there is a fixed r such the emptiness problem for 2DCM(2,r) over bounded languages is undecidable. Let M be such an automaton. Clearly, we can convert M to an equivalent automaton M' which has $2(1 + \frac{r-1}{2})$ 1-reversal counters where each counter starts at zero, and on input w, M' accepts if and only if it halts with all counters zero. To insure this, we can add to M' a dummy counter which is incremented at the beginning and only decremented, i.e., becomes zero when the input is accepted. Suppose M' has k 1-reversal counters, $C_1, ..., C_k$ (one of these is the dummy counter). Note that k is fixed since r is fixed. We modify M' to a nondeterministic 2FAMC M'' with $2k$ monotonic counters: $C_1^+, C_1^-, ..., C_k^+, C_k^-$, where C_i^+ and C_i^- are associated with counter C_i. M'' on input w simulates the computation of M' , first using counter C_i^+ to simulate C_i when the latter is in a nondecreasing mode. When counter C_i reverses (thus entering the nonincreasing mode), M'' continues the simulation but using counter C_i^-: incrementing this counter when M' decrements C_i. At some time during the computation (which may be different for each i), M' guesses that C_i has reached the zero value. From that point on, M'' will no longer use counters C_i^+ and C_i^-, but continues the simulation. When M'' has guessed that $C_i^+ = C_i^-$ for all i's (note that for sure the two counters corresponding to the dummy counter are equal), it halts in a unique state f. Clearly, w is accepted by M' if and only if M'' on w can reach a configuration with $C_i^+ = C_i^-$ for $i = 1, ..., k$ (this is relation E) in state f.

For Part 2, given an 2FAMC M with monotonic counters $C_1, ..., C_k$, we construct a 2NCM(1) (i.e., two-way nondeterministic finite automaton with reversal-bounded counters) over a unary input M'. M' simulates M faithfully. At some point, M' guesses that the values of the monotonic counters satisfy the relation E. M' then uses another set of counters to verify that this is the case, and accepts. The result follows from the decidability of the emptiness problem for 2NCM(1) over unary input (Theorem 2(c)) and and the fact that a Presburger relation on $C_1, ..., C_k$ can be verified using additional reversal-bounded counters (Theorem 4(a)).

□

In Theorem 12, Part 2, the relation E is defined over monotonic counters $C_1, \ldots, C_k$ and a specified state. In fact, the theorem still holds when the relation is defined over $C_1, \ldots, C_k, i_1, \ldots, i_n$, the input head position, and the state. This is because $n+2$ additional monotonic counters can be added to "store" the values of $i_1, \ldots, i_n$, the input head position, and the state at the time when the test for E is performed. The next result looks at the case when $k = 1$.

Theorem 13. *The reachability problem is decidable when M is a nondeterministic 2FAMC and $k = 1$, i.e., there is only one monotonic counter. The result holds even when the relation E is a Presburger relation that involves the state, the input head position, the input $(i_1, \ldots, i_n)$, and the monotonic counter.*

Proof. Given M, we construct a 2NCM(1) (i.e., a two-way nondeterministic finite automaton with one reversal-bounded counter) M' which accepts a nonempty language if and only if M satisfies the relation E. Let $\Sigma = \{a_1, \ldots, a_n\}$ be the input alphabet of M. The input alphabet of M' is $\Delta = \{a_1, \ldots, a_n, \diamond, b, c, d\}$, where $\diamond, b, c, d$ are new symbols. Given input y, M' first checks that y has exactly one occurrence of $\diamond$, and y with this $\diamond$ deleted is a string of the form $a_1^{i_1} \ldots a_n^{i_n} b^j c^k d^m$. M' then simulates M faithfully on $a_1^{i_1} \ldots a_n^{i_n}$, using its reversal-bounded counter to simulate the monotonic counter of M. During the simulation, when M' sees $\diamond$, it nondeterministically either ignores it and continues the simulation, or enters the testing phase. When M' decides to enter the testing phase (thus its input head is on $\diamond$), it checks the following: (1) the number j of b's is the input head position (i.e., the distance of $\diamond$ from the left end of the input), (2) the number k of c's is the value of the counter, and (3) the number m of d's represents the current state. M' accepts the input y if $i_1, \ldots, i_n, j, k, m$ satisfy the Presburger relation E. Note that M' can check the Presburger relation by Theorem 4(b). Clearly, M' accepts a bounded language. The result follows since the emptiness problem for 2NCM(1)'s over bounded languages is decidable (Theorem 2(b)). □

It is open whether Theorem 13 holds when $k = 2$. But consider a nondeterministic machine M with k monotonic counters, $C_1, \ldots, C_k$, *ordered* in that for $2 \leq i \leq k$, when C_i is first incremented, $C_1, \ldots, C_{i-1}$ can no longer change in value. The following corollary generalizes Theorem 13:

Corollary 2. *It is decidable to determine, given a nondeterministic machine M with k monotonic ordered counters and an arbitrary Presburger relation E (over the state, the input head position, $i_1, \ldots, i_n$, and the k counters), whether M satisfies E.*

Proof. Since the k counters are ordered, it is straightforward to generalize the construction in the proof of Theorem 13. M' will now have alphabet $\Delta = \{a_1, \ldots, a_n, \diamond_1, \ldots, \diamond_k, b, c_1, \ldots, c_k, d\}$. The marker $\diamond_i$ is used to remember the input head position when M goes from counter C_i to C_{i+1} ($1 \leq i < k$). As before, marker $\diamond_k$ is used to record the position of the input head prior to testing E. Symbol c_i is used to record the value of counter C_i. We leave the details to the reader. □

Remark: It is easy to check that Theorem 13, and Corollary 2 remain valid when the monotonic counter is replaced by a reversal-bounded counter.

Open Problem: In Theorem 12, Part 1, it would be interesting to find the smallest k for which the problem is undecidable (even the case $k = 2$ is open).

When E is a semi-equality relation, we have the following result in [13] which contrasts Theorem 11:

Theorem 14. *It is undecidable to determine, given a nondeterministic 2FAMC M and a width 1 semi-equality relation E, whether M satisfies E.*

Next, consider the following problem, where M is a 2NCM(1) over $a_1^* \ldots a_n^*$ and $E(x_1, \ldots, x_m)$ is a Presburger formula. We use $C[t]$ to denote the value of counter C at time t ("time" is used to count the total number of moves). Let $q_1, \ldots, q_m$ be given states. We say that M *m-satisfies* E if for some input $(i_1, \ldots, i_n)$, there is a computation of M such that for some $t_1 < \ldots < t_m$, $E(C[t_1], \ldots, C[t_m])$ is true and for each q_i ($1 \leq i \leq m$), M is at q_i when the current time is t_i. We show that this problem is decidable. Notice that this decidability does not simply follow from Corollary 2.

Theorem 15. *It is decidable to determine, given a 2NCM(1) M and a Presburger formula $E(x_1, \ldots, x_m)$, whether M m-satisfies E.*

Proof. We construct from M and E another 2NCM(1) M' over input alphabet $\{a_1, \ldots, a_n, b_1, \ldots, b_m, d_1, \ldots, d_m\}$. An input w to M' is valid if: (a) w has exactly one occurrence of d_k for each $k = 1, \ldots, m$; (b) w with the d_k's deleted results in a string of the form $a_1^{i_1} \ldots a_n^{i_n} b_1^{j_1} \ldots b_m^{j_m}$; (c) All the d_k's occur before the occurrence of any b_i in w. We refer to $d_1, \ldots, d_k$ as "markers".

M' first checks that input w is valid. Then it simulates the computation of M on $(i_1, \ldots, i_n)$ ignoring the markers. At some time t_1 during the computation, chosen nondeterministically, M' will be in some position within i_r. M' checks that the symbol directly to the right of this position is marker d_1 (and the state of M is q_1). (Note that M' may have seen this marker many times earlier but ignored it during the simulation until it decided that it has reached time t_1.) This marker is needed so that M' can return to this position after doing the following: M' moves its input head to the right and checks that j_1 is equal to the value of the counter C (if not, it halts and rejects). If it checks okay, then M' restores the value of the counter and moves its input head to marker d_1. Then M' resumes the simulation of M, and in the same way nondeterministically guesses times $t_2, \ldots, t_m$ and checks that the values of counter C at these times are $j_2, \ldots, j_m$ (as well as the states are $q_2, \ldots, q_m$), respectively. Finally, M' verifies that $E(j_1, \ldots, j_m)$ is true.

Clearly, M m-satisfies $E(C[t_1], \ldots, C[t_m])$ for some $i_1, \ldots, i_n, t_1, \ldots, t_m$ if and only if M' accepts some input w. The result now follows since emptiness for 2NCM(1) over bounded languages is decidable by Theorem 2(b). □

When the times $t_1, \ldots, t_m$ are involved in the Presburger formula, i.e., we now have $E(x_1, \ldots, x_m, t_1, \ldots, t_m)$, then the above problem is undecidable.

Theorem 16. *It is undecidable to determine, given a 2NCM(1) M and a Presburger formula $E(x_1, \ldots, x_m, t_1, \ldots, t_m)$, whether M m-satisfies E.*

Proof. It is known that, in general, a system of quadratic Diophantine equations is unsolvable [18]. Hence, it is undecidable to determine, given an n and a Presburger formula

$$P(A_1, \ldots, A_n, A_1A_1, \ldots, A_1A_n, \ldots, A_iA_i, \ldots, A_iA_n, \ldots, A_{n-1}A_n, A_nA_n) \quad (1)$$

(note that we write P in terms of the A_i's and the A_iA_j's, $1 \leq i \leq j \leq n$), whether it has a nonnegative integer solution in $A_1, \ldots, A_n$. Now, we construct an m, a 2NCM(1) M and an E such that (1) has a solution if and only if M m-satisfies E. The result then follows.

M operates on input $(A_1, \ldots, A_n)$ in two phases as follows with the counter initially being 0. In the first phase, for each $i = 1, \ldots, n$, M increments the counter to A_i while reading the block of A_i (and enters the state q_i – we use t_i to denote the current time) and then decrements the counter to 0. As a result, $C[t_i] = A_i$ $(1 \leq i \leq n)$. Suppose that the state is now q_{n+1} and the current time is t_{n+1}. In the second phase, from $i = 1$ to n, M executes the following subroutine:

```
1. $j := i$;
2. Repeat below for 0 or more times (nondeterministically chosen):
   2.1. Read the segment of $A_i$ on the input from left to the right and back;
   2.2. Increment the counter by 1;
3. Decrement the counter to 0;
4. $j := j + 1$;
5. If $j > n$ then exit this subroutine else goto 2.
```

Let q_{ij} be the state of M when it is about to execute step 3. Note that each q_{ij} is visited only once during M's computation. We use T_{ij} to denote the time when M visits q_{ij} (for simplicity and without loss of generality, we only count the times spent in step 2 and step 3). Assume that $\wedge_{i \leq j} C[T_{ij}] = A_j$. Under this assumption, the loop in step 2 must be repeated for A_j times for each i and j. For $1 \leq i \leq n$ and $i \leq j < n$, $T_{i(j+1)} - T_{ij} = A_j + (2A_i + 1)A_{j+1}$. This is because, between time T_{ij} and time $T_{i(j+1)}$, M executes step 3 (takes A_j time units) and executes the loop in step 2 for

A_{j+1} times (each loop takes $2A_i$ time units in step 2.1 and one time unit in step 2.2). Similarly, for $1 \leq i < n$, $T_{(i+1)(i+1)} - T_{in} = A_n + (2A_{i+1} + 1)A_{i+1}$, and $T_{11} = (2A_1 + 1)A_1 + t_{n+1}$ (t_{n+1} is the time when the second phase started). Therefore, each A_iA_j, $i \leq j$, can be expressed as a linear combination (with positive, negative, zero coefficients) of the T_{ij}'s and A_i's. Hence, combining the result of phase 1, each A_iA_j as well as each A_i can be expressed as a linear combination of the T_{ij}'s, $C[t_i]$'s, and t_{n+1}. Substituting each A_iA_j and each A_i in the conjunction of the assumption and (1) with the linear combinations representing them, it is not hard to obtain an E with $m = (n+1) + \frac{n(n+1)}{2}$ as required. Note that M is $m - 1$ reversal bounded. □

6 Stateless Automata

In this section, we briefly mention some recent results we have obtained concerning "stateless automata" (that will be reported in full elsewhere). These automata are useful in analyzing some problem in membrane computing (P systems).

A *stateless multihead two-way NFA* M is equipped with an input on alphabet Σ and heads $H_1, \ldots, H_k$ for some k. The heads are two-way, the input is read-only, and there are no states. An H_i-move (also called a *local move*) MOVE$_i$ of the NFA can be described as a triple (H_i, a, D), where H_i is the head involved in the move, a is the input symbol under the head H_i, and $D \in \{R, L, S\}$ meaning that, as a result of the move, the head H_i goes to the right, goes to the left, or simply stays. When a head H_i tries to execute a local move (H_i, a, D), it requires that the symbol under H_i must be a, otherwise M just crashes. A generalized move is in the form of $(H_i, \mathcal{S}, \mathcal{D})$, where $\mathcal{S}$ is a set of symbols, and $\mathcal{D}$ is a set of directions (i.e., R, L, S). When executing a generalized move $(H_i, \mathcal{S}, \mathcal{D})$, the symbol H_i reads must belong to $\mathcal{S}$, and H_i nondeterministically picks up a direction from $\mathcal{D}$.

Note that a local move is a special case of a generalized move. An *instruction* of M is a sequence of local or generalized moves, in the form of [MOVE$_{i_1}$, MOVE$_{i_2}$, . . ., MOVE$_{i_m}$], for some m, $1 \leq m \leq k$, and $i_1 < \ldots < i_m$. (If $m = 1$, the instruction is simple called a *local* instruction.) When an instruction is executed, the heads $H_{i_1}, \ldots, H_{i_m}$ perform the moves MOVE$_{i_1}$, . . ., MOVE$_{i_m}$, respectively and simultaneously. Any head falling off the tape will cause M to crash. The NFA M has a finite set of such instructions, and each time, nondeterministically picks an instruction to execute. M has a distinguished (accept ing) heads $F \subseteq \{H_1, ..., H_k\}$. Initially, all heads are at the leftmost cell of the input tape. *M halts and accepts* the input when the designated heads are all at the rightmost cell. We assume that the input tape of M has a left end marker $\triangleright$ and a right end marker \$. Thus, for any input $a_1...a_n$, $n \geq 2$, $a_1 = \triangleright$, $a_n = \$$, and for $2 \leq i \leq n-1$, each a_i is different from the end markers.

Theorem 17. *The emptiness problem for stateless multihead two-way NFAs is undecidable. The problem becomes decidable when the NFAs have only local instructions.*

Actually, we can show that the above theorem holds even if M has only one-way heads:

Theorem 18. *The emptiness problem for stateless 3-head one-way NFAs is undecidable.*

We don't know whether the result above holds for NFAs with only two one-way heads.

Next, we consider the case when the input is restricted. Recall that a language is bounded if it is a subset of $a_1^* a_2^* \ldots a_k^*$ for some given symbols $a_1, \ldots, a_k$.

It is known [11] that if M is a multihead one-way NFA with states but with bounded input, the language it accepts is a semilinear set effectively constructable from M. In fact, this result holds, even if M has two-way heads, but the heads can only reverse directions from right to left or from left to right at most r times, for some fixed r independent of the input. It follows that Theorem 18 can not be strengthen to hold for one-way machines accepting bounded languages. However, for two-way machines, by using the unsolvability of Diophantine systems, we can prove the following:

Theorem 19. *The emptiness problem for stateless multihead two-way NFAs remains undecidable even when the input is bounded.*

A special case of Theorem 19 is when the input is unary and without end markers. In this case, the heads are initially at the leftmost input cell and the automaton accepts when the heads are all at the rightmost cell (we assume that there are at least two cells on the input). Using a VAS to simulate the multihead position changes, one can show:

Theorem 20. *The emptiness problem for stateless multihead two-way NFAs is decidable when the input is unary and without end markers.*

However, the above theorem does not hold when we add a left end marker (resp., right end marker) to the unary input, i.e., the input is in the form $\triangleright a \ldots a$ (resp., $a \ldots a\$$), where the input is of length at least 2). The following theorem (whose proof is quite intricate) is an improvement of Theorem 19.

Theorem 21. *The emptiness problem for stateless multihead two-way NFAs is undecidable even when the input is unary but with the left end marker (resp., right end marker).*

7 Conclusion

In this paper, we gave a brief survey of results that use reversal-bounded counters in studying reachability problems for various classes of transition systems. We also discussed the connection between the decidability of reachability in counter machines and the solvability of certain quadratic Diophantine equations. Finally, we reported on some preliminary results concerning the emptiness problem for stateless multihead two-way (resp., one-way) nondeterministic finite automata.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. Marius Bozga, Radu Iosif, and Yassine Lakhnech. Flat parametric counter automata. In *ICALP (2)*, pages 577–588, 2006.

3. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 1998.
4. Z. Dang. Pushdown time automata: a binary reachability characterization and safety verification. *Theoretical Computer Science*, 302(1-3).
5. Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer, and J. Su. Binary reachability analysis of discrete pushdown timed automata. In *Proceedings of the International Conference on Computer Aided Verification (CAV 2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 2000.
6. Z. Dang, O. H. Ibarra, and Z. Sun. On the emptiness problems for two-way nondeterministic finite automata with one reversal-bounded counter. In *Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC 2002)*, volume 2518 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2002.
7. J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.
8. S. Ginsburg and E. Spanier. Semigroups, presburger formulas, and languages. *Pacific J. of Mathematics*, 16:285–296, 1966.
9. E. M. Gurari and O. H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Sciences*, 22:220–229, 1981.
10. E. M. Gurari and O. H. Ibarra. Two-way counter machines and diophantine equations. *Journal of the ACM*, 29(3):863–873, 1982.
11. O. H. Ibarra. A note on semilinear sets and bounded-reversal multihead pushdown automata. *Inf. Processing Letters*, 3(1): 25-28, 1974.
12. O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.
13. O. H. Ibarra and Z. Dang. On two-way fa with monotonic counters and quadratic diophantine equations. *Theor. Comput. Sci.*, 312(2-3):359–378, 2004.
14. O. H. Ibarra and Z. Dang. On the solvability of a class of diophantine equations and applications. *Theor. Comput. Sci.*, 352(1-3):342–346, 2006.
15. O. H. Ibarra, T. Jiang, N. Tran, and H. Wang. New decidability results concerning two-way counter machines. *SIAM Journal of Computing*, 24:123–137, 1995.
16. Oscar H. Ibarra, Zhe Dang, and Ömer Egecioglu. Catalytic p systems, semilinear sets, and vector addition systems. *Theor. Comput. Sci.*, 312(2-3):379–399, 2004.
17. L. Lipshitz. The diophantine problem for addition and divisibility. *Transactions of AMS*, 235:271–283, 1978.
18. Y. V. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.
19. M. Minsky. Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines. *Ann. of Math.*, 74:437–455, 1961.
20. P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. Int. Conf. on Computer-Aided Verification (CAV)*, Lecture Notes in Computer Science, pages 88–97. Springer, 1998.
21. Gh. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
22. Gh. Paun. *Membrane Computing, An Introduction*. Springer-Verlag, 2002.
23. O. H. Ibarra J. Karhumaki T. Harju and A. Salomaa. Some decision problems concerning semilinearity and commutation. *Journal of Computer and System Sciences*, 65:278–294, 2002.
24. Gaoyan Xie, Zhe Dang, and Oscar H. Ibarra. A solvable class of quadratic diophantine equations with applications to verification of infinite-state systems. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 668–680. Springer, 2003.

25. L. Yang, Z. Dang, and O.H. Ibarra. Bond computing systems: a biologically inspired and high-level dynamics model for pervasive computing. The 6th International Conference on Unconventional Computation (UC'07), Lecture Notes in Computer Science.

Is the Joint Spectral Radius of Rational Matrices Reachable by a Finite Product?[⋆]

Raphaël M. Jungers and Vincent D. Blondel[⋆⋆]

Department of Applied Mathematics, Université catholique de Louvain, 4 avenue Georges Lemaitre, B-1348 Louvain-la-Neuve, Belgium
{raphael.jungers,vincent.blondel}@uclouvain.be

Abstract. A set of matrices is said to have the finiteness property if the maximal rate of growth of long products of matrices taken from the set can be obtained by a periodic product. It was conjectured a decade ago that all finite sets of real matrices have the finiteness property. This "finiteness conjecture", is now known to be false but no explicit counterexample is available and in particular it is unclear if a counterexample with matrices having rational or binary entries is possible. In this paper, we prove that all finite sets of nonnegative rational matrices have the finiteness property if and only if *pairs* of *binary* matrices do, we state a similar result when negative entries are allowed, and we prove that all pairs of 2×2 binary matrices have the finiteness property. We also describe implications of our results for the stability problem for sets of matrices.

1 Introduction

The joint spectral radius of a set of matrices characterizes the maximum rate of growth that can be obtained by forming long products of matrices. Let $\Sigma \subset \mathbb{R}^{n\times n}$ be a finite set of matrices. The *joint spectral radius* of Σ is defined by:

$$\rho(\Sigma) = \limsup_{t\to\infty} \max\{\|A\|^{1/t} : A \in \Sigma^t\} \tag{1}$$

where Σ^t is the set of products of length t of matrices from Σ, i.e., $\Sigma^t = \{A_1 \dots A_t : A_i \in \Sigma\}$. It is easy to verify that the quantity $\rho(\Sigma)$ does not depend on the chosen matrix norm. It has been proved in [1] that the following equality holds for finite (or bounded) sets Σ:

$$\rho(\Sigma) = \limsup_{t\to\infty} \max\{\rho(A)^{1/t} : A \in \Sigma^t\}$$

[⋆] This is a shortened version of a journal paper submitted for publication; see [16].

[⋆⋆] The research reported here was performed while the authors were at MIT, Cambridge. It was partially supported by the "Communauté francaise de Belgique - Actions de Recherche Concertées", by the EU HYCON Network of Excellence (contract number FP6-IST-511368), and by the Belgian Programme on Interuniversity Attraction Poles initiated by the Belgian Federal Science Policy Office. The scientific responsibility rests with its authors. Raphaël Jungers is a FNRS fellow (Belgian Fund for Scientific Research).

(ρ is used here to denote the usual spectral radius). It is also known that the following inequalities hold for all t:

$$\max\{\rho(A)^{1/t} : A \in \Sigma^t\} \leq \rho(\Sigma) \leq \max\{||A||^{1/t} : A \in \Sigma^t\}. \tag{2}$$

These inequalities provide a straightforward way to approximate the joint spectral radius to any desired accuracy: evaluate the upper and lower bounds for products of increasing length t, until ρ is squeezed in a sufficiently small interval and the desired accuracy is reached. Unfortunately, this method, and in fact any other general method for computing or approximating the joint spectral radius, is bound to be inefficient. Indeed, it has been proved that, unless $P = NP$, there is no algorithm that computes or even approximates with a priori guaranteed accuracy the joint spectral radius of a set of matrices in polynomial time [5]. And this is true even if the matrices have binary entries.

For some sets Σ, the right hand side inequality in (2) is strict for all t. This is the case for example for the set consisting of just one matrix

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Thus, there is no hope to reach the exact value of the joint spectral radius by simply evaluating the right hand side in (2). On the other hand, since $\rho(A^k) = \rho^k(A)$ the left hand side always provides the exact value when the set Σ consists of only one matrix and one can thus hope to reach the exact value of the joint spectral radius by evaluating the maximal spectral radii of products of increasing length. If for some t and $A \in \Sigma^t$ we have $\rho(A)^{1/t} = \rho(\Sigma)$, then the value of the joint spectral radius is reached. Sets of matrices for which such a product is possible are said to have the finiteness property:

Definition 1. *A set Σ of matrices is said to have the* finiteness property *if there exists some product $A = A_1 \dots A_t$ with $A_i \in \Sigma$ for which $\rho(\Sigma) = \rho(A)^{1/t}$.*

One of the interests of the finiteness property arises from its connection with the stability question for a set of matrices. This problem is of practical interest in a number of application contexts. A set of matrices Σ is *stable* if all long products of matrices taken from the set converge to zero. There are no known algorithms for deciding stability of a set of matrices and it is unknown if this problem is algorithmically decidable. One can verify that stability of the set Σ is equivalent to the condition $\rho(\Sigma) < 1$ and we may therefore hope to decide stability as follows: for increasing values of t evaluate $\underline{\rho}_t = \max\{\rho(A)^{1/t} : A \in \Sigma^t\}$ and $\overline{\rho}_t = \max\{||A||^{1/t} : A \in \Sigma^t\}$. From (2) we know that $\underline{\rho}_t \leq \rho \leq \overline{\rho}_t$ and so, as soon as a t is reached for which $\overline{\rho}_t < 1$ we stop and declare the set stable, and as soon as a t is reached for which $\underline{\rho}_t \geq 1$ we stop and declare the set unstable. This algorithm will always stop unless $\rho = 1$ and $\underline{\rho}_t < 1$ for all t. But this last situation never occurs for sets of matrices that satisfy the finiteness property and so we conclude:

Proposition 1. *Stability is algorithmically decidable for sets of matrices that have the finiteness property.*

It was first conjectured in 1995 by Lagarias and Wang that all sets of real matrices have the finiteness property [13]. This conjecture, known as the finiteness conjecture, has attracted intense attention in recent years and has been proved to be false by several authors [4,7,12]. So far all proofs provided are nonconstructive and all sets of matrices whose joint spectral radius is known exactly satisfy the finiteness property. The finiteness property is also known to hold in a number of particular cases including the case were the matrices are symmetric, or if the Lie algebra associated with the set of matrices is solvable [20, Corollary 6.19]: in this case the joint spectral radius is simply equal to the maximum of the spectral radii of the matrices. This follows directly from a result of Liberzon et al. [14] after a conjecture of Gurvits [9].

The definition of the finiteness property leads to a number of natural questions: When does the finiteness property hold? Is it decidable to determine if a given set of matrices satisfies the finiteness property? Do matrices with rational entries satisfy the finiteness property? Do matrices with binary entries satisfy the finiteness property? In the first theorem in this paper we prove a connection between rational and binary matrices:

Theorem 1. *The finiteness property holds for all sets of nonnegative rational matrices if and only if it holds for all* pairs *of* binary *matrices.*

The case of binary matrices appears to be important in a number applications [8, 15, 17–19]. These applications have led to a number of joint spectral radius computations for binary matrices [10, 17, 18]. The results obtained so far seem to indicate that for binary matrices there is always an optimal infinite periodic product. Moreover, when the matrices have binary entries they can be interpreted as adjacency matrices of graphs on an identical set of nodes and in this context it seems natural to expect optimality to be obtained for periodic products. Motivated by these observations, the following conjecture is phrased in [3]:

Conjecture 1. Pairs of binary matrices have the finiteness property.

Of course if this conjecture is correct then nonnegative rational matrices also satisfy the finiteness property and this in turn implies that stability, that is, the question $\rho < 1$, is decidable for sets of matrices with nonnegative rational entries. From a decidability perspective this last result would be somewhat surprising since it is known that the closely related question $\rho \leq 1$ is not algorithmically decidable for such sets of matrices [2, 6].

Motivated by the relation between binary and rational matrices, we prove in our second theorem that pairs of 2×2 binary matrices satisfy the finiteness property. We have not been able to find a unique argument for all possible pairs and we therefore proceed by enumerating a number of cases and by providing separate proofs for each of them. This somewhat unsatisfactory proof is nevertheless encouraging in that it could possibly be representative of the difficulties arising for pairs of binary matrices of arbitrary dimension. In particular, some of the techniques we use can be applied to matrices of arbitrary dimension.

Let us finally notice that in all the numerical computations that we have performed on binary matrices not only the finiteness property always seemed to hold but the period length of optimal products was always very short. The computation of the joint spectral radius is known to be NP-hard for binary matrices but this does not exclude the possibility of a bound on the period length that is linear in the dimensions of the matrices. In the case of matrices characterizing the capacity of codes avoiding forbidden difference patterns, the length of the period is even suspected to be sublinear (see Conjecture 1 in [10]).

2 Finiteness property for rational and binary matrices

In this section, we show that the finiteness property holds for nonnegative rational matrices if and only if it holds for *pairs* of *binary* matrices. The proof proceeds in three steps. First we reduce the nonnegative rational case to the nonnegative integer case, we then reduce this case to the binary case, and finally we show how to reduce the number of matrices to two. For the sake of conciseness, the ideas of the proofs are rapidly sketched, while the precise reasoning can be found in the appendices.

Proposition 2. *The finiteness property holds for sets of nonnegative rational matrices if and only if it holds for sets of nonnegative integer matrices.*

Proof. From the definition of the joint spectral radius we have that for any $\alpha > 0$, $\rho(\Sigma) = (1/\alpha)\rho(\alpha\Sigma)$. Now, for any set Σ of matrices with nonnegative rational entries , let us pick an $\alpha \neq 0 \in \mathbb{N}$ such that $\alpha\Sigma \subseteq \mathbb{N}^{n\times n}$. If there exists a positive integer t and a matrix $A \in (\alpha\Sigma)^t$ such that $\rho(\alpha\Sigma) = \rho^{1/t}(A)$, then $\rho(\Sigma) = (1/\alpha)\rho^{1/t}(A) = \rho^{1/t}(A/\alpha^t)$, where $A/\alpha^t \in \Sigma^t$.

We now turn to the reduction from the integer to the binary case.

Theorem 2. *The finiteness property holds for sets of nonnegative integer matrices if and only if it holds for sets of binary matrices.*

Proof. Consider a finite set of nonnegative integer matrices $\Sigma \subset \mathbb{N}^{n\times n}$. We think of the matrices in Σ as adjacency matrices of weighted graphs on a set of n nodes and we construct auxiliary graphs such that paths of weight w in the original weighted graphs are replaced by w paths of weight one in the auxiliary graphs. For every matrix $A \in \Sigma \subset \mathbb{N}^{n\times n}$, we introduce a new matrix $\tilde{A} \in \{0,1\}^{nm\times nm}$ as follows. We define m as the largest entry of the matrices in Σ. Then, for every node v_i $(i = 1,\ldots,n)$ in the original graphs, we introduce m nodes $\tilde{v}_{i,1},\ldots,\tilde{v}_{i,m}$ in the auxiliary graphs. The auxiliary graphs have nm nodes; we now define their edges. For all $A \in \Sigma$ and $A_{i,j} = k \neq 0$, we define km edges in $\tilde{A}$ from nodes $\tilde{v}_{i,s} : 1 \leq s \leq k$ to the nodes $\tilde{v}_{j,t} : 1 \leq t \leq m$. The following reasoning leads now to the claim:

– For any product $A \in \Sigma^t$, and any couple of indices (i,j), the corresponding product $\tilde{A} \in \tilde{\Sigma}^t$ has the following property: $\forall s, A_{i,j} = \sum_r \tilde{A}_{\tilde{v}_{i,r},\tilde{v}_{j,s}}$. This is easy to show by induction on the length of the product.

– $\forall t, \forall A \in \Sigma^t,\ ||A||_1 = ||\tilde{A}||_1$, where $||\cdot||_1$ represents the maximum sum of the absolute values of all entries of any column in a matrix.
– $\rho(\Sigma) = \rho(\tilde{\Sigma})$, and if $\rho(\tilde{\Sigma}) = \rho^{1/T}(\tilde{A}) : \tilde{A} \in \tilde{\Sigma}^T$, then $\rho(\Sigma) = \rho^{1/T}(A)$, where A is the product in Σ^T corresponding to $\tilde{A}$.

We finally consider the last reduction: we are given a set of matrices and we reformulate the finiteness property for this set into the finiteness property for two particular matrices constructed from the set. The construction is such that the entries of the two matrices are exactly those of the original matrices, except for some entries that are equal to zero or one.

More specifically, assume that we are given m matrices $A_1, \ldots, A_m$ of dimension n. From these m matrices we construct two matrices $\tilde{A}_0, \tilde{A}_1$ of dimension $(2m-1)n$. The matrices $\tilde{A}_0, \tilde{A}_1$ consist of $(2m-1) \times (2m-1)$ square blocks of dimension n that are either equal to 0, I or to one of the matrices A_i. The explicit construction of these two matrices is best illustrated with a graph.

Consider the graph G_0 on a set of $2m-1$ nodes s_i $(i = 1, \ldots, 2m-1)$ and whose edges are given by $(i, i+1)$ for $i = 1, \ldots, 2m-2$. We also consider a graph G_1 defined on the same set of nodes and whose edges of weight a_i are given by $(m+i-1, i)$ for $i = 1, \ldots, m$. These two graphs are represented on Figure 1 for the case $m = 5$. In such a graph a directed path that leaves the node m returns there after m steps and whenever it does so, the path passes exactly once through an edge of graph G_1. Let us now describe how to construct the matrices $\tilde{A}_0, \tilde{A}_1$. The matrices are obtained by constructing the adjacency matrices of the graphs G_0 and G_1 and by replacing the entries 1 and 0 by the matrices I and 0 of dimension n, and the weight a_i by the matrices A_i. For the case $m = 5$ the matrices $\tilde{A}_0, \tilde{A}_1$ are thus given by:

$$\tilde{A}_0 = \begin{pmatrix} 0&I&0&0&0&0&0&0&0 \\ 0&0&I&0&0&0&0&0&0 \\ 0&0&0&I&0&0&0&0&0 \\ 0&0&0&0&I&0&0&0&0 \\ 0&0&0&0&0&I&0&0&0 \\ 0&0&0&0&0&0&I&0&0 \\ 0&0&0&0&0&0&0&I&0 \\ 0&0&0&0&0&0&0&0&I \\ 0&0&0&0&0&0&0&0&0 \end{pmatrix} \quad \tilde{A}_1 = \begin{pmatrix} 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ A_1&0&0&0&0&0&0&0&0 \\ 0&A_2&0&0&0&0&0&0&0 \\ 0&0&A_3&0&0&0&0&0&0 \\ 0&0&0&A_4&0&0&0&0&0 \\ 0&0&0&0&A_5&0&0&0&0 \end{pmatrix}$$

The two matrices so constructed inherit some of the properties of the graphs G_0 and G_1 and this allows us to prove the following theorem.

Theorem 3. *Consider a set of $m \geq 1$ matrices $\Sigma = \{A_1, \ldots, A_m : A_i \in \mathbb{R}^{n\times n}\}$ and define $\tilde{\Sigma} = \{\tilde{A}_0, \tilde{A}_1\}$ with the matrices $\tilde{A}_0$ and $\tilde{A}_1$ as defined above. Then $\rho(\tilde{\Sigma}) = \rho(\Sigma)^{1/m}$. Moreover, the finiteness property holds for Σ if and only it holds for $\tilde{\Sigma}$.*

Proof. The crucial observation for the proof is the following. Consider a path in G_0 and G_1. Edges in G_0 and G_1 have outdegree at most equal to one and so if

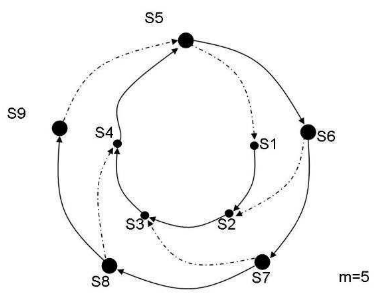


Fig. 1. Schematic representation of the macro transitions between subspaces. The full edges represent transitions in $\tilde{A}_0$ and the dashed edges transitions in $\tilde{A}_1$.

a sequence of graphs among G_0 and G_1 is given, there is only one path leaving i that follows that particular sequence. This fact ensures that any block in any product of matrices in $\tilde{\Sigma}$ is a pure product of blocks of the matrices in $\tilde{\Sigma}$, and not a sum of such products. Moreover, any path leaving from i and of length km either returns to i after passing through k edges of G_1, or ends at node $i+m$ after passing through $k-1$ edges of G_1, or ends at node $i+m \pmod{2m}$ after passing through $k+1$ edges of G_1. From this it follows that in a product of length km of the matrices $\tilde{A}_0$ and $\tilde{A}_1$ there is exactly one nonzero block in every line of blocks, and this block is a product of length $k-1$, k, or $k+1$ of matrices from Σ.

We now show that $\rho(\tilde{\Sigma}) \geq \rho(\Sigma)^{1/m}$ by proving that for any matrix $A \in \Sigma^t$, there is a matrix $\tilde{A} \in \tilde{\Sigma}^{tm}$ such that $||\tilde{A}|| \geq ||A||$. Define $\tilde{B}_i = \tilde{A}_0^{i-1}\tilde{A}_1\tilde{A}_0^{m-i} \in \tilde{\Sigma}^m$ for $i = 1, \dots, m$ so that the block in position (m,m) in $\tilde{B}_i$ is simply equal to A_i. Consider now some product of length t, $A = A_{i_1} \cdots A_{i_t} \in \Sigma^t$ and construct the corresponding matrix product $\tilde{A} = \tilde{B}_{i_1} \dots \tilde{B}_{i_t} \in \tilde{\Sigma}^{tm}$. The block in position (m,m) in $\tilde{A}$ is equal to $A_{i_1} \dots A_{i_t}$ and so $||\tilde{A}|| \geq ||A||$ and $\rho(\tilde{\Sigma}) \geq \rho(\Sigma)^{1/m}$.

Let us now show that $\rho(\tilde{\Sigma}) \leq \rho(\Sigma)^{1/m}$. Consider therefore an arbitrary product $\tilde{A} \in \tilde{\Sigma}^l$ and decompose $\tilde{A} = \tilde{C}\tilde{A}'$ with $\tilde{C}$ a product of at most m factors and $\tilde{A}' \in \Sigma^{km}$. By the observation above we know that there is at most one nonzero block in every line of blocks of $\tilde{A}'$, and this block is a product of length $k-1$, k, or $k+1$ of matrices from Σ. Therefore, if the norm is chosen to be the maximum line sum, we have $||\tilde{A}|| \leq K_1K_2||A||$ where A is some product of length $k-1$ of matrices from Σ, K_1 is the maximal norm of a product of at most m matrices in $\tilde{\Sigma}$, and K_2 is the maximal norm of a product of at most 2 matrices in Σ. Using the last inequality, we arrive at

$$||\tilde{A}||^{1/(k-1)} \leq (K_1K_2)^{1/(k-1)}||A||^{1/(k-1)}.$$

The initial product $\tilde{A}$ is an arbitrary product of length $l = km + r$ and so by letting k tend to infinity and using the definition of the joint spectral radius we conclude $\rho(\tilde{\Sigma}) \leq \rho(\Sigma)^{1/m}$.

We have thus proved that $\rho(\tilde{\Sigma}) = \rho(\Sigma)^{1/m}$. It remains to prove the equivalence of the finiteness property. If Σ satisfies the finiteness property then

$\rho(\Sigma) = \rho(A_1 \dots A_T)^{1/t}$, then $\rho(\tilde{\Sigma}) = \rho(\Sigma)^{1/m} = \rho(\tilde{B}_1 \dots \tilde{B}_T)^{1/(Tm)}$ and so $\tilde{\Sigma}$ does too. In the opposite direction, if the finiteness property holds for $\tilde{\Sigma}$, then we must have $\rho(\tilde{\Sigma}) = \rho(\tilde{B}_1 \dots \tilde{B}_T)^{1/T}$, and then $\rho(\Sigma) = \rho(\tilde{\Sigma})^m = \rho(A_1 \dots A_T)^{1/T}$.

By combining the results obtained so far in this section, we can now state our main result. Before we do so, let us notice that the nonnegativeness of the matrices was not essential in the arguments given so far and this makes a separate statement possible for matrices with arbitrary rational entries.

Theorem 4. *The finiteness property holds for all sets of nonnegative rational matrices if and only if it holds for all pairs of binary matrices.*

The finiteness property holds for all sets of rational matrices if and only if it holds for all pairs of matrices with entries in $\{0, 1, -1\}$.

Proof. The proof for the nonnegative case is a direct consequence of Proposition 2, Theorem 2 and Theorem 3. For the case of arbitrary rational entries, the results and proofs of Proposition 2 and Theorem 3 may be applied as they are. For the construction in the proof of Theorem 2, one just has to weight any edge with -1 whenever it represents a negative entry, and the reasoning in the proof still holds.

Let us finally remark that for the purpose of reducing the finiteness property of rational matrices to pairs of binary matrices, we have proved here that for any set Σ of m matrices with integer (resp. nonnegative integer) entries, there is always a *pair* of matrices $\tilde{\Sigma}$ with entries in $\{0, 1, -1\}$ (resp. in $\{0, 1\}$) such that $\rho(\Sigma) = \rho(\tilde{\Sigma})^m$. Loosely speaking pairs of binary matrices have the same combinatorial complexity.

3 The finiteness property for pairs of 2 × 2 binary matrices.

In this section, we prove that the finiteness property holds for pairs of binary matrices of size 2×2. Even if this result may at first sight appear anecdoctic, it has some relevance since it has been shown in the previous section that pairs of binary matrices are not restrictive. Moreover, even for this 2×2 case, non-trivial behaviours occur. For instance, the set of matrices

$$\left\{ \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right\},$$

whose behaviour could at first sight seem very simple, happens to have a joint spectral radius equal to $((3 + \sqrt{13})/2)^{1/4}$, and this value is only reached by products of length at least four. Another interest of this section is to present techniques that may prove useful to establish the finiteness property for matrices of larger dimension.

There are 256 ordered pairs of binary matrices of dimension 2. Since we are only interested in unordered sets we can lower this number to $(2^4(2^4 - 1))/2 =$

120. We first present a series of simple properties that allow us to handle most of these cases and we then give a complete analysis of the few remaining cases. The proofs of these properties are given in the appendices. In the following, we note $A \leq B$ if the matrix $B - A$ has nonnegative entries.

Proposition 3. *For any set of matrices $\Sigma = \{A_0, A_1\} \subset \mathbb{R}^{2\times 2}$, we have*

- $\rho(\{A_0, A_1\}) = \rho(\{A_0^T, A_1^T\})$, *where A^T is the transpose of A,*
- $\rho(\{A_0, A_1\}) = \rho(\{SA_0S, SA_1S\})$, *where $S = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$.*

Moreover, in both cases the finiteness property holds for one set if and only if it holds for the other.

Proof. The spectral radius of a matrix is not changed by transposing or by applying similarity transformations. So for any product of matrices in Σ there is an obvious product of identical length of matrices in the other set that has the same spectral radius.

Proposition 4. *[20, Proposition 6.13] The finiteness property holds for sets of symmetric matrices.*

Proof. The matrix norm induced by the euclidean vector norm is given by the largest singular value of the matrix. For symmetric matrices the largest singular value is equal to the largest magnitude of the eigenvalues. Thus

$$\max\{||A|| : A \in \Sigma\} = \max\{\rho(A) : A \in \Sigma\}$$

and from (2) it follows that $\rho(\Sigma) = \max\{\rho(A) : A \in \Sigma\}$.

Proposition 5. *Let $\Sigma = \{A_0, A_1\} \in \mathbb{N}^{n\times n}$. The finiteness property holds in the following situations:*

1. $\rho(\Sigma) \leq 1$,
2. $A_0 \leq I$ *(or $A_1 \leq I$).*

Proof. (1) It is known that for sets of nonnegative integer matrices, if $\rho \leq 1$, then either $\rho = 0$ and the finiteness property holds, or $\rho = 1$, and there is a product of matrices in Σ with a diagonal entry equal to one [11]. Such a product $A \in \Sigma^t$ satisfies $\rho(\Sigma) = \rho(A)^{1/t} = 1$ and so the finiteness property holds when $\rho(\Sigma) \leq 1$.
(2) If $\rho(A_1) \leq 1$, then $\rho(A) \leq 1$ for all $A \in \Sigma^t$ and thus $\rho(\Sigma) \leq 1$ and the result follows from (1). If $\rho(A_1) > 1$ then $\rho(\Sigma) = \rho(A_1)$ and so the finiteness property holds.

Proposition 6. *Let $\Sigma = \{A_0, A_1\} \in \mathbb{N}^{n\times n}$. The finiteness property holds in the following situations:*

1. $A_0 \leq A_1$ *(or $A_1 \leq A_0$),*
2. $A_0A_1 \leq A_1^2$ *(or $A_1A_0 \leq A_1^2$),*

3. $A_0A_1 \leq A_1A_0$.

Proof. (1) Any product of length t is bounded by A_1^t. Hence the joint spectral radius of Σ is given by $\lim_{t\to\infty} ||A_1^t||^{1/t} = \rho(A_1)$.
(2) and (3). Let $A \in \Sigma^t$ be some product of length t. If $A_0A_1 \leq A_1^2$ or $A_0A_1 \leq A_1A_0$ we have $A \leq A_1^{t_1}A_0^{t_0}$ for some $t_0 + t_1 = t$. The joint spectral radius is thus given by

$$\begin{aligned}\rho = \lim_{t\to\infty} \max_{t_1+t_0=t} ||A_1^{t_1}A_0^{t_0}||^{1/t} &\leq \lim_{t\to\infty} \max_{t_1+t_0=t} ||A_1^{t_1}||^{1/t}||A_0^{t_0}||^{1/t} \\ &\leq \max\left(\rho(A_0), \rho(A_1)\right).\end{aligned}$$

Hence the joint spectral radius is given by $\max\left(\rho(A_0), \rho(A_1)\right)$.

In order to analyse all possible sets of matrices, we consider all possible pairs (n_0, n_1), where n_i is the number of nonzero entries in A_i. From Proposition 6, we can suppose $n_i = 1, 2,$ or 3 and without loss of generality we take $n_0 \leq n_1$.

- $n_0 = 1$:
  - If $n_1 = 1$ or $n_1 = 2$, the maximum row sum or the maximum column sum is equal to one for both matrices, and since these quantities are norms it follows from (2) that the joint spectral radius is less than one and from Proposition 5 that the finiteness property holds.
  - If $n_1 = 3$, it follows from Proposition 6 that the only interesting cases are:
$$\Sigma = \left\{ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \right\} \text{ and } \Sigma_0 = \left\{ \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \right\}.$$
In the first case the matrices are symmetric and so the finiteness property holds by Proposition 4. We keep Σ_0 for later.
- $n_0 = 2$:
  - $n_1 = 2$: The only interesting cases are:
$$\Sigma = \left\{ \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \right\} \text{ and } \Sigma_1 = \left\{ \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \right\}.$$
Indeed in all the other cases either the maximum row sum or the maximum column sum is equal to one and the finiteness property follows from Proposition 5. The joint spectral radius of the first set is equal to one. Indeed, it is known that for nonnegative integer matrices, if the joint spectral radius is larger than one, then there must be a product of matrices with a diagonal entry larger than one [11]. This is impossible here, since as soon as a path leaves the first node, it cannot come back to it, and no path can leave the second node. By Proposition 5 the finiteness property holds for the first set. We keep Σ_1 for further analysis.
  - $n_1 = 3$: If the zero entry of A_1 is on the diagonal (say, the second diagonal entry), then, from Proposition 5 we only need to consider the following case:
$$\left\{ \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \right\}.$$

These matrices are such that $A_0A_1 \leq A_1^2$ and so the finiteness property follows from Proposition 6.
If the zero entry of A_1 is not a diagonal entry, we have to consider the following cases:

$$\Sigma_2 = \left\{ \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \right\} \text{ and } \Sigma_3 = \left\{ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \right\}.$$

We will handle Σ_2 and Σ_3 later.

– $n_0, n_1 = 3$: It has already been noticed by several authors (see, e.g., [20, Proposition 5.17]) that

$$\rho\left(\left\{ \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \right\}\right) = \rho\left(\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \right)^{1/2} = \sqrt{\frac{1+\sqrt{5}}{2}}.$$

After excluding the case of symmetric matrices and using the symmetry argument of Proposition 3, the only remaining case is:

$$\left\{ \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \right\},$$

but again these matrices are such that $A_0A_1 \leq A_1^2$ and so the finiteness property follows from Proposition 6.

We now analyse the cases that we have identified above. For Σ_0, notice that $A_0^2 \leq A_0A_1$. Therefore, any product of length t is dominated by a product of the form $A_1^{t_1}A_0A_1^{t_2}A_0 \dots A_1^{t_l}$ for some $t_1, t_l \geq 0$ and $t_i \geq 1$ $(i = 2, \dots, l-1)$. The norm of such a product is equal to $(t_1+1)(t_l+1)t_2 \cdots t_{l-1}$. The maximal rate of growth of this norm with the product length is given by $\sqrt[5]{4}$ and so the joint spectral radius is equal to $\sqrt[5]{4} = \rho(A_1^4A_0)^{1/5}$. The maximal rate of growth is obtained for $t_i = 4$.

For Σ_1, simply notice that $\max_{A\in\Sigma^2} \rho(A) = \max_{A\in\Sigma^2} ||A||_\infty = 2$, where $|| \cdot ||_\infty$ denotes the maximum row sum norm. Hence by (2) we have $\rho(\Sigma) = \rho(A_0A_1)^{1/2} = \sqrt{2}$.

Consider now Σ_2. These matrices are such that $A_0^2 \leq A_0A_1$ and so any product of length t is dominated by a product of the form $A_1^{t_1}A_0A_1^{t_2}A_0 \dots A_1^{t_l}$ for some $t_1, t_l \geq 0$ and $t_i \geq 1$ $(i = 2, \dots, l-1)$. We have

$$A_1^{t_1}A_0 \dots A_1^{t_l}A_0 = \begin{pmatrix} (t_1+1)\dots(t_l+1) & 0 \\ (t_2+1)\dots(t_l+1) & 0 \end{pmatrix}$$

and the maximum rate of growth of the norm of such a product is equal to $\sqrt{2}$. This rate is obtained for $t_i = 3$ and $\rho = \rho(A_1^3A_0)^{1/4} = \sqrt{2}$.

The last case, Σ_3, is more complex and we give an independent proof for it.

Proposition 7. *The finiteness property holds for the set*

$$\left\{ \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right\}.$$

Proof. Because $A_0^2 = I$ we can assume the existence of a sequence of maximal-normed products Π_i of length L_i, of the shape $B_{t_1} \ldots B_{t_l}$ with $B_{t_i} = A_1^{t_i} A_0$, $\sum t_k + l = L_i$, and $\lim ||\Pi_i||^{1/L_i} = \rho(\Sigma)$. We show that actually any maximal-normed product only has factors B_3, except a bounded number of factors that are equal to B_1, B_2, or B_4 and so the finiteness property holds.
Let us analyse one of these products Π. We suppose without loss of generality that Π begins with a factor B_3. First, it does not contain any factor $B_t : t > 4$ because for such t, $B_{t-3}B_2 \geq B_t$ and we can replace these factors without changing the length.
Now, our product Π has less than 8 factors B_4, because replacing the first seven factors B_4 with B_3, and the eighth one with $(B_3)^3$ we get a bigger-normed product of the same length (this is because $B_3 \geq (3/4)B_4$, and $(B_3)^3 \geq (33/4)B_4$). We remove these (at most) seven factors B_4 and by doing this, we just divide the norm by at most a constant K_0.
We now construct a bigger-normed product Π' by replacing the left hand sides of the following inequalities by the respective right hand sides, which are products of the same length:

$$\begin{aligned} B_i B_1 B_1 B_j &\leq B_i B_3 B_j \\ B_2 B_1 B_2 &\leq B_3 B_3 \\ B_3 B_1 B_2 &\leq B_2 B_2 B_2 \\ B_2 B_1 B_3 &\leq B_2 B_2 B_2. \end{aligned}$$

If the factor $B_3 B_1 B_3$ appears eight times, we replace it seven times with $B_2^3 \geq (4/5) B_3 B_1 B_3$ and the last time with $B_2^3 B_3^2$ which is greater than $7B_2^3$. By repeating this we get a new product $\Pi'' \geq 7(4/5)^8 \Pi'(1/K_0) > \Pi'(1/K_0)$ that has a bounded number of factors B_1. We remove these factors from the product and by doing this we only divide by at most a constant K_1.
If there are more than four factors B_2 in the product, we replace the first three ones with B_3, and remove the fourth one. It appears that for any $X \in \{B_2, B_3\}$, $B_3^2 X > 1.35 B_3 B_2 X$, and on the other hand, $B_3^2 X \geq B_3^2 B_2 X \frac{1}{2{,}4349}$. Then each time we replace four factors B_2 we get a new product: $\Pi''' \geq \frac{1.35^3}{2.4348} \Pi''(1/K_1) > \Pi''(1/K_1)$. Finally we can remove the (at most) three last factors B_2 and by doing this, we only divide the product by at most a constant K_2. By doing these operations to every Π_i, we get a sequence of products Π_i''', of length at most L_i. Now, introducing $K = K_0 K_1 K_2$, we compute

$$\rho \geq \lim ||\Pi_i'''||^{1/(L_i)} \geq \lim ||(1/K)\Pi_i||^{1/(L_i)} = \rho.$$

Hence $\rho = \lim ||(A_1^3 A_0)^t||^{1/(4t)} = \rho(A_1^3 A_0)^{1/4} = ((3+\sqrt{13})/2)^{1/4}$, and the finiteness property holds.

4 Conclusion

This paper provides a contribution to the analysis of the finiteness property for matrices that have rational entries. We have shown that the finiteness property holds for all sets of matrices with nonnegative rational entries if and only if it holds for pairs of matrices with binary entries. For pairs of binary matrices of dimension 2×2 we have shown that the property holds true and we conjecture that it holds for binary matrices of arbitrary dimension. A natural way to prove the conjecture for pairs of binary matrices would be to use induction on the size of the matrices, but this does not seem to be easy. If the conjecture is true, it follows that the stability question for matrices with nonnegative rational entries is algorithmically decidable. If the conjecture is false, then the results and techniques developed in this paper can possibly help for constructing a counterexample.

References

1. M. A. Berger and Y. Wang. Bounded semigroups of matrices. *Linear Algebra and its Applications*, 166:21–27, 1992.
2. V. D. Blondel and V. Canterini. Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computing Systems*, 36(3):231–245, 2003.
3. V. D. Blondel, R. M. Jungers, and V. Protasov. On the complexity of computing the capacity of codes that avoid forbidden difference patterns. *IEEE Transactions on Information Theory*, 52(11):5122–5127, 2006.
4. V. D. Blondel, J. Theys, and A. Vladimirov. An elementary counterexample to the finiteness conjecture. *SIAM Journal on Matrix Analysis*, 24(4):963–970, 2003.
5. V. D. Blondel and J. N. Tsitsiklis. The lyapunov exponent and joint spectral radius of pairs of matrices are hard - when not impossible - to compute and to approximate. *Mathematics of Control, Signals, and Systems*, 10:31–40, 1997. Correction in 10, 381.
6. V. D. Blondel and J. N. Tsitsiklis. The boundedness of all products of a pair of matrices is undecidable. *Systems and Control Letters*, 41(2):135–140, 2000.
7. T. Bousch and J. Mairesse. Asymptotic height optimization for topical ifs, tetris heaps, and the finiteness conjecture. *Journal of the Mathematical American Society*, 15(1):77–111, 2002.
8. V. Crespi, G. V. Cybenko, and G. Jiang. The Theory of Trackability with Applications to Sensor Networks. Technical Report TR2005-555, Dartmouth College, Computer Science, Hanover, NH, August 2005.
9. L. Gurvits. Stability of discrete linear inclusions. *Linear Algebra and its Applications*, 231:47–85, 1995.
10. R. M. Jungers. NP-completeness of the computation of the capacity of constraints on codes. Master's thesis, Université Catholique de Louvain, 2005.
11. R. M. Jungers, V. Protasov, and V. D. Blondel. Efficient algorithms for deciding the type of growth of products of integer matrices, 2006. submitted to publication.
12. V. Kozyakin. A dynamical systems construction of a counterexample to the finiteness conjecture. *Proc. 44th IEEE Conference on Decision and Control and ECC 2005*, 2005.
13. J. C. Lagarias and Y. Wang. The finiteness conjecture for the generalized spectral radius of a set of matrices. *Linear Algebra and its Applications*, 214:17–42, 1995.
14. D. Liberzon, J. P. Hespanha, and A. S. Morse. Stability of switched systems: a lie-algebraic condition. *Systems and Control Letters*, 37(3):117–122, 1999.
15. B. E. Moision, A. Orlitsky, and P. H. Siegel. On codes with local joint constraints. Unpublished.

16. R. M. Jungers and V. D. Blondel. On the Finiteness Property for Rational Matrices. Submitted to publication.
17. B. E. Moision, A. Orlitsky, and P. H. Siegel. Bounds on the rate of codes which forbid specified difference sequences. In *Proc. 1999 IEEE Global Telecommun. Conf. (GLOBECOM'99)*, 1999.
18. B. E. Moision, A. Orlitsky, and P. H. Siegel. On codes that avoid specified differences. *IEEE Transactions on Information Theory*, 47:433–442, 2001.
19. V. Y. Protasov. On the asymptotics of the partition function. *Sb. Math.*, 191(3-4):381–414, 2000.
20. J. Theys. *Joint Spectral Radius : theory and approximations.* PhD thesis, Université Catholique de Louvain, 2005.

Reachability Problems in Low-Dimensional Iterative Maps

Oleksiy Kurganskyy[1], Igor Potapov[2], and Fernando Sancho Caparrini[3]

[1] Institute of Applied Mathematics and Mechanics, Ukrainian National Academy of Sciences, 74 R. Luxemburg St, Donetsk, Ukraine, kurgansk@gmx.de
[2] Department of Computer Science, University of Liverpool, Chadwick Building, Peach St, Liverpool L69 7ZF, U.K., igor@csc.liv.ac.uk
[3] Department of Computer Science and Artificial Intelligence, University of Seville, Spain, fsancho@us.es

Abstract. In this paper we analyse the dynamics of one-dimensional piecewise maps (PAMs). We show that one-dimensional PAMs are equivalent to pseudo-billiard or so called "strange billiard" systems. We also show that the more general class of rational functions leads to undecidability of reachability problem for one-dimensional piecewise maps with a finite number of intervals.

Keywords: *Theory of computing, iterative piecewise maps, pseudo-billiard systems, universality*

1 Introduction

In the present work we investigate a class of hybrid systems defined by one-dimensional piecewise maps. We are mainly interested in a class of one-dimensional piecewise-affine maps (PAMS) [2]. The analysis of piecewise-affine maps is one of the simplest model that generate complex behaviour, see [2–4, 6, 7]. It is known that the reachability problem is undecidable for the two-dimensional case and it is open for dimension one [1, 2].

It was recently shown that PAM is equivalent to hierarchical piecewise constant derivatives system (HPCD)[2]. In this paper we show that PAM is also equivalent to planar pseudo-billiard systems (PBSs) or so called "strange billiards" model that is a well known object in bifurcation and chaos theory [10, 11]. In contrast to HPCD which is a hybrid automaton where each state is defined by planar piecewise constant derivatives system (PCD), the model of PBS can also be seen as two dimensional linear hybrid automaton but with only one state.

Although the reachability for PAMs is known to be open we think that the shown equivalence between PBSs and PAMs can be useful and the results from chaos theory about "strange billiards" [10, 5, 11] could help understand the complexity in one-dimensional piecewise-affine maps.

In the second part of this paper we are exploring the complexity of more general class of rational maps that includes affine maps. It was shown in [8] that piecewise iterative maps defined by a very restricted basis of elementary functions:

$$\{x^2, x^3, \sqrt[2]{x}, \sqrt[3]{x}, x \pm 1, 10 \cdot x\}$$

can simulate a Minsky machine even in dimension one. Comparing to [8], we found a new way how to create a copy of information for a temporal use in dimension one. We show that it is possible to avoid square root and cube root functions using only rational functions. However in the current construction two of the finite number of intervals we define are infinite. As a main result in this part we show how to simulate (in direct way) a Minsky machine in one-dimensional piecewise rational maps (PRM) of degree 2. From it follows that the reachability problem for PRM is undecidable.

It would be interesting to investigate a natural class of one-dimensional piecewise linear rational maps that is in between affine and rational maps. The main motivation for this class of systems is based on the fact that the reachability in one-dimensional piecewise linear rational maps can be seen as parameterized reachability in two dimensional linear[1] maps. Another interesting question is nondeterministic maps where transformations can be applied in any order. In this case reachability problems for nondeterministic linear rational maps corresponds to parametrized membership in 2×2 matrix semigroups. According to undecidability in PRM we think that it is also very likely to find undecidability of reachability problems in a nondeterministic version of one-dimensional rational maps.

2 Preliminaries

In what follows we use traditional denotations $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$ for the sets of naturals, rationals and reals respectively.

A function from a set A to a set B, we will denote by $f : A \to B$. If f is an injection such that $dom(f) = A$, then it will be denoted by $f : A \hookrightarrow B$. In some cases we put $x \mapsto y$ under the definition of a function f to express that $y = f(x)$, for example:

$$\begin{aligned} f : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto ax + b \end{aligned}$$

is a way to say that $f(x) = ax + b$.

If we have a set A in a topological space (usually we will consider $\mathbb{R}$ or $\mathbb{R}^2$ with the euclidian topology), we will denote by $int(A)$ (the interior of A) the greatest open subset of A ($int(A) = \cup\{G : \ G \text{ open and } G \subseteq A\}$).

[1] A two dimensional linear function f is a function of the following type $f(x, y) = ax + by$

2.1 Dynamical Systems

Definition 1. *A* dynamical transition system *is a triple $S = (X, T, \Sigma)$, where X is a set (the set of points of the system), $T : X \to X$ (the transition function that produces the evolution of the system), and Σ is a collection of subsets of X (this component is only considered in the case we are interested in the symbolic behavior of the system).*

Remark 1. Usually, we will require Σ to be a partition of X, or at least to be a collection of pairwise disjoint subsets of X (in the case we are interested in the dynamical behavior of some parts of X, using the rest as *auxiliary computation*). Also, we will see Σ as an alphabet, and we will study the language generated by the system on this alphabet.

Definition 2. *Let $S = (X, T, \Sigma)$ a dynamical system, and $x \in X$. The sequence $\{x_n\}_{n \geq 0}$, such that:*

- *$x_0 = x$,*
- *for every $n \geq 0$, $x_{n+1} = T(x_n)$.*

is called the orbit of x by the system S*, and it will be denoted as $O_S(x)$.*

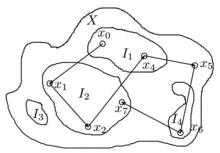


Fig. 1. Example of a dynamical system with an orbit in it

In Figure 1 we have a dynamical system $S = (X, T, \Sigma) = \{I_1, I_2, I_3, I_4\})$, where a partial orbit of a point, x_0, is shown.

Definition 3. *Let $S = (X, T, \Sigma)$ be a dynamical system, and $x \in X$. Let's associate the set $X \setminus \cup_{w \in \Sigma} w$ to the element ε (the empty word). The* symbolic dynamics of x in terms of Σ *is the set:*

$$\mathcal{S}_S(x) = \{w \in \Sigma^* : \ \forall n \geq 0 \ (O_S(x)_n \in w_n)\}$$

Where we use the notation $w = w_1 w_2 \ldots$.

In example above, $\mathcal{S}_S(x_0) = I_1 I_2 I_2 I_1 \varepsilon I_4 I_2 = I_1 I_2 I_2 I_1 I_4 I_2$. Note that point x_5 in the orbit has no representation in its symbolic dynamics.

Remark 2. If Σ is a collection of pairwise disjoint subsets of X, then for every point $x \in X$, $\mathcal{S}_S(x)$ has only one element.

Definition 4. *Let $S_1 = (X_1, T_1, \Sigma_1)$ and $S_2 = (X_2, T_2, \Sigma_2)$ two dynamical systems. We will say that S_2 simulates S_1 if there exists an injection $\varphi : X_1 \hookrightarrow X_2$, and an injection $\sigma : \Sigma_1 \hookrightarrow \Sigma_2$ such that for every $x \in X_1$, we have:*

$$\mathcal{S}_{S_2}(\varphi(x)) = \widehat{\sigma}(\mathcal{S}_{S_1}(x))$$

where $\widehat{\sigma} : \Sigma_1^* \hookrightarrow \Sigma_2^*$ is the morphism generated by σ.

2.2 Pseudo Billiard Systems

Let us introduce the pseudo billiard model that already appeared in a different context and became an abstract framework for some practical problems. In this system we consider a number of segments with vector fields assigned to them. The computation in this system can be described by the dynamics of the particle, which initially moves with the constant velocity (in a particular direction) inside a given region (not necessarily a polyhedron) and changes it instantaneously at the moment of a collision with the boundary to the velocity defined by a given vector field on the boundary.

We start with a more general definition for PBS's, where we have no constraints on distributing the segments around the space. In this case, a particle can touch the segments by both faces, and therefore it may cross them by the action of their projection vectors.

Definition 5. *A Pseudo Billiard System (PBS) is a pair $(\mathcal{A}, \mathcal{V})$, where $\mathcal{A}$ is a set of pairwise disjoint segments in $\mathbb{R}^2$ (closed, open or semi-open), and $\mathcal{V} = \{\boldsymbol{v}_A\}_{A \in \mathcal{A}}$ is a set of vectors in $\mathbb{R}^2 - \{(0,0)\}$ ($\boldsymbol{v}_A$ is called the projection vector of A).*

The dynamics of a particle in PBS can be defined as follows. Let a particle P that is represented by a vector x and is located on a segment $A \in \mathcal{A}$, i.e. $x \in A$. The transition function that moves P from x to a position x' can be defined as follows: $x' = x + \lambda \boldsymbol{v}_A$, where $\lambda = \min\{\delta > 0 \ : \ x + \delta \boldsymbol{v}_A \in \bigcup_{A' \in \mathcal{A}} A'\}$. We will suppose that for every $x \in A$ there exists such a λ (the particle is trapped inside the system).

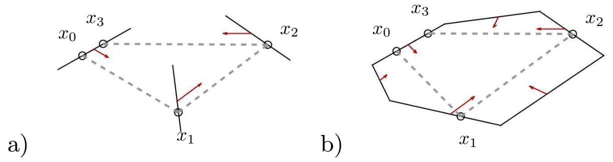


Fig. 2. An example of partial orbit: a) in a PBS, b) in a reflecting PBS

The PBS can be seen as a dynamical transition system $S = (X, T, \Sigma)$ where:

– $X = \bigcup_{A \in \mathcal{A}} A$,
– $T(x) = x + \lambda \boldsymbol{v}_A$, where $x \in A$ and $\lambda = \min\{\delta > 0 \ : \ x + \delta \boldsymbol{v}_A \in \bigcup_{A' \in \mathcal{A}} A'\}$
– Σ is any collection of subsets of X (usually it will be a subset of $\mathcal{A}$).

Definition 6. *A PBS is* reflecting, *if for every $A \in \mathcal{A}$, the set $T^{-1}(A)$ and $T(A)$ are in the same half-plane determined by A.*

2.3 Piecewise Affine Maps

Definition 7. *We say that $f : \mathbb{R} \to \mathbb{R}$ is a piecewise affine map (PAM) if there exists a partition of $dom(f)$ in a finite number of intervals of $\mathbb{R}$ (we allow the intervals to be closed, open or semi-open intervals), $\mathcal{I}$, and for every $I \in \mathcal{I}$, there exists $a_I, b_I \in \mathbb{R}$ such that: $\forall x \in I,\ f(x) = a_I x + b_I$.*

Remark 3. If we have $f(dom(f)) \subseteq dom(f)$, then we can consider a dynamical system associated to it, $S = (X, T, \Sigma)$ where:

– $X = dom(f)$,
– $T = f(x)$ and
– Σ is any collection of subsets of X (usually it will be a subset of $\mathcal{I}$).

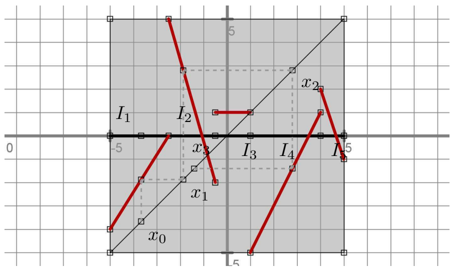


Fig. 3. An example of partial orbit in a PAM (represented on the diagonal)

Definition 8. *A rational function is a function defined as a ratio of polynomials. For a single variable x a typical rational function is therefore $f(x) = P(x)/Q(x)$, where P and Q are polynomials in x as indeterminate, and Q is not the zero polynomial.*

We also give the definition of a more general class of rational functions that we are going to study in the paper. We define it over $\mathbb{Q}$ to show that even in this case the predictability of its behaviour is an undecidable problem.

Definition 9. *A Piecewise (one-dimensional) rational map (PRM) is a function that is defined on a finite sequence of disjoint intervals $I_- = (-\infty, r_-]$, $I_+ = [l_+, +\infty)$, $I_i = [l_i, r_i]$ with $r_-, l_+, l_i, r_i \in \mathbb{Q}$, $i = 1..k$ and uses rational functions for different parts of its domain.*

The computation in the above system can be understood as a generation of sequence of points. One of the obvious problems that arises in such systems is a point-to-point reachability problem that can be formulated as follows:

Problem 1. Given two points $x, y \in \mathbb{Q}$ and a one-dimensional piecewise map P. Decide whether y is reachable from x in P.

3 Equivalence between Dynamical Systems

In this section we will study the equivalence between the models introduces above. We will say that two models are equivalent if for every system of one type there exists a system of another type that simulates it and vice versa. In particular we are giving geometrical constructions to show the equivalence of one-dimensional PAM, planar PBS and planar reflective PBS. Moreover using the result that model of hierarchical piecewise constant derivative systems (HPCDs) is equivalent to one-dimensional PAMs we can state that planar PBS is equivalent to two-dimensional HPCDs (see [2]). Hence the complexity that can be obtained with any of them is the same.

3.1 PAM simulates PBS

The first step through the equivalence will be devoted to prove that any PBS system (reflecting or not) can be simulated by a PAM system.

Theorem 1. *For every Pseudo Billiard System, $\{\mathcal{A}, \mathcal{V}\}$, there exists a Piecewise Affine Map that simulates it.*

Proof. Let us consider a PBS given by a set of segments $\mathcal{A} = \{A_i\}$ and a set of associated projection vectors $\{\boldsymbol{v}_i\}$. You can see an example on Figure 4 .a.

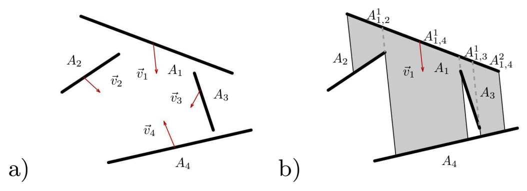


Fig. 4. a) Example of PBS to be simulated by a PAM; b) Projection of A_1 on other segments of the PBS, and the partition on A_1 that it generates

The dynamics of the PBS is defined by projecting every point of A_i on some other segment by using the projection vector $\boldsymbol{v}_i$. It is clear, from the definition of PBS, that we can make a partition of A_i in segments, $\{A^k_{i,j}\}$, in such a way that every point of $\{A^k_{i,j}\}$ is projected on a point of A_j (see Figure 4 .b). The next step is to associate for every segment of the system, A_i, an interval on the line, I_i, by using an affine bijection, $\mu_i : A_i \to I_i$. Also, we will require these intervals to be pairwise disjoint. For every i, j, k such that $A^k_{i,j} \neq \emptyset$, the projection $P^k_{i,j} : A^k_{i,j} \to A_j$ is an affine transformation. Also, all the functions μ_i are affine transformations.

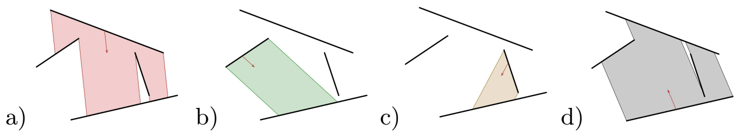


Fig. 5. Projections: a) from A_1; b) from A_2; c) from A_3; d) from A_4

Hence, we can define an affine map

$$\begin{aligned} f_{i,j,k} : \mu_i(A^k_{i,j}) &\to \mu_j(A_j) \\ x \qquad &\mapsto \mu_j(P^k_{i,j}(\mu_i^{-1}(x))) \end{aligned}$$

Since $\{A^k_{i,j}\}_{i,j,k}$ is a partition of the points of the PBS, $\{I^k_{i,j}\}_{i,j,k}$ is a partition of the set of intervals considered, hence we obtain that the map

$$\begin{aligned} f : \bigcup_{i,j,k} I^k_{i,j} &\to \bigcup_{i,j,k} I^k_{i,j} \\ x \qquad &\mapsto f_{i,j,k}(x), \text{ if } x \in I^k_{i,j} \text{ is a piecewise affine map.} \end{aligned}$$

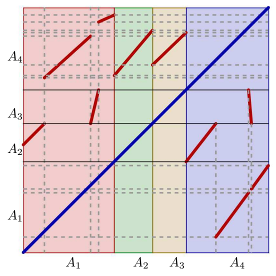


Fig. 6. PAM obtained after the process

In order to prove the dynamical simulation, let us consider the following injection between the set of points of the systems:

$$\begin{aligned} \varphi : \bigcup A_i &\hookrightarrow \bigcup I_i \\ x \quad &\mapsto \mu_i(x), \text{ if } x \in A_i. \end{aligned}$$

If Σ is the subset of A that produces the symbolic dynamics, then $\Sigma' = \{\varphi(I) : I \in \Sigma\}$ is the collection to be considered in the PAM, and $\sigma : \Sigma \to \Sigma'$

defined by $\sigma(I) = \varphi(I)$ is the injection for the dynamics. From the construction, it is obvious that f simulates the given PBS.

Lemma 1. *The number of affine functions we need in order to simulate a PBS is bounded*[2] *by* $|\mathcal{A}|(|\mathcal{A}| + 2)$.

Proof. Idea: for every segment of the PBS, the partition we need to make all possible projections on the other segments is bounded in size by $|\mathcal{A}| + 2$.

3.2 PBS simulates PAM

Next, we will prove that for any PAM we can build a PBS simulating its dynamical behavior. Indeed, we will see that they can be simulated using only reflecting PBS's, hence there is no difference (regarding the dynamical complexity of the system) in using a general PBS or restrict ourselves to reflecting PBS's.

Nevertheless, we will prove in a first step that, for every PAM we can get a general PBS (usually not reflecting) that simulates the given PAM. The proof is based on the graphical idea about how to compute the orbit of a point directly on the graph generated by the PAM (where we consider the dynamics on the diagonal rather than on the X axis) using the iterated projections between the affine map and the diagonal.

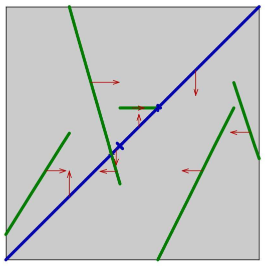


Fig. 7. PBS associated to a PAM

In this example we can see that the PBS obtained by the dynamics of the PAM over the diagonal is, in general, a non reflecting one, because depending its definition, we will need to cross some of the affine map graphs to reach the diagonal. In any case, it is not a problem, because if we must cross one map from another one, it is because both of them are in the same half-plane from the diagonal, and then their associated vectors are parallel and in the same direction.

It is easy to see that, if we restrict the dynamics on the set of segments over the diagonal, rather than on the X axis, the system we obtain is equivalent to the original one.

[2] In case of reflecting PBSs, the bound can be reduced to $|\mathcal{A}| + 2$

Theorem 2. *For every Piecewise Affine Map there exists a Pseudo Billiard System that simulates it.*

Proof. Let $f = \bigcup_{i=1}^n f_i$ a PAM where every f_i is an affine map over an interval I_i. We consider the following segments on $\mathbb{R}^2$:

- For every $I_i = [a_i, b_i]$, we consider its projection on the diagonal, $x = y$, that we note as A_i.
- For every I_i we consider the segment given by $(a_i, f_i(a_i)) - (b_i, f_i(b_i))$, that we note as $f(A_i)$.

We can consider that there is no intersection between the interior of segments of the PAM (otherwise if any A_i intersects with some $f(A_j)$, we consider the intersection point, and subdivide both segments, leaving this point in the diagonal segment (in Figure 7 we have split the second affine function in order to have segments with no intersection in their interiors). Of course, the obtained PAM is equivalent to the original one.

The vectors associated with every segment is given by the following rule: for every i

- if $int(f(A_i))$ is inside the half-plane $x < y$ (the upper half) then the vector associated to A_i is $(0, 1)$, and the vector associated to $f(A_i)$ is $(1, 0)$.
- if $int(f(A_i))$ is inside the half-plane $x > y$ (the lower half) then the vector associated to A_i is $(0, -1)$, and the vector associated to $f(A_i)$ is $(-1, 0)$.

In order to prove the dynamical simulation, let us consider the following injection between the set of points of the systems:

$$\begin{aligned} \varphi : \bigcup I_i &\hookrightarrow \bigcup A_i \\ x \quad &\mapsto (x, x) \end{aligned}$$

and, following the same procedure as in theorem 1, the same injection between the dynamics of the systems.

From the construction, it is obvious that the resulting PBS simulates the given PAM (note that we use only a part of the dynamics of the PBS, considering only the dynamics on the diagonal, and not the evolution of the points through the other segments, necessary for the correct computing of the evolution, but not for the dynamics itself).

3.3 Reflecting PBS simulates PAM

Theorem 3. *For every Piecewise Affine Map there exists a Reflecting Pseudo Billiard System that simulates it.*

Proof. Let $f : I \to I$ be a PAM expressed in such a way that $I = \bigcup_{i=1}^n I_i$ is union of pairwise disjoint intervals, and for every i, $f_{|I_i} = f_i$, where $f_i(x) = a_i x + b_i$ is an affine function.

The first step of the proof consists in assigning to every interval of the PAM a segment in $\mathbb{R}^2$ where we simulate the dynamics of the system. Since $f_i : I_i \to I$

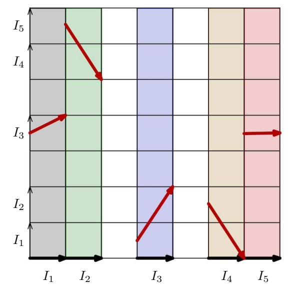


Fig. 8. PAM to be simulated

is affine, and I_i is an interval, $f_i(I_i)$ must be an interval too. Hence, the image of every interval of our partition must be inside an union of intervals of our partition that constitutes a larger interval. To make more direct the proof, we will maintain the continuity among intervals of f by considering for every interval, $I_i \subseteq \mathbb{R}$, of f, the segment $A_i = I_i \times \{0\} \subseteq \mathbb{R}^2$.

Now, we will simulate the dynamics of each affine map separately. Because the segments A_i are in the same line, we can't go directly from one to another by using projections, therefore we will make use of auxiliary reflection segments to produce the same result as f produces.

Depending on the coefficients of the affine map, there are three different cases:

1. **Case 1:** $a_i > 0$. In this case there is no flip from A_i to $f_i(A_i)$, so we will need only one reflecting auxiliary segment to simulate the application of f, B_i (see figure 9 .a).

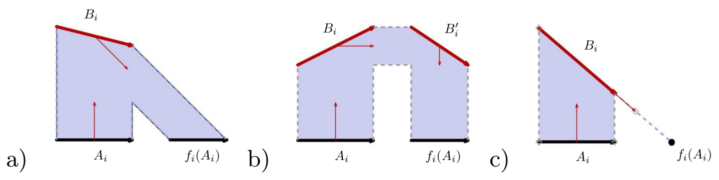


Fig. 9. a) Case 1: $a_i > 0$ b) Case 2: $a_i < 0$ c) Case 3: $a_i = 0$

2. **Case 2:** $a_i < 0$. In this case there is a flip from A_i to $f_i(A_i)$, so we will need two reflecting auxiliary segments, B_i and B_i', to simulate the application of f (see figure 9 .b).
3. **Case 3:** $a_i = 0$. In this case $f(A_i)$ is a point, and we will make use of only one reflecting auxiliary segment, B_i, to project to this point (see Figure 9 .c). Indeed, it can be seen as a extremal subcase of case 1.

We can construct simultaneously all these segments with projection vectors on $\mathbb{R}^2$ without disturbing one to each other, obtaining a reflecting PBS (see Figure 10 for a complete construction for PAM in Figure 8).

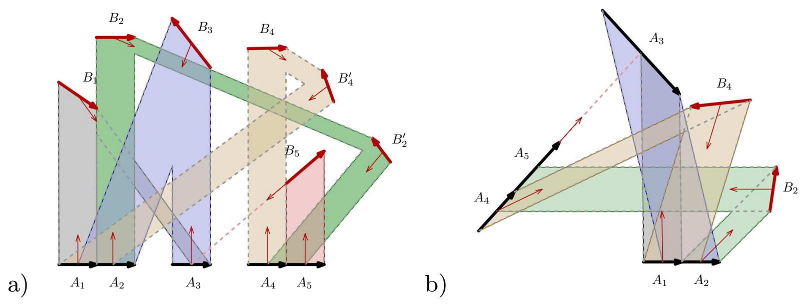


Fig. 10. a) Reflecting PBS simulating PAM b) PBS with reduced number of segments

In order to prove the dynamical simulation, let us consider the following injection between the set of points of the systems:

$$\begin{array}{rcl} \varphi : \bigcup I_i & \hookrightarrow & \bigcup A_i \\ x & \mapsto & (x, 0) \end{array}$$

and the same injection between the dynamics as in the previous theorems.

From the construction, it is obvious that the resulting PBS simulates the given PAM (note that, again, we use only a part of the dynamics of the PBS, considering only the dynamics on the segments A_i, and not the evolution of the points through the other segments, necessary for the correct computing of the evolution, but not for the dynamics itself).

From above construction, we obtain an upper bound to the number of segments we need in a reflecting PBS to simulate a PAM.

Corollary 1. *Let f be a PAM with N affine functions. Let R be the number of affine maps, f_i, with $a_i < 0$. Then, there is a reflecting PBS simulating f using, at most, $2N + R$ reflecting segments.*

Remark 4. The method of construction presented previously is not efficient in general, but it works for any possible PAM. In a number of PAM's, it is possible to reduce the number of elements of the PBS simulating the PAM. For example, in Figure 10 .a, we can identify segment A_3 with B_3 (of course, taking A_3 out of the X-axis), making unnecessary the use of B_1, B_3 and B_5. Also, in this example, if we change the orientation of A_4 and A_5 we can avoid the use of some auxiliary segments, B_2' and B_4'. We have reduced the construction from 12 segments to only 7 (it is easy to check that in this example we need, at least, 5 segments in order to simulate the dynamics), see Figure 10 .b.

Since $\mathbb{Q}$ is closed under linear rational transformations, if we restrict segments and vectors in $\mathbb{Q}^2$ for the PBS, and intervals and coefficients in $\mathbb{Q}$ for the PAMs, everything can be proved in the same way and the equivalence remains true.

4 Unpredictability in rational piecewise maps

In this section we show that the reachability problem in one dimensional rational piecewise maps is undecidable since for every Minsky machine [9] we can define a PRM that simulates its computation. Actually we need to show how the states, transition function and updates of integer counters can be simulated by a piecewise rational map P.

We found a new way how to create a copy of information for a temporal use in dimension one by means of rational functions. It allows us to simulate (in direct way) a Minsky machine in one-dimensional piecewise rational maps (PRM) of degree 2. Note that one-dimensional piecewise affine maps is a subclass of PRMs.

Let A be a 2-counter machine with a set of states $S = \{1, 2, \ldots, n\}$. The configuration of A is a triple $[k, l, s]$ where k and l are values of two counters and s is a current state of A. Let us define the mapping $\phi : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ that is an isomorphism between a configuration $[k, l, s]$ of A and a rational number $s + \frac{1}{2^{k+1}3^{l+1}}$ that is shifted to the interval (0,1)

$$\phi([k,l,s]) \to \frac{1}{10^H}(s + \frac{1}{2^{k+1}3^{l+1}}), H = \lceil \lg(|S|) \rceil$$

Instead of classical Minsky machine from now on we will consider a well-known equivalent model of two counter machine where one of the counters is used as a scratchpad. Another, counter holds an integer whose prime factorization is $2^c \cdot 3^d$. The exponents c, d can be thought of as two virtual counters that are being simulated. If the real counter is set to zero then incremented once, that is equivalent to setting all the virtual counters to zero. If the real counter is doubled, that is equivalent to incrementing c, and if it is halved, that is equivalent to decrementing c. By a similar procedure, it can be multiplied or divided by 3, which is equivalent to incrementing or decrementing d.

To check if a virtual counter such as c (d) is equal to zero, just divide the real counter by 2 (3), see what the remainder is, then multiply by 2 (3) and add back the remainder. That leaves the real counter unchanged. The remainder will have been nonzero if and only if c (d) was zero.

Let A be in configuration $[k, l, s]$ and it is represented by a number

$$x = \frac{1}{10^H}(s + \frac{1}{2^{k+1}3^{l+1}}).$$

Let us show that we can perform the operations of multiplication and division by 2 and 3 in a piecewise rational map P. To multiply/divide virtual counter by 2 or/and 3 we can use the following expression for x, where a,b are integers:

$$\frac{(10^H x - s)2^a 3^b + s}{10^H}$$

Now, we construct a system of intervals with rational functions, associated to them, that allows us to check divisibility of the value of the virtual counter by 2 and 3 or in other words to perform a zero testing on counters of original Minsky machine. For each state s of a counter machine we define the following intervals and functions:

Let us assume that the current configuration $[k, l, s]$ of a machine M is represented by a rational number x. If M is in a state s then x belongs to the interval $[\frac{s}{10^H}, \frac{s+1}{10^H}]$. Assuming that we know the current state we can add to x an integer $2^{k+1}3^{l+1}$ by expression $\frac{1}{(10^H x - s)} + x$. In fact for further simulation of checking the emptiness of the one Minsky machine counter we would need to add an integer $2^k 3^{l+1}$ using the expression $\frac{1}{2(10^H x - s)} + x$. Such operation gives us an extra information about the counter values in integer part of the number. It is important that we can use it now for some temporal computations and keep another copy of the current state and counter values in the decimal part of the number.

Now we can easily check whether a virtual counter is divisible by 2 iteratively applying $x - 2$ while the point x is in the interval $[3, +\infty)$. Finally a point x should reach either the interval $[2, 3]$, which corresponds to $k \neq 0$, or the interval $[1, 2]$, which corresponds to $k = 0$.

In a similar way we can check divisibility by 3 from a state s using negative numbers. If $x \in [\frac{s}{10^H}, \frac{s+1}{10^H}]$ we apply $-(\frac{1}{3(10^H x - s)} + x)$ and then $x + 3$ for any point in the interval $(-\infty, -4]$. Next the number x should appear in the interval $[-4, -3]$, which corresponds to $l \neq 0$ or in the interval $[-3, -1]$, which corresponds to $l = 0$.

Now we define a piecewise rational map to simulate all operations of Minsky machine such as state transitions, update of counters and testing them for zero. Initially let us define two intervals for intermediate computations related to the zero testing in counters:

If $x \in [3, +\infty)$ then apply $x - 2$, If $x \in (-\infty, -4]$ then apply $x + 3$

Next for every command of the Minsky machine

State s: IF $k \neq 0$ THEN k=k+a, l=l+b GOTO State t ELSE GOTO State p

we define a set of intervals with assigned rational functions:

If $x \in [\frac{s}{10^H}, \frac{s+1}{10^H}]$ then apply $\frac{1}{2(10^H x - s)} + x$

If $x \in [2 + \frac{s}{10^H}, 2 + \frac{s+1}{10^H}]$ then apply $\frac{(10^H(x-2)-s)\cdot 2^a 3^b + t}{10^H}$

If $x \in [1 + \frac{s}{10^H}, 1 + \frac{s+1}{10^H}]$ then apply $\frac{(10^H(x-1)-s)+p}{10^H}$

where $a \in \mathbb{Z}$ stands for increasing (decreasing) of the first counter by an integer a, and $b \in \mathbb{Z}$ stands for increasing (decreasing) of the second counter by an integer b.

Next for every command of the Minsky machine with testing of the second counter for zero

State s: IF $l \neq 0$ THEN k=k+a, l=l+b GOTO State t ELSE GOTO State p

We define a set of intervals in a similar way:

If $x \in [\frac{s}{10^H}, \frac{s+1}{10^H}]$ then apply $-(\frac{1}{3(10^H x-s)} + x)$

If $x \in [-(3 + \frac{s}{10^H}), -(3 + \frac{s+1}{10^H})]$ then apply $\frac{(10^H(x+4)-s)\cdot 2^a 3^b + t}{10^H}$

If $x \in [-(2 + \frac{s}{10^H}), -(2 + \frac{s+1}{10^H})]$ then apply $\frac{(10^H(x+3)-s)+p}{10^H}$

If $x \in [-(1 + \frac{s}{10^H}), -(1 + \frac{s+1}{10^H})]$ then apply $\frac{(10^H(x+2)-s)+p}{10^H}$

Since the computation of a Minsky machine can be simulated by a specially designed PRM the following theorem holds:

Theorem 4. *One-dimensional piecewise rational map with a finite number of intervals is the universal model of computations.*

Corollary 2. *The reachability problem (Problem 1) for one-dimensional PRM is undecidable.*

Corollary 3. *There exists a particular one-dimensional PRM , that corresponds to the universal Minsky machine, for which the point-to-point reachability problem is undecidable.*

5 Conclusion

In this paper we show that the model of one-dimensional PAMs is equivalent to a known model of strange billiards from bifurcation and chaos theory. On the other hand we show that predictability in more general one-dimensional class of functions (that includes one-dimensional PAMs) is not possible since we can encode a universal model of computation such as Minsky Machine.

It would be interesting to investigate a natural class of one-dimensional linear rational maps that is in between affine and rational maps. As far as we know the reachability problem for piecewise or nondeterministic maps is open in both cases. The reachability in piecewise linear rational maps related to parameterized reachability in two-dimensional linear maps and the reachability in nondeterministic linear rational maps can be interpreted as parameterized vector reachability problem in 2×2-matrix semigroups.

References

1. Asarin, E.; Maler, O.; Pnueli, A. Reachability analysis of dynamical systems having piecewise-constant derivatives, Theoretical Computer Science 138, (1995) 35-65
2. Asarin, E.; Schneider, G. Widening the boundary between decidable and undecidable hybrid systems. CONCUR'2002,LNCS 2421 (2002) 193-208
3. Blondel, V.; Bournez, O.; Koiran, P.; Papadimitriou, C.; Tsitsiklis, J.: Deciding stability and mortality of piecewise affine dynamical systems. Theor. Comput. Sci. 255(1-2) (2001) 687-696
4. Blondel, V.; Tsitsiklis, J. A survey of computational complexity results in systems and control, Automatica 36 (2000) pp. 1249-1274.
5. Chase, C.; Serrano, J.; Ramadge, P.J. Periodicity and chaos from switched flow systems:contrasting examples of discretely controlled continuous systems. IEEE Trans. Automatic Control, 38 (1993) 70-83
6. Koiran, P.; Cosnard, M.; Garzon, M. Computability with Low-Dimensional Dynamical Systems. Theor. Comput. Sci. 132(2) (1994) 113-128.
7. Koiran, P. The topological entropy of iterated piecewise affine maps is uncomputable. DMTCS 4(2), (2001) 351-356.
8. Kurganskyy, O.; Potapov, I.. Computation in One-Dimensional Piecewise Maps and Planar Pseudo-Billiard Systems. Unconventional Computation, LNCS 3699, 169-175, 2005.
9. Minsky, M. Computation: Finite and Infinite Machines, Prentice-Hall, Inc., N.J., 1967.
10. Peters, K.; Parlitz, U. Hybrid systems forming strange billiards. Int. J. of Bifurcations and Chaos, 19, (2003) 2575-2588.
11. Schurmann, T.; Hoffman, I. The entropy of strange billiards inside n-simplexes. J. Phys. A 28, (1995) 5033-5039.

Reachability Analysis in Verification via Supercompilation

Alexei Lisitsa[1] and Andrei P. Nemytykh[2]*

[1] Department of Computer Science, The University of Liverpool
alexei@csc.liv.ac.uk

[2] Program Systems Institute of Russian Academy of Sciences
nemytykh@math.botik.ru

Abstract. We present an approach to verification of parameterized systems, which is based on program transformation technique known as supercompilation. In this approach the statements about safety properties of a system to be verified are translated into the statements about properties of the program that simulates and tests the system. The supercompilation is used then to establish the required properties of the program. In this paper we show that reachability analysis performed by supercompilation can be seen as the proof of a correctness condition by induction. We formulate suitable induction principles and proof strategies and illustrate their use by examples of verification of parameterized protocols.

Keywords: Program verification, cache coherence protocols, program specialization, supercompilation.

1 Introduction

The verification of infinite-state or parameterized problems is, in general, an undecidable problem. The research in this area is focused on finding restricted classes of problems, for which verification is decidable and the development of efficient verification procedures for practical applications. The research is active and taking different routes [10, 8, 3, 2, 1]. But still many practically interesting verification problems lie outside the scope of existing automated verification methods and further development of these methods is required.

One of the recent interesting and promising directions for tackling infinite-state, or parameterized, verification is to apply the methods developed in the area of *program transformation* and *metaprogramming*, and in particular, *program specialization* [11, 19, 18].

In this paper we are interested in one particular approach in program transformation and specialization, known as supercompilation[1]. *Supercompilation* [36]

* The second author is supported by the Program for Basic Research of the Presidium of Russian Academy of Sciences (as a part of "Development of the basis of scientific distributed informational-computing environment on the base of GRID technologies"), and the Russian Ministry of Sciences and Education (grant 2007-4-1.4-18-02-064).

[1] from *super*vised *compilation*

has not drawn much attention yet in the context of verification, although it has been mentioned in [19, 18] as potentially applicable here. The supercompilation is a powerful semantic based program transformation technique [36, 39, 34] having a long history well back to the 1960-70s, when it was proposed by V. Turchin. The main idea behind a supercompiler is to observe the behavior of a functional program P running on *partially* defined input with the aim to define a program, which would be equivalent to the original one (on the domain of latter), but having improved properties. The supercompiler unfolds a potentially infinite tree of all possible computations of a parameterized program. In the process, it reduces the redundancy that could be present in the original program. It folds the tree into a finite graph of states and transitions between possible (paramemetrized) configurations of the computing system. And, finally, it analyses global properties of the graph and specializes this graph with respect to these properties (without an additional unfolding). The resulting definition is constructed solely based on the meta-interpretation of the source program rather than by a (step-by-step) transformation of the program.

The result of supercompilation may be a specialized version of the original program, taking into account the properties of partially known arguments, or just a re-formulated, and sometimes more efficient, equivalent program (on the domain of the original) [14].

Turchin's ideas have been studied by a number of authors for a long time and have, to some extent, been brought to the algorithmic and implementation stage [29]. From the very beginning the development of supercompilation has been conducted mainly in the context of the Refal programming language [38, 28], another creation of V.Turchin. A number of the simplest model supercompilers for subsets of LISP-like languages were implemented as well with an aim to formalize some aspects of the supercompilation algorithms [33, 35, 34]. The most advanced supercompiler for Refal is SCP4 [29, 27, 26, 28].

In [21–24] we proposed to use supercompilation for verification of parameterized systems using a particular scheme of *parameterized testing*. Using this scheme we translate the statements about safety properties of a system to be verified into the statements about properties of the program that *simulates and tests* the system. The supercompilation is used then to establish the required properties of the program. We have conducted series of experiments on verification of parameterized *cache coherence protocols* and successfully verified [22] all cache coherence protocols presented in [5] and [7]. We have also verified in this way parameterized Java MetaLock algorithm and series of Petri Nets models. This work started mainly as an experiment driven one and the approach proved to be empirically successful. This left however the questions on its correctness and completeness for classes of verification problems open. In this paper we address the issue of correctness of proposed method. We develop a formal model, which renders supercompilation process in the particular context of parameterized testing as a reachability analysis for term-rewriting systems by means of inductive proofs of safety properties. This establises the correctness of the method.

Further we illustrate our method by verification of parameterized MOESI protocol [5]. Interestingly enough, using supercompilation to perform parameterized testing allows not only to verify the protocol but also to discover new facts about the protocol. The facts are formulated by automatic generalization of configurations - one of the tools of supercompilation. In particular, an analysis of the supercompilation trace shows that the protocol is correct with more general assumptions on the initial state than reported in [5].

The paper is organized as follows. In the next section we give general description of our *verification via parameterized testing* approach in language-independent terms. Section 3 presents the formal model and correctness result.

Then in section 4 we introduce some of the strategies leading to successful verification of a class of parameterized cache coherence protocols. In section 5 we present a free monoid of terms and specify the strategies of the supercompiler SCP4 in its terms. The size limit of the paper does not allow us to present a detailed verification of the MOESI protocol using these strategies. We refer the reader to the Appendix [25], which we put on an Internet page as a separated document.

2 Parameterized Testing

In this section we describe our general technique for the verification of parameterized systems. The technique is based on the translation of the statements about *safety* properties of a system to be verified into the statements about properties of the program that *simulates and tests* the system.

The scheme works as follows. Let S be a parameterized system (a protocol) and we would like to establish some safety property P of S. We write a program φ_S simulating execution of S for n steps, where n is an input parameter. Let the n be given in the unary system, as a string of characters. If the system is non-deterministic, we label each step with an action, whose value is assumed to be chosen at the branching point of execution, e.g. it may be a character labelling the choice. Thus, we assume that given the values of input parameter n, the program φ_S returns the state of the system S after the execution of n steps of the system, following the choices provided by the labels of the steps. Let $T_P(\_)$ be a testing program, which given a state s of S returns the result of testing the property P on s (*True* or *False*). Consider a composition $T_P \circ \varphi_S$. This program first simulates the execution of the system and then tests the property required. Let the both programs terminate. Now the statement "the safety property P holds in any possible state reachable by the execution of the system S from an initial state" is equivalent to the statement "the program $T_P(\varphi(n))$ never returns the value *False*, no matter what values are given to the input parameter".

In practical implementation of the scheme we use functional programming language Refal to implement a program $T_P(\varphi_S(n))$ and optimizer SCP4 (a supercompiler) to transform the program to a form, from which one can easily establish the required property.

The idea of using testing and supercompilation for the verification purposes is not new. In the classical paper [36] V.Turchin writes: *Proving the correctness of a program is theorem proving, so a supercompiler can be relevant. For example, if we want to check that the output of a function $F(x)$ always has the property $P(x)$, we can try to transform the function $P(F(x))$ into an identical T.* The idea has not been tried until recently for the problems interesting for verification community. Our experiments have shown that indeed, the idea is viable and can be adopted for non-trivial verifications problems for parameterized distributed systems.

3 Correctness Issue and Formal Model

One of the immediate questions posed by almost everyone seeing the approach in work for the first time is "Is this correct at all? Why should I believe your claims about verification?"

Firstly, one can argue as follows. It has been shown, in particular in [33, 35, 32] that (variants of) supercompilation is a correct transformation, in a sense it always returns (if any) the program equivalent to the input program (on the domain of latter). Then we repeat the argument from Section 2. We should note, in this respect, that SCP4 [27, 26, 29] is a large program dealing with the concrete functional language Refal, which has specific semantic assumptions, like built-in associativity of concatenation as a term forming construct. Furthermore, SCP4 supercompiler, is a result for the more than two decades development and it is highly optimized program, implementing different strategies which can be tailored by the user to the particular cases. Proving the correctness of the whole SCP4 is far from being trivial and is, actually, irrelevant to our experiments. Even if we accept the correctness, it does not explain *why* supercompilation works for establishing correctness properties. We address these issues in the present paper by developing a formal model, which renders supercompilation process (in the case of verification tasks) as an inductive proof of safety properties. That establishes the correctness of the method. The formal model is a very simplified and refined theoretical version of SCP4, which, nevertheless, is sufficient for verification of a class of (parameterized) cache coherence protocols.

In this paper we confine ourselves by the claim that supercompiler SCP4 indeed implements the formal model we present. We provide some relevant comments but detailed discussion of the claim lies outside the scope of this paper. The model formulated in terms of term rewriting systems.

3.1 Term Rewriting Systems and Safety Properties

Let $\mathcal{V}$ be a denumerable set of symbols for variables and $\mathcal{F} = \cup_i \mathcal{F}_i$ be a finite set of functional symbols, here $\mathcal{F}_i$ is a set of functional symbols of arity i. Let $\mathcal{T}(\mathcal{V}, \mathcal{F})$ be a free algebra of all terms build with variables from $\mathcal{V}$ and functional symbols from $\mathcal{F}$ in a usual way. Let every $\mathcal{F}_i$ be divided into disjoint sets $\mathcal{F}_i = \mathcal{F}n_i \cup \mathcal{C}_i$. We refer to $\mathcal{F}n_i$ as function names and to $\mathcal{C}_i$ as constructor names. Let $\mathcal{C} = \cup_i \mathcal{C}_i$.

A term without function names is passive. Let $\mathcal{G}(\mathcal{T}) \subset \mathcal{T}(\mathcal{V}, \mathcal{F})$ be the set of ground terms, i.e. terms without variables. Let $\mathcal{O}(\mathcal{T}) \subset \mathcal{G}(\mathcal{T})$ be the set of object terms, i.e. ground passive terms. For a term t we denote the set of all variables in t by $V(t)$.

A substitution is a mapping $\theta : \mathcal{V} \to \mathcal{T}(\mathcal{V}, \mathcal{F})$. A substitution can be extended to act on all terms homomorphically. A substitution is called *ground*, *object*, or *strict* iff its range is a subset of $\mathcal{G}(\mathcal{T})$, $\mathcal{O}(\mathcal{T})$ or $\mathcal{T}(\mathcal{V}, \mathcal{C})$ (i.e. passive terms), respectively. We use notation $s = t\theta$ for $s = \theta(t)$, call s be an *instance* of t and denote this fact by $s \ll t$.

A term-rewriting system is a pair $P = \langle t, R \rangle$, where t is a term called *initial* and R is a finite set of rules of the form $f(p_1, \ldots, p_k) \to r$, where $f \in \mathcal{F}n_k$, $\forall i \; (p_i \in \mathcal{T}(\mathcal{V}, \mathcal{C}))^2$, $r \in \mathcal{T}(\mathcal{V}, \mathcal{F})$, $V(r) \subseteq V(f(p_1, \ldots, p_k))$.

Given a set of rules R define one-step transition relation $\Rightarrow_R \subseteq \mathcal{T}(\mathcal{V}, \mathcal{F}) \times \mathcal{T}(\mathcal{V}, \mathcal{F})$ as follows: $t_1 \Rightarrow_R t_2$ holds iff there exist a strict substitution θ and a rule $(l \to r) \in R$ such that $t_1 = l\theta$ and $t_2 = r\theta$. *Reachability* relation $\Rightarrow_R^*$ is a *transitive* and *reflexive* closure of $\Rightarrow_R$. Notice, that any term reachable for a ground term is also ground.

Definition 1. *A binary relation $\Rightarrow$ on a set $\mathcal{T}$ is terminating (or well-founded) if there exists no infinite chain $t_1 \Rightarrow t_2 \Rightarrow t_3 \Rightarrow \ldots$ of elements of $\mathcal{T}$.*

Henceforth we assume that *transitive* closure $\Rightarrow_R^+$ of the restriction of $\Rightarrow_R$ on $\mathcal{G}(\mathcal{T}) \times \mathcal{G}(\mathcal{T})$ is terminating.

An arbitrary subset Q of $\mathcal{O}(\mathcal{T})$ is called a *property*. Let q be a finite set (*collection*) of passive terms. A property Q_q defined by q is a set of all object instances of all terms from q, that is $Q_q = \{\tau \mid \tau \in \mathcal{O}(\mathcal{T}) \wedge \exists \rho \in q \; (\tau \ll \rho)\}$

We consider the following *reachability* problem on for term-rewriting systems.

Verification of safety property
Given: *A term-rewriting system $P = \langle t, R \rangle$ and property Q_q.*
Question: *Is it true that all passive terms reachable in P from any ground instance of t do not satisfy the property Q_q? In formal notation, is the statement*

$$\forall s \in \mathcal{O}(\mathcal{T}) (\forall t' \in \mathcal{O}(\mathcal{T}) ((t' \ll t) \wedge (t' \Rightarrow_R^* s)) \to s \notin Q_q)$$

true?
Many interesting verification problems for parameterized systems may be reduced to the above problem. See the section [25] for an example. In the next subsection we present a method suitable for solving such a problem and demonstrate its correctness.

3.2 Inductive Proofs of Safety Properties

Consider an instance of the above verification problem $I = (\langle t, R \rangle, Q_q)$.

[2] For simplicity we use only such kind of term-rewriting systems.

The proof of the safety property for I can be established by constructing successful *proof attempt* which consists of a sequence of trees. Vertices of trees will be labeled by terms. For a vertex a denote by t_a the term labeling a.

We assume that we have a testing procedure, which given a vertex a checks whether *all ground instances* of the t_a *do not satisfy* the property Q_q.

Another assumption is that any vertex in a tree is labelled as *unready*, *open* or *closed* (one flag per vertex) and all generated vertices are unready until they are explicitly open. We assume also that when any vertex of the proof tree labeled by a passive term is generated it is immediately tested. If the testing produces the negative result the whole proof tree building procedure stops and returns the answer NO to the verification problem, otherwise the vertex is closed.

Given a directed tree T and its edge $(a \xrightarrow{e} b)$, we say the vertex a is the *parent* of b and b is a *child* of a. A vertex a_1 is an ancestor of a vertex a_n if there exists a sequence of edges of T such that $(a_1 \xrightarrow{e} a_2), (a_2 \xrightarrow{e} a_3), \ldots, (a_{n-1} \xrightarrow{e} a_n)$.

Definition 2. *For a given $I = (\langle t, R\rangle, Q_q)$ a* proof attempt *is a sequence of directed trees $T_0, T_1, \ldots$ such that T_0 is generated by the START rule, and T_{i+1} is obtained from T_i by application of one of the following rules: UNFOLD, CLOSE, GENERALIZE.*

The proof rules are defined as follows

START Create a root of the tree, label it by the initial term t of the term rewriting system.

UNFOLD Choose any of the unready vertices a with the labeling term t_a and generate all terms $t_1, \ldots t_n$ such $t_a \Rightarrow_R t_i$. For every such t_i create a child vertex a_i for a and put $t_{a_i} = t_i$. Open the vertex a. If the parent of a is open, then close the parent.

CLOSE I Choose any of the open vertices a and check whether there is a closed vertex b, such that $t_a \ll t_b$. If yes close the vertex a and delete its children. If there are no such a b do nothing.

CLOSE II Choose any of the open vertices a and check whether all its children are closed there. If yes close the vertex a.

GENERALIZE Choose any of the open vertices a and any vertex b, ancestor of the a in the tree. Generate a term τ such that both $t_b \ll \tau$ and $t_a \ll \tau$ hold. Delete the subtree with the root b, except the vertex b itself. Replace the label t_b with τ. Mark the vertex b as unready.

Significance of the *unready* flag is related to effectiveness issue and will be considered in the section 4.

Definition 3. *A* proof *of an I is a finite proof attempt $T_0, \ldots, T_n$ for I such that all vertices in T_n are closed.*

Let R be a term rewriting system, $t \in \mathcal{T}(\mathcal{V}, \mathcal{F})$ and t_0 be a ground instance of the term t. Let $\bar{t_0} = t_0 \Rightarrow_R t_1 \Rightarrow_R t_2 \ldots \Rightarrow_R t_l$ be an arbitrary sequence of terms derived from t_0 by application of rules from R. Denote by $\mathcal{R}(t)$ the set of all passive terms reachable in R from any ground instance of t and $\mathcal{CR}(t)$ the set of all the sequences $\bar{g_0}$ such that g_0 is a ground instance of t and $g_{|\bar{g_0}|-1} \in \mathcal{R}(t)$. The following proposition is trivial.

Proposition 1. *Let t be a term and τ be a term such that $t \ll \tau$, then $\mathcal{CR}(t) \subset \mathcal{CR}(\tau)$ holds. (And hence $\mathcal{R}(t) \subset \mathcal{R}(\tau)$.)*

Theorem 1. *For an instance $I = (\langle t, R\rangle, Q_q)$ of the verification problem above if there is a proof for I then the answer for this I is YES.*

Proof Let $T_0, \ldots, T_N$ be a proof for I.

The statement of the theorem follows from the following statement. Let t_0 be a ground instance of the initial term t. Let $\bar{t} = t_0 \Rightarrow_R t_1 \Rightarrow_R t_2 \ldots \Rightarrow_R t_{l-1}$ be an arbitrary sequence of terms derived from t_0 by application of rules from R. Then every t_i is an instance of a term t_{a_i} for some vertex a_i of T. The proof of the statement is by induction on the length of the sequence. Let $|\bar{t}| = 1$, that is $\bar{t} = t_0$ By construction of the proof attempt $T_0, \ldots, T_N$ the label τ of the initial vertex of T_N is (possibly generalized several times) term t, i.e. we have $t \ll \tau$ and therefore $t_0 \ll \tau$. Notice that once a term labelling some vertex is generated it may be generalized several times later in the proof attempt by application of GENERALIZE rule.

Consider now the step of induction. Assume the statement for all sequences of the length up to some l and let $\bar{t} = t_0 \Rightarrow_R t_1 \Rightarrow_R t_2 \ldots \Rightarrow_R t_l$. By induction hypothesis we have $t_{l-1} \ll t_a$ for some vertex $a \in T_N$. Then two cases are possible.

If there are some children $a_1, \ldots, a_k$ of a in T_N then there exists some child a_j of a such that t_{a_j} is (possibly generalization of) the term t_l (by the semantics of UNFOLD rule). It follows then $t_l \ll t_{a_j}$.

If there are no children of a in T_N then t_a should be active, otherwise there could not be any term t_l such that $t_{l-1} \ll t_a$ and $t_{l-1} \Rightarrow_R t_l$. Moreover in that case the vertex a is closed by the application CLOSE I rule and there should be another vertex $b \in T_N$ such that $t_a \ll t_b$. If b has some children we repeat the argument for the previous case taking vertex b instead of a. If it does not we find yet another vertex c such that $t_a \ll t_b \ll t_c$ and repeat the argument for c. Notice that there is no more than finitely many vertices in T_N, so after finitely many steps this case is reduced to the previous one. Step of induction is proved.

It follows that any ground passive term derivable from t_0 is an instance of one of the passive terms in T_N. Since all reachable passive vertices in T_N are tested the statement of the theorem follows. $\square$

Example 1. Let $f \in \mathcal{F}n_2$, $A, B \in \mathcal{C}_1$, $x, y, x_i, y_i \in \mathcal{V}$. Consider $I = (\langle t, R\rangle, Q_q)$. Here R is:

$f(B(x), y) = f(x, B(y));$
$f(A(A(x)), y) = f(A(x), B(y));$
$f(A(B(x)), y) = y;$

q contains the only term $A(x)$ and $t = f(B(x_1), y_1)$.

Let $[\tau]$ be a vertex labelled with a term τ. We denote each closed vertex as $[\tau]^c$, each open vertex as $[\tau]^o$ and each unready vertex as $[\tau]^u$. The first proof attempt is successful: START rule gives the tree T_0 containing the only vertex $a^u = [f(B(x_1), y_1)]^u$, UNFOLD rule yields $T_1 = \{a^o, b^u = [f(x_2, B(y_1))]^u, (a^o \xrightarrow{e} b^u)\}$; after the second UNFOLD we have $T_2 = \{a^c, b^o, d_1^u = [f(x_3, B(B(y_1)))]^u, d_2^u = [f(A(x_4), B(B(y_1)))]^u, d_3^u = [B(y_1)]^c, (a^c \xrightarrow{e} b^o), (b^o \xrightarrow{e} d_1^u), (b^o \xrightarrow{e} d_2^u), (b^o \xrightarrow{e} d_3^u)\}$; now two applications of UNFOLD rule open d_1^u, d_2^u and close b^o; two applications of CLOSE rule close d_1^o and d_2^o with b^c as the witness. We have $T_6 = \{a^c, b^c, d_1^c, d_2^c, d_3^c, (a^c \xrightarrow{e} b^c), (b^c \xrightarrow{e} d_1^c), (b^c \xrightarrow{e} d_2^c), (b^c \xrightarrow{e} d_3^c)\}$, where all vertices are closed.

The second proof attempt fails: T_0, T_1, T_2 are the same as in the first attempt; GENERALIZE gives $T_3 = \{g^u = [f(x_3, y_3)]^u\}$. Now it is easy to see this attempt does not terminate.

The third proof attempt fails: T_0, T_1, T_2 are the same as in the first attempt; GENERALIZE gives $T_3 = \{g^u = [x_3]^u\}$. Now g^u can never be closed.

We show now that, in fact, the proof sequence is a compact representation of the inductive proof of the correctness condition (none of the ground instances of the reachable passive terms has the property Q_q). First, we formulate the induction scheme in general terms.

Let $\rhd$ be a well-founded partial ordering on a set $\mathcal{K}$. Let $\mathcal{M}$ is the set of all minimal elements of $\mathcal{K}$: $\mathcal{M} = \{t \in \mathcal{K} \mid \neg\exists(\tau \in \mathcal{K}).(\tau \neq t) \wedge (t \rhd \tau)\}$. Note that $\mathcal{M}$ is not empty. Let Q be a predicate on $\mathcal{K}$ and $\mathcal{S}$ be a subset of $\mathcal{K}$. The following induction scheme can be used then to prove that Q holds everywhere on $\mathcal{K}$ (we assume $y \lhd x \equiv x \rhd y$ here):

$$\frac{(\forall t \in \mathcal{M}.Q(t)) \wedge (\forall x \in \mathcal{K}.(\forall y \in \mathcal{S}.y \lhd x \to Q(y)) \to Q(x))}{\forall x \in \mathcal{K}.Q(x)}$$

Retuning to our context, let $\mathcal{L}$ is the set of the terms generated by applications of GENERALIZE rule during the proof given above and t is the initial term of the I. Let H_g be the following hypothesis: "none of the ground instances of $g \in \mathcal{L}$ reaches a passive term having the property Q_q".

Then the proof given by the successful proof attempt can be considered as simultaneous proofs of all hypotheses H_g, such that each of them follows the inductive scheme given above and moreover the proofs may refer one to another. Here $\mathcal{K} = \mathcal{O}(\mathcal{T})$, $\rhd$ is $\Rightarrow_R^+$, $Q(t) = H_t$, $\mathcal{M}_g$ is the set of all passive object terms reachable from all ground instances of g, and $\mathcal{S}_g$ is the set of the ground instances of the terms closed during applications of CLOSE rule. The subscript g indicates the concrete proof of H_g.

4 Towards Effectiveness

The proof procedure presented in the previous section is non-determinstic. That leads to necessity of development of deterministic proof strategies which would be complete and/or efficient for classes of verification problems. In this section we make first steps towards resolving these largely open issues, and present the strategies which empirically has turned out to be sufficient for (practically efficient) proofs of correctness of cache coherence protocols [5].

The second proof attempt given in the Example 1 demonstrates that critical information may be lost during an application of GENERALIZE rule. The information guaranteed transformation of the initial term uniformly on the values of the parameters. The start vertex is not a branching point: there exists the only edge outgoing from the vertex. Terminating of $\Rightarrow_R^+$ (see the section 3.1) means there cannot be an infinite sequence of such kind of vertices one after another. Thus it is desirable to exclude such vertices from generalization.

Definition 4. *An open or closed vertex b is* pivot *in a tree T_j iff b has at least two outgoing edges.*

A closed vertex can be both basic and pivot. Henceforth we impose the following retsriction on the strategy of rule applications: *both CLOSE and GENERALIZE rules choose only pivot vertices.*

Given two terms t_1 and t_2 there can be a number of different generalizations, see example 1 for the illustration. Aiming to preserve as much as possible the structure of the terms, we impose the next restriction on GENERALIZATION rule: *result of generalization of any two terms t_1 and t_2 should be most specific term τ*, meaning both $t_1 \ll \tau$ and $t_2 \ll \tau$ hold and for any other term ξ such that $(t_1 \ll \xi) \wedge (t_2 \ll \xi \ll \tau)$ implies that ξ equals to τ modulo variable's names.

Further restriction is concerned with the choice of terms to be generalized. In order to preserve the structure of terms it is natural and desirable to generalize only terms, which are similar (in a sense) one to another. There is delicate trade-off here. Informally, the fewer applications of GENERALIZE rule happened during a proof attempt the less information on the terms structure is lost and more chances to close the passive vertices. On the other hand, to close active vertices one may need more applications of GENERALIZE rule. The following criteria based on well-quasi-ordering have turned out to be empirically successful.

A *quasi-ordering* is any reflexive and transitive binary relation.

Definition 5. *A quasi-ordering $\preceq$ on a set T is a* well-quasi-ordering *if every infinite sequence $t_1, t_2, \ldots$ of elements of T contains t_i, t_j $(i < j)$ such that $t_i \preceq t_j$.*

Given a well-quasi-ordering $\preceq$ on $\mathcal{T}(\mathcal{V}, \mathcal{F})$, we specify the strategy choosing the vertices by GENERALIZE rule as follows: *choose any of the pivot open vertices a and any pivot vertex b, ancestor of the a in the tree such that $t_b \preceq t_a$; if there exists no such a b do nothing.*

Further, there can be a number of such vertices b. Intuitively, the closer a vertex b to the vertex a (among the all its ancestors) the closer any ground instance of the b to a passive ground term terminating evaluation of the instance by the term-rewriting system. So we add to the above generalization strategy the requirement *to choose the closest such a vertex* b.

All our experiments verifying the class parameterized protocols [5] were successful both under lazy (call by need) and under applicative (call by value) strategies developing the stack of functions. For simplicity we selected the applicative strategy to demonstrate the main example given in the section [25]. We encode this semantic concept in syntax as follows. Given a composition $t = f(\dots, g(\dots), \dots)$, where $f, g \in \mathcal{F}n$, we transform the term to

$$\texttt{Let}(x,\ \texttt{eq},\ g(\dots),\ \texttt{in},\ f(\dots,\ x,\ \dots)),$$

$\texttt{Let} \in \mathcal{F}n$ is a auxiliary name. The term $g(\dots)$ is transformed recursively in the same fashion. We note the semantics of both the t and the transformed term is the same. We stress that without such representation of the composition the other strategies do not lead to successful experiments with the cache coherence protocols.

5 A Free Monoid of Terms

In this section we consider a free monoid of terms, which was actually used in our experiments. Using this data structure and concepts and strategies given above allows to obtain automatic proofs of correctness of cache coherence protocols from [5]. See also remarks in Section 6.

We construct the monoid from $\mathcal{T}(\mathcal{V}, \mathcal{F})$ by minor modification of definition. Let all the function names be unary $\mathcal{F}n_1$, while the constructor set be $\mathcal{C} = \mathcal{C}_2 \cup \mathcal{C}_1 \cup \mathcal{C}_0$. Let us denote terms constructed with a $f \in \mathcal{F}n_1$ as $\texttt{<}f\ \ t\texttt{>}$, where t is a term. Let $\mathcal{C}_2$ contains the only *associative* element named as concatenation, used in infix notation and denoted with the blank. The associativity allows to drop the parentheses of the constructor at all. Let $\mathcal{C}_1$ contains the only constructor, which denoted only with its parentheses (that is without a name). $\mathcal{C}_0 = \mathcal{K} \cup \{\lambda\}$. We denote the constants from $\mathcal{K}$ with its names: that is without the parentheses. The constant λ is denoted with nothing: it is the unit of the concatenation. Let the variable set $\mathcal{V}$ be disjoined in two sets $\mathcal{V} = \mathcal{E} \cup \mathcal{S}$, where the names from $\mathcal{E}$ are prefixed with 'e.', while the names from $\mathcal{S}$ – with 's.'. For a term t we denote the set all e-variables (s-variables) in t by $\mathcal{E}(t)$ (correspondingly $\mathcal{S}(t)$). $\mathcal{V}(t) = \mathcal{E}(t) \cup \mathcal{S}(t)$. The monoid of the terms may be defined with the following grammar:

t `::=` λ `|` c `|` v `|` `<`f t`>` `|` t_1 t_2 `|` `(`t`)`
λ `::=`

where $c \in \mathcal{K}$, $v \in \mathcal{V}, f \in \mathcal{F}n_1$. Thus a term is a finite sequence (including the empty sequence). We denote the constructed free monoid as $\mathcal{A}(\mathcal{V}, \mathcal{F})$. Any substitution has to map every $v \in \mathcal{S}$ into $\mathcal{K} \cup \mathcal{S}$.

5.1 Restrictions on Term-rewriting Systems

Given a term-rewriting system $\langle t, R\rangle$ on the set $\mathcal{A}(\mathcal{V},\mathcal{F})$. Associativity of the concatenation simplifies the syntax structure of the terms, but it creates a problem with the one-step transition relation $\Rightarrow_R \subseteq \mathcal{A}(\mathcal{V},\mathcal{F})\times\mathcal{A}(\mathcal{V},\mathcal{F})$, namely, given a term τ and a rule $(l \to r) \in R$, then there can be several substitutions matching τ with the l. Thus we have a new kind of non-determinism here. An example is as follows:

Example 2. $\tau = \texttt{<}f\ \ A\texttt{>}$ and $l = \texttt{<}f\ \ e.x\ \ e.y\texttt{>}$, where $A \in \mathcal{K}$, $e.x, e.y \in \mathcal{E} \subset \mathcal{V}$. There exist two substitutions matching the terms: the first is $\theta_1(e.x) = \lambda, \theta_1(e.y) = A$, the second is $\theta_2(e.x) = A, \theta_1(e.y) = \lambda$.

Multiplicity of $v \in \mathcal{V}$ in a term t is the number of occurrences of v in t. A variable $x \in \mathcal{E}(t)$ is closed in a term t iff (1) $t = (t_1)$ and x is closed in t_1; (2) $t = t_1 \ldots t_n$, where there exists at most one $t_i = x$ and $\forall j$ the x is closed in t_j.

We impose the following restriction on the left sides of the rules from R. The multiplicity of any $v \in \mathcal{E}(l)$ equals 1 and v is closed in l. These restrictions exclude recursive equations that have to be solved when we are looking for the substitutions matching a given parameterized term with a left side of a rule. The following term $\tau = \texttt{<}f\ \ (A\ \ e.p)\ \ (e.p\ \ A)\texttt{>}$ and $l = \texttt{<}f\ \ (e.x)\ \ (e.x)\texttt{>}$ is an example showing that the recursive equation $A\ e.p = e.p\ A$ arises on $e.p$; the reason of the recursion is the fact that the multiplicity of $e.x \in \mathcal{E}(l)$ is 2.[3] The second example $\tau = \texttt{<}f\ \ e.p\texttt{>}$ and $l = \texttt{<}f\ \ e.x\ \ A\ \ e.y\texttt{>}$ as well as the example 2 demonstrate the problems, which are caused by unclosed variables (here both $e.x$ and $e.y$) in the left hand-sides of the rules.

Consider the example 2. Let us think of the τ as a left part of a rule ρ, while of the l as a term to be match with τ with the goal to unfold. The both substitutions given in the example 2 match the term l with τ. Hence, during the application of UNFOLD rule we have to take into account the both substitutions and generate two children of l from ρ. We solve this problem with the following additional sub-rule:

SPLIT Given a term t to be unfolded (with a rule $\rho = (l \to r)$) such that $\mathcal{E}(t)$ includes unclosed variables. Take a subterm of t of the form $\xi = t_1\ \ldots\ e.x\ \ldots\ e.y\ \ldots t_n$ (i.e. the both variables $e.x, e.y \in \mathcal{E}(t)$ are not enclosed with the parenthesis) such that there exist at least two substitutions which match ξ with the corresponding subterm of l. Generate the following three substitutions $\theta_1(e.x) = s.n\ \ e.x_1, \theta_2(e.x) = (e.z)\ \ e.x_2, \theta_3(e.x) = \lambda$. Here $e.x_1, e.x_2, e.z$ are fresh variables from $\mathcal{E}$, $s.n$ is a fresh variable from $\mathcal{S}$. Unfold $t\theta_i$ with the rule ρ.

The SPLIT rule is recursive and terminates. See the section [25] for the examples using this rule.

The example 2 shows also another problem. The term l may be considered as a generalization of the term τ: $\tau \ll l$. The problems is: there exists a term

[3] The solution of the equation is $e.p = A^*$.

$\xi = e.z$ such that $l \neq \xi, \tau \ll \xi$ and both $l \ll \xi$ and $\xi \ll l$ hold. Now we specify generalization. Given two terms $t_1, t_2 \in \mathcal{A}(\mathcal{V}, \mathcal{F})$ and the set G of all the most specific terms generalizing both t_1 and t_2 (see the section 4). Let $\nu_x(t)$ be the multiplicity of an $x \in \mathcal{E}(t)$. We use (as the result of generalization of t_1, t_2) $g \in G$ such that $\sum\limits_{x \in \mathcal{E}(g)} \nu_x(g)$ is minimal over G.

5.2 The Well-quasi-ordering on $\mathcal{A}(\mathcal{V}, \mathcal{F})$

Given $t_1, t_2 \in \mathcal{A}(\mathcal{V}, \mathcal{F})$, there exist two elementary functions constructing a new term from the given term. The functions are $F_1(t_1, t_2) = t_1\ t_2$ and $F_2(t_1) = \texttt{(}t_1\texttt{)}$. There exists also a family of functions $F_f(t_1) = \texttt{<}f\ \ t_1\texttt{>}$, where $f \in \mathcal{F}n_1$. We consider a quasi-ordering such that: (1) with respect to it all these functions are monotone non-decreasing $t_1 \underline{\propto} F_1(t_1, t_2), t_1 \underline{\propto} F_2(t_1), t_1 \underline{\propto} F_f(t_1)$; (2) these functions are matched with the quasi-ordering: $t_1 \underline{\propto} t_2$ implies $F_2(t_1) \underline{\propto} F_2(t_2), F_f(t_1) \underline{\propto} F_f(t_2)$ and for any term t both $F_1(t, t_1) \underline{\propto} F_1(t, t_2)$ and $F_1(t_1, t) \underline{\propto} F_1(t_2, t)$ hold. The following relation is a variant of the Higman-Kruskal relation and is a well-quasi-ordering [12, 15].

Definition 6. *The homeomorphic embedding relation* $\underline{\propto}$ *is the smallest transitive relation on* $\mathcal{A}(\mathcal{V}, \mathcal{F})$ *satisfying the following properties, where* $h \in \mathcal{F}n_1, s, t, t_i \in \mathcal{A}(\mathcal{V}, \mathcal{F})$.

1. $\forall x, y \in \mathcal{E}.\ x \underline{\propto} y, \forall u, v \in \mathcal{S}.\ u \underline{\propto} v$;
2. $t \underline{\propto} \texttt{<}h\ t\texttt{>}, t \underline{\propto} \texttt{(}t\texttt{)}, t \underline{\propto} s\ t, t \underline{\propto} t\ s$;
3. $s \underline{\propto} t$, *then* $\texttt{<}h\ s\texttt{>} \underline{\propto} \texttt{<}h\ t\texttt{>}$, $\texttt{(}s\texttt{)} \underline{\propto} \texttt{(}t\texttt{)}$, $s\ t_1 \underline{\propto} t\ t_1, t_1\ s \underline{\propto} t_1\ t$.

Corollary 1. *1.* $\lambda \underline{\propto} t \underline{\propto} t$, *where* λ *is the empty sequence;*
2. $\exists i_1, \ldots, i_j$ *such that* $1 \leq i_1 < i_2 < \ldots < i_j \leq n$, *then* $t_{i_1} \ldots t_{i_j} \underline{\propto} t_1 \ldots t_n$.

Given an infinite sequence of terms $t_1, \ldots, t_n, \ldots$, this relation is relevant to approximation of increasing loops in the sequence; or in other words to looking for the regular similar cases of mathematical induction on the structure of the terms. That is to say the cases, which allow refer one to another by a step of the induction. An additional restriction separates the basic cases of the induction from the regular ones. The restriction is:

$$\forall c \in \mathcal{K}.\texttt{()} \underline{\not\propto} \texttt{(}c\texttt{)} \wedge \forall v \in \mathcal{S}.\texttt{()} \underline{\not\propto} \texttt{(}v\texttt{)}.$$

We impose this restriction on the relation $\underline{\propto}$ and denote the obtained relation as $\preceq$. It is easy to see that such a restriction does not violate the quasi-ordering property. Note that the restriction may be varied in the obvious way, but for our experiments its simplest case given above is used to control applications of GENERALIZE rule and has turned out to be sufficient.

6 Discussion

In addition to the MOESI protocol described in the Appendix [25] the supercompiler SCP4 verified by our scheme the following parameterized cache coherence protocols: IEEE Futerbus+, MESI, MSI, "The University of Illions", DEC Firefly, "Berkeley", Xerox PARC Dragon [5, 6]. All these protocols are specified analogously to the description given in the section [25]. In the case of the MOESI protocol the time of automatic verification is 1 second (Windows XP/Service Pack 2, Intel Pentium III, 450 MHz, 256 MB of RAM); verification of the other protocols takes times, which slightly differ from the indicated.

One of the questions left open is why do we work in terms of the free monoid $\mathcal{A}(\mathcal{V}, \mathcal{F})$ and how important is such a choice? Actually, supercompiler SCP4 is able to prove correctness of the main example considered in the section [25] encoded in terms of a free algebra terms too, but the proof is much more bulky as compared with the proof presented in section [*The Inductive Proof*] of [25] Moreover, the proof in this case requires additional capabilities of the supercompiler which are not presented in our formal model. We leave detailed analsys and comparisons of different encodings to future work.

The work reported in this paper has started as mainly driven by experiments. There is still much work to be done, both theoretically and experimentally. On the theory side we would like to have the completeness results for classes of verification problems and particular strategies. The applicability of the strategies already implemented in SCP4 is also worth to explore further. Recent experiments have shown that SCP4 strategies are quite robust with respect to order in wich rewriting rules are encoded in Refal programs. For example, the above MOESI protocol can be verifed with any of 120 (=5!) permutations of rewriting rules for `RandomAction`. See [24] for details to this subject.

Finally, comparisons with related work, especially with [11, 19, 20] and [31] should be done. In both these approaches transformations of logic (as opposed to our functional) programs are used to perform verification of parameterized systems. Despite the differences in programing languages, systems encodings and verifications schemes used, all three approaches have a common ground and rely on variants of unfold/fold transformations.

References

1. Abdulla, P.A., Jonsson, B., Nilsson, M., Saksena, M.: A Survey of Regular Model Checking. In *Proc. 15th Int. Conf. on Concurrency Theory*, LNCS, 2004.
2. Baukus, K., Stahl, K., Bensalem, S., Lakhnech, Y.: Networks of Processes with Parameterized State Space. In *Electronic Notes in Theoretical Computer Science*, January 2004, vol. 50, no.4, pp 1–15.
3. Bjorner, N., Browne, A., Chang, E., Colon, M., Kapur, A., Manna, Z., Sipma, H.B., Uribe T.E.: STeP: Deductive-Algorithmic Verification of Reactive and Real-time Systems. In *Proc. International Conference on Computer Aided Verification, CAV'96*, vol. 1102 of LNCS, Springer-Verlag, pp.415-418, 1996
4. Clarke, E.M., Grumberg, Peled, D.: *Model Checking.* MIT Press, 1999

5. Delzanno, G.: Automatic Verification of Parameterized Cache Coherence Protocols. In *Proc. of the 12th Int. Conf. on Computer Aided Verification*, LNCS, vol. 1855, pp. 53-68 (2000)
6. Delzanno, G.: Automatic Verification of Cache Coherence Protocols via Infinite-state Constraint-based Model Checking,
http://www.disi.unige.it/person/DelzannoG/protocol.html.
7. Delzanno, G.: Verification of Consistency Protocols via Infinite-state Symbolic Model Checking, A Case Study. In *Proc. of FORTE/PSTV*, 2000, pp: 171-188.
8. Delzanno, G.: Contsraint-based Verification of Paremeterized Cache Coherence Protocols. *Formal Methods in System Design* 23(3):257-301, 2003.
9. Ershov, A.P.: Mixed computation in the class of recursive program schema. *Acta Cybernetica*, 4(1), 1978.
10. Esparza, J.: Decidability of model checking of infinite state concurrent systems. *Acta Informatica*, 34:85-107, 1997.
11. Glück, R., Leuschel, M.: Abstraction-based partial deduction for solving inverse problems – a transformational approach to software verification. In *Proc. of Systems Informatics*, LNCS 1755, pages 93-100, Novosibirsk, Russia, 1999. Springer-Verlag.
12. Higman, G.: Ordering by divisibility in abstract algebras. Proc. London Math. Soc. **2(7)** (1952) 326–336
13. Jones, N.D., Gomard, C.K., Sestoft, P.: Partial Evaluation and Automatic Program Generation. (1993) Prentice Hall International
14. Korlyukov, A.V., Nemytykh, A.P.: Supercompilation of Double Interpretation. (How One Hour of the Machine's Time Can Be Turned to One Second), 2002.
http://www.refal.net/~korlukov/scp2int/Karliukou_Nemytykh.pdf.
Sources, demonstration: www.refal.net/~korlukov/demo_scp4xslt.zip.
15. Kruskal, J.B.: Well-quasi-ordering, the tree theorem, and vazsonyi's conjecture. Trans. Amer. Math. Society, **95** (1960) 210–225
16. Leuschel, M., Martens, B.: Global Control for Partial Deduction through Characteristic Atoms and Global Trees. *Proceeding of the PEPM'96*, LNCS 1110, Springer-Verlag, 1996.
17. Leuschel, M.: On the Power of Homeomorphic Embedding for Online Termination. In *Proc. of the SAS'98*, LNCS 1503, 1998.
18. Leuschel, M., Lehmann, H.: Program Specialization, Inductive Theorem Proving and Infinite State Model Checking, Invited talk,LOPSTR'03, Uppsala, 2003, avaialable at:www.ecs.soton.ac.uk/~mal/presentations/ITP_Lopstr03.ppt.
19. Leuschel, M., Lehmann, H.: Solving coverability problems of Petri nets by partial deduction. In *Proc. 2nd Int. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming (PPDP'2000)*, Montreal, Canada, pp: 268-279, 2000.
20. Leuschel, M., Massart, T.: Infinite state model checking by abstract interpretation and program specialisation. In A. Bossi, editor, *Logic-Based Program Synthesis and Transformation. In Proc. of LOPSTR'99*, LNCS 1817, pages 63-82, Venice, Italy, 2000.
21. Lisitsa, A.P., Nemytykh, A.P.: Towards Verification via Supercompilation. In *Proc. of COMPSAC 05, the 29th Annual International Computer Software and Applications Conference, Workshop Papers and Fast Abstracts*, pages 9-10, IEEE, 2005.
22. Lisitsa, A.P., Nemytykh, A.P.: Verification via Supercompilation.
`http://www.csc.liv.ac.uk/~alexei/VeriSuper/`
23. Lisitsa, A.P., Nemytykh, A.P.: Verification as a Parameterized Testing (Experiments with the SCP4 Supercompiler). Programmirovanie. No.**1** (2007) (In Rus-

sian). English translation in J. Programming and Computer Software, Vol. **33**, No.1 (2007) 14–23, Pleiades Publishing, Ltd.
24. Lisitsa, A.P., Nemytykh, A.P.: A Note on Specialization of Interpreters. Accepted by the 2nd International Computer Science Symposium in Russia - CSR07.
25. Lisitsa, A.P., Nemytykh, A.P.: Appendix to "Reachability Analysis in Verification via Supercompilation". Avaialable at: http://www.botik.ru/pub/local/scp/refal5/rp07_appendix.pdf
26. Nemytykh, A.P.: A Note on Elimination of Simplest Recursions.. In Proc. of the ACM SIGPLAN Asian Symposium on Partial Evaluation and Semantics-Based Program Manipulation, (2002) 138–146, ACM Press
27. Nemytykh, A.P.: The Supercompiler SCP4: General Structure (extended abstract). In Proc. of the Perspectives of System Informatics, LNCS, **2890** (2003) 162–170, Springer-Verlag
28. Nemytykh, A.P.: Playing on REFAL. In: Proc. of the International Workshop on Program Understanding. A.P. Ershov Institute of Informatics Systems, Syberian Branch of Russian Academy of Sciences, pp:29-39, July 2003. (ftp://www.botik.ru/pub/local/scp/refal5/nemytykh_PU03.ps.gz)
29. Nemytykh, A.P., Turchin, V.F.: The Supercompiler SCP4: sources, on-line demonstration, http://www.botik.ru/pub/local/scp/refal5/, (2000).
30. Romanenko, S.A.: Arity raiser and its use in program specialization. *Proceeding of the ESOP'90*, LNCS, 432:341–360, 1990.
31. Roychoudhury A., Ramakrishnan C.R.: Unfold/fold Transformations for Automated Verification of Parameterized Concurrent Systems. In *Program Development in Computational Logic*, LNCS 3049, 2004, pp 262-291, 2004
32. Sands, D.: Proving the correctness of recursion-based automatic program transformation. In *Theory and Practice of Software Development*, volume 915 of LNCS, pp 681–695, 1995.
33. Sørensen, M.H., Glück, R.: An algorithm of generalization in positive supercompilation. In *Logic Programming: Proceedings of the 1995 International Symposium*, pages 486–479. MIT Press, 1995.
34. Sørensen, M.H., Glück, R.: Introduction to Supercompilation. *Partial Evaluation - Practice and Theory*, DIKU 1998 International Summer School. June 1998. http://repository.readscheme.org/ftp/papers/pe98-school/D-364.pdf
35. Sørensen, M.H., Glück, R., Jones, N.D.: A positive supercompiler. In *Journal of Functional Programming*, **6(6)** (1996) 811–838
36. Turchin, V.F.: The concept of a supercompiler. *ACM Transactions on Programming Languages and Systems*, 8:292–325, 1986.
37. Turchin, V.F.: The algorithm of generalization in the supercompiler. In *Proceedings of the IFIP TC2 Workshop, Partial Evaluation and Mixed Computation*, pages 531–549. Amsterdam: North-Holland Publishing Co., 1988.
38. Turchin, V.F.: Refal-5, Programming Guide and Reference Manual. Holyoke, Massachusetts. (1989) New England Publishing Co. (electronic version: http://www.botik.ru/pub/local/scp/refal5/ ,2000)
39. Turchin, V.F.: Metacomputation: Metasystem transition plus supercompilation. *Proceeding of the PEPM'96*, LNCS, Springer-Verlag, 1110:481–509, 1996.
40. Turchin, V.F., Turchin, D.V., Konyshev, A.P., Nemytykh, A.P.: Refal-5: sources, executable modules. http://www.botik.ru/pub/local/scp/refal5/, (2000)
41. Wadler, P.: Deforestation: Transforming programs to eliminate tree. *Theoretical Computer Science*, 73:231–238, 1990.

The Finite Tiling Problem Is Undecidable in the Hyperbolic Plane

Maurice Margenstern[1]

Laboratoire d'Informatique Théorique et Appliquée, EA 3097,
Université de Metz, I.U.T. de Metz,
Département d'Informatique,
Île du Saulcy,
57045 Metz Cedex, France,
`margens@univ-metz.fr`

Abstract. In this paper, we consider the finite tiling problem which was proved undecidable in the Euclidean plane by Jarkko Kari, see [5]. Here, we prove that the same problem for the hyperbolic plane is also undecidable.

Keywords: hyperbolic plane, tilings, finite tiling problem

1 Introduction

A lot of problems deal with tilings. Most of them are considered in the setting of the Euclidean plane. A certain number of these problems turn out to be undecidable in this frame, thanks to the facility to simulate the computation of a Turing machine in this setting. The most famous case of such a problem is the **general tiling problem** proved to be undecidable by Berger in 1966, see [1]. In 1971, R. Robinson gave a simplified proof of the same result, see [18]. Sometimes, the general problem is simply called the **tiling problem**. The reason of these different names lies in the fact that several conditions were put on the problem, leading to different settings, and a dedicated proof was required each time when the problem turned out to be undecidable. Among these variations, the most well-known is the **origin-constrained** problem, proved to be undecidable by Wang in 1958, see [21].

The general tiling prolem consists in the following. Given a finite set of tiles T, is there an algorithm which says whether it is possible to tile the plane with copies of the tiles of T or not? The **origin-constrained** problem consists in the same question to which a condition is appended: given a finite set of tiles T and a tile $T_0 \in T$, is there an algorithm which says whether it is possible or not to tile the plane with copies of the tiles of T or not, the first tile being T_0? In the general problem there is no condition on the first tile: it can be a copy of any tile of T.

There are a lot of variants of these problems and the reader is referred to [18], where an account is given on several such conditions.

The **finite tiling problem** is a slightly different question. Given a finite set of tiles T and a tile b with $b \notin T$, called the **blank**, is there an algorithm which says whether there is a tiling of the plane with copies of $T \cup \{b\}$ in which there are only finitely many copies of tiles of T but at least one? This problem was first formulated by J. Kari in [5], where it was proved to be undecidable. In the case of the general tiling problem, a **solution** to the problem is a tiling of the plane with copies of the tiles from T only. In the case of the finite tiling problem, a **solution** to the problem is a tiling in which only finitely many copies of T are used and at least one is used.

The general tiling problem for the hyperbolic plane was raised by R. Robinson in his 1971 paper, see [18]. In 1978, R. Robinson proved that the origin-constrained problem is undecidable in the hyperbolic plane, see [19]. The general tiling problem for the hyperbolic plane remained pending a long time. In 2006, the present author proved that the tiling problem with an intermediate condition, so called **generalized origin-constrained problem** is undecidable, see [9, 10]. Recently, the present author proved the general tiling problem to be undecidable in the hyperbolic plane, see [11, 14]. At the same time, J. Kari announced the same result, using a completely different approach, see [6].

In this paper, we prove that:

Theorem 1 *The finite tiling problem is undecidable in the hyperbolic plane.*

As we shall see, the solution makes use of the technique used in [9, 10].

In the next section, we remind the reader with necessary preliminaries to make the paper self-contained. Then, in Section 2, we sketchilly remind the solution to the origin-constrained problem which we have given in [9, 10]. In Section 3, we prove the theorem. In the conclusion, we discuss a few possible issues.

2 Preliminaries

In this section, we remind the basics of hyperbolic geometry and a few other features about tilings in this setting. We conclude the section with a few explanations on the space-time diagram of a Turing machine.

2.1 The hyperbolic plane

Hyperbolic geometry appeared in the first half of the 19th century, as the conclusion of the very long quest to prove the famous axiom of parallels of Euclid's *Elements* from the other axioms. As presently known, the axiom on parallels is independent from the others. The discovery of hyperbolic geometry also raised the notion of independency in an axiomatic theory. In the second half of the 19th century, several models were devised in which the axioms of hyperbolic geometry are satisfied. Among these models, Poincaré's models became very popular. One model makes use of the half-plane in the Euclidean plane, the other makes use of a disc, also in the Euclidean plane. Each time we shall need to refer to a model, especially for illustrations, we shall use Poincaré's disc model.

Let us fix an open disc U of the Euclidean plane. Its points constitute the points of the hyperbolic plane $I\!H^2$. The border of U, ∂U, is called the set of **points at infinity**. Lines are the trace in U of its diameters or the trace in U of circles which are orthogonal to ∂U. The model has a very remarkable property, which it shares with the half-plane model: hyperbolic angles between lines are the euclidean angles between the corresponding circles. The model is easily generalised to higher dimension, see [8] for definitions and properties of such generalizations as well as references for further reading.

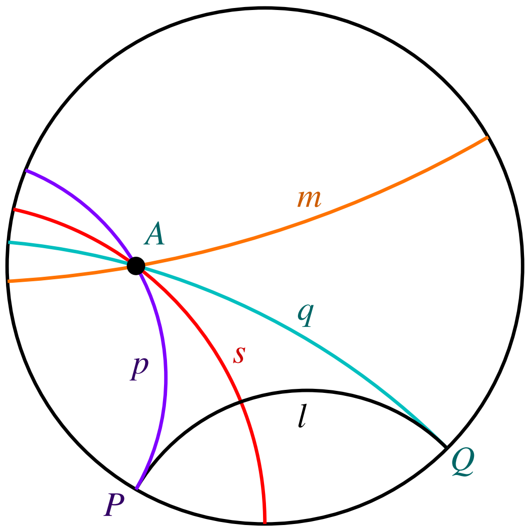


Figure 1 *An illustration of the Poincaré model.*

On Figure 1, the lines p and q pass through the point A and they are parallel to the line ℓ. We notice that each of them has a common point at infinity with ℓ: P in the case of p and Q in the case of q. The line s which also passes through A cuts the line ℓ: it is a **secant** to this line. However, the line m, which also passes through A does not meet ℓ, neither in U, nor at infinity, *i.e.* on ∂U. Such a line is called **non-secant** with ℓ. Non-secant lines have a nice characterisitic property: two lines are non-secant if and only if they have a common perpendicular which is unique.

2.2 A tiling of the hyperbolic plane: the ternary heptagrid

Tessellations are a particular case of tilings. They are generated from a regular polygon by reflection in tis sides and, recursively, of the images in their sides. In the Euclidean case, there are, up to isomorphism and up to similarities, three tesselations, respectively based on the square, the equilateral triangle and on the regular hexagon.

In the hyperbolic plane, there are infinitely many tessellations. They are based on the regular polygons with p sides and with $\frac{2\pi}{q}$ as vertex angle and they are denoted by $\{p,q\}$. This is a consequence of a famous theorem by Poincaré which characterizes the triangles starting from which a tiling can be generated by the recursive reflection process which we already mentioned. Any triangle tiles the hyperbolic plane if its vertex angles are of the form $\frac{\pi}{p}$, $\frac{\pi}{q}$ and $\frac{\pi}{r}$ with the condition that $\frac{1}{p}+\frac{1}{q}+\frac{1}{r} < 1$.

Among these tilings, we choose the tiling $\{7,3\}$ which we called the thernary heptagrid in [2]. It is below illustrated by Figure 2.

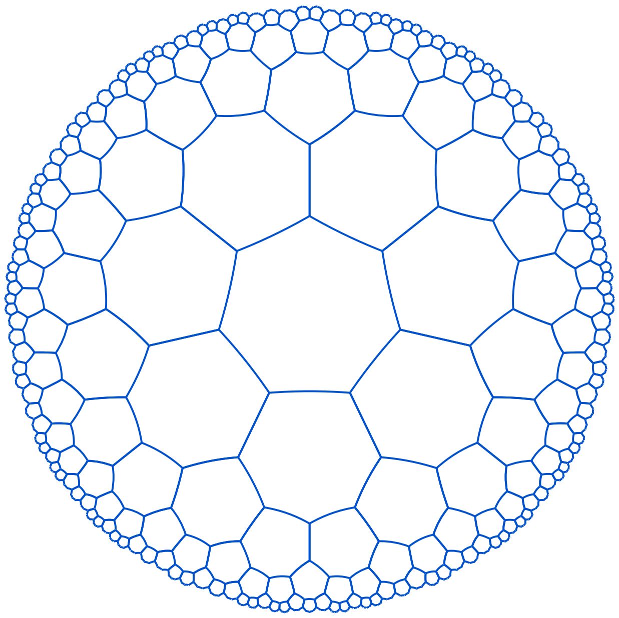


Figure 2 *The tiling* $\{7,3\}$ *of the hyperbolic plane in the Poincaré's disc model.*

In [2, 8], many properties of the ternary heptagrid are described. An important tool to establish them is the splitting method, prefigured in [7] and for which we refer to [8]. Here, we just suggest the use of this method which allows to exhibit a tree, spanning the tiling: the **Fibonacci tree**. Below, the left-hand side of Figure 3 illustrates the splitting of $I\!H^2$ into a central tile T and seven sectors dispatched around T. Each sector is spanned by a Fiboncacci tree. The right-hand side of Figure 3. Illustrates how the sector can be split into sub-regions. Now, we notice that two of these regions are copies of the same sector and that the third region S can be split into a tile and then a copy of a sector and a copy of S. Such a process gives rise to a tree which is in bijection with the tiles of the sector. The tree structure will be used in the sequence and other illustrations will allow the reader to better understand the process.

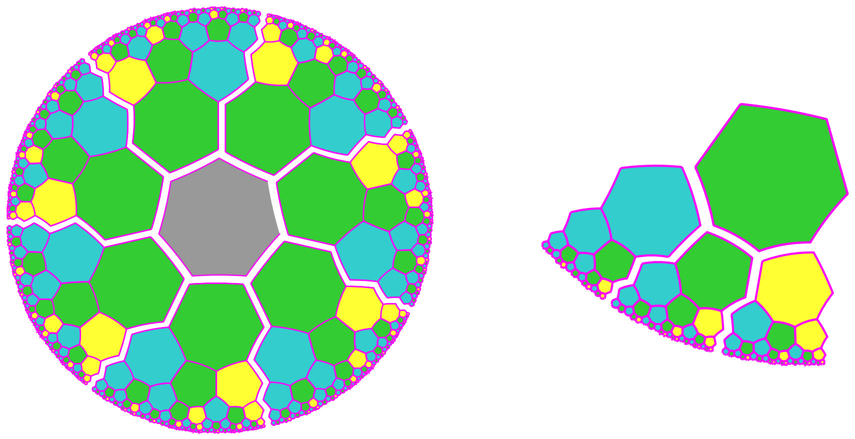


Figure 3 *Left-hand side: the standard Fibonacci trees which span the tiling* $\{7,3\}$ *of the hyperbolic plane. Right-hand side: the mid-point lines.*

Another important tool to study the tiling $\{7,3\}$ is given by the **mid-point** lines, which are illustrated by Figure 4, below. The lines have this name because they join the mid-points of contiguous edges of tiles. We can see on the figure how such lines allow to delimit a sector, a property which is proved in [2, 8].

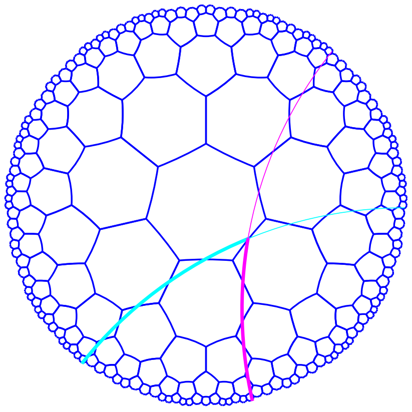


Figure 4 *The mid-point lines.*

2.3 The space-time diagram of a Turing machine

As our proof of Theorem 1 makes use of the simulation of a Turing machine, we skecthily remind a few features of the Turing machinesi, see [20, 4]. We shall specifically remind the use of space-time diagrams.

A Turing machine is a device which consists of an infinite **tape** and of a **head**. The tape is constituted of adjacent **squares**, each one containing a symbol belonging to a finite alphabet A. The head looked at a square, the **scanned square** and it is in a given **state**. Depending on its state and on the symbole read in the scanned square, the machine performs an action specified by an **instruction**: replace the read letter by a new letter, possibly the same one, turn to a new state, possibly the same one, and go to the next scanned square which is either the right-hand side neighbour of the scanned square, or its left-hand side neighbour or again, the same scanned square. There are finitely many instructions whose set constitutes the **program** of the machine. One symbol of A plays a special rôle. It is called the **blank** and a square containing the blank is called **empty**.

The configuration of a Turing machine is defined by two squares: the leftmost and the rightmost ones, called the **borders** of the configuration. For the initial configuration, these borders define the smallest finite interval of the tape which contains both the square scanned by the head of the machine and all the non-empty squares. It is assumed that at the initial time, there are finitely non-empty squares in the tape. When the borders of the current configuration are defined, the borders of the next configuration are either the same or one of them is move by one square further: this is the case when the head scans a border and when its action moves outside the current configuration.

Now, a **space-time diagram** of the execution of a Turing machine M consists in placing successive configurations of the computation of M on the plane. The initial configuration C_0 is put along the X-axis with the origin at a fixed position of the tape: it corresponds to the ordinate 0. The configuration i which is obtained after i steps of computation is placed on the parallel to the X-axis which has the ordinate i.

As can immediately be seen, the important feature is not that we have strictly parallel lines, and that squares are alined along lines which are perpendicular to the tapes. What is important is that we have a **grid**, which may be a more or less distorted image of the just described representation.

3 The harp

The solution of the origin-constrained problem which we now consider takes place in the tiling $\{7,3\}$ of the hyperbolic plane. We shall now see how it is possible to implement a kind of grid which will allow us to simulate the space-time diagram of a Turing machine. This is an important ingredient in the proof of Theorem 1.

We use the angular sector determining a Fibonacci tree to construct a frame in which we insert the space-time diagram of a Turing machine working on a semi-infinite tape and starting its computation from an empty tape.

Such an angular sector is represented on the left-hand side of Figure 5 by the two thick yellow rays supported by mid-point lines.

On the right-hand side of the figure, we can see the **harp** itself. It contains a Fibonacci tree which can be viewed as a space time diagram of the Turing computation. The rightmost branch of the tree represents the Turing tape at the initial time, call it the **frame**. In the right-hand side of Figure 5, the frame consists of the dark green tiles, the first four of them being numbered 1, 4, 12 and 33. Define **levels** in the Fibonacci tree as the set of tiles which are at the same distance, in tiles, of the root of the tree. The distance from T to the root is the smallest length of a sequence of tiles joining T to the root with the condition that two consecutive terms of the sequence share a common side. We call **borders** of the tree the two mid-point lines which determine it. Note that each level meets the right-hand side border in two points, say A and B, with, for instance, A being the closest to the root. From B, we define the other mid-point line determined by B and we consider the ray r_B of this line which is issued from B and which goes inside the tree. We call **chord** the set of tiles which belong to the tree, which meet r_B and which are in the half-plane defined by r_B which does not contain the root of the tree. A chord represents the evolution, in time, of the square of the tape: the one which is associated with the tile of the chord which is in contact with the right-hand side border of the tree.

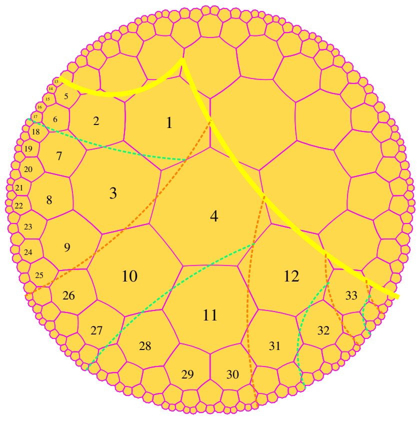


Figure 5 *The guidelines for the harp.*

Now, the simulation of the computation of a Turing machine goes as follows.

The starting point of the computation is the tile which bears the root of the tree: it is the single tile which is in contact with both borders.

Next, the computing signal, which bears the current state of the machine head, goes to the middle son of the root and there, on the level 1, it goes to the right as the head of the Turing machine never goes to the left of its initial position. The tile which is immediately reached on the same level is a tile of the border. At this point, it performs the required instruction.

Now, assume that the computing signal just performed an instruction: it is on a chord. The tile sends the new content of the tape to the next tile on the chord. The computing signal has now the new current state. It goes down along the chord by one level and, on the new level, it goes in the direction indicated by the instruction it has just performed. It remains on the same level until it meets the next chord, in the direction which it follows. When the chord is met, it performs the instruction. And the process is repeated.

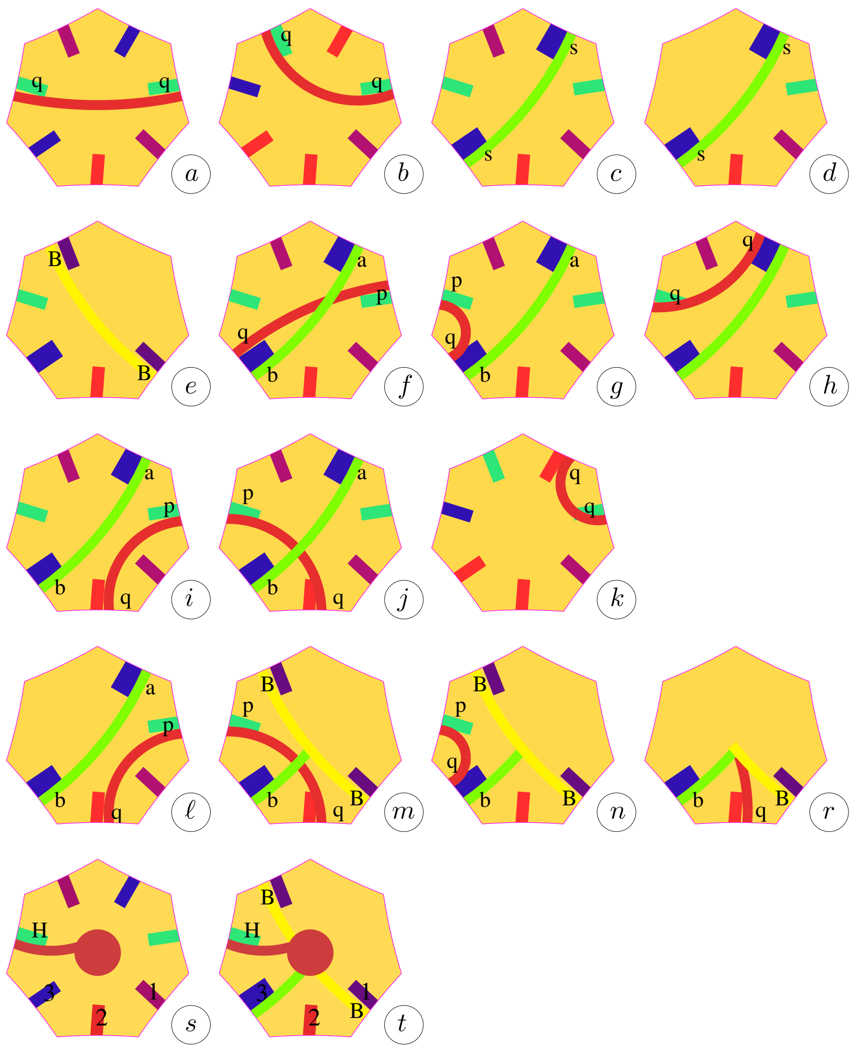


Figure 6 *The tiles for the Turing computation.*

The transportation of the signals: tiles a up to e. The rest of the tiles is devoted to the implementation of the instructions of the Turing machine.

It is not difficult to see that this gives an alternative solution to the origin-constrained problem, as indicated in [10]. Also in [10], we have indicated the tiles which allow to construct the harp and its simulation process. For the convenience of the reader, the tiles are again indicated here, by Figure 6.

4 A solution to the finite tiling problem

Now, we come to the proof of Theorem 1.

We first notice that there is just a slight modification to perform on the tiles given in [10] in order to prove the theorem.

Indeed, we give a special colour to the border, call it the **silver signal**. This colour is also generated by the halting instruction if it occurs. A silver signal spreads on both sides along the level where the halting instruction was triggered. Accordingly, it meets the silver signal of the borders. At the meeting point, the signals merge in the shape of a corner. We als call **border** this part of the level delimited by the two borders of the tree. It is clear that any tile which contains the silver signal has an inward side and an outward one. We decide that the edges which delimit the outside and which are not crossed by the silver signal are blank. Accordinly, such a side may abut with the blank tile and so, this delimits the computation and outside this area, the solution which consists in tiling the area with the blank gives a solution. As we have to prove that there is a finite solution, we need not that the blank edge must match with the blank tile only: it is enough that such a matching can be performed.

And so, we proved that if the Turing machine halts, there is a tiling realized by copies of T which contains at least the copy one tile of T and which only contains finitely many copies of tiles of T.

Now, assume that there is such a tiling τ, that τ contains finitely many tiles from T and at least one of them. Necessarily, in τ, there are tiles with a silver signal. Indeed, the tiles of T which do not belong to the border have no blank edge. This property can be checked from the tiles for the harp given in Figure 6 and the additional tiles which are given below, in Figure 7. In fact, the last two rows of tiles in Figure 7 are actually new tiles, appended to the set of tiles belonging to T. The other tiles of Figure 7 replace the corresponding tiles of Figure 6 which define the root of the tree and its both borders: in the figure, the replacing tiles have the same label as the former ones. Note that the new tiles of Figure 7 are the tile $\textcircled{u}$ which transforms the silver signal raised by the halting state into two signals for the basis of the finite figure. One signal goes to the left and the other to the right. They meet the silver signals on the border of the tree thanks to the corner tiles which are provided by the tile $\textcircled{y}$ for the left-hand side corner and the tile $\textcircled{z}$ for the right-hand side one.

Now, it is not difficult to see that as the non blank area of the tiling is finite, it must contain borders of the three possible kinds: left-hand side border, right-hand side one and the border on a level. If at least one component is missing, this means we have an infinite part with tiles of T only. Now, as there is a left-hand side border and a right-hand side one, they must meet. And so, the tiling necessarily contains an origin. Accordingly, the tiling simulates the commputation of a Turing machine. Now, as it is finite, the simulated machine halts. And so, we proved that there is a finite solution containing at least one non blank tile if and only if the simulated Turing machine halts. And so, the problem is undecidable.

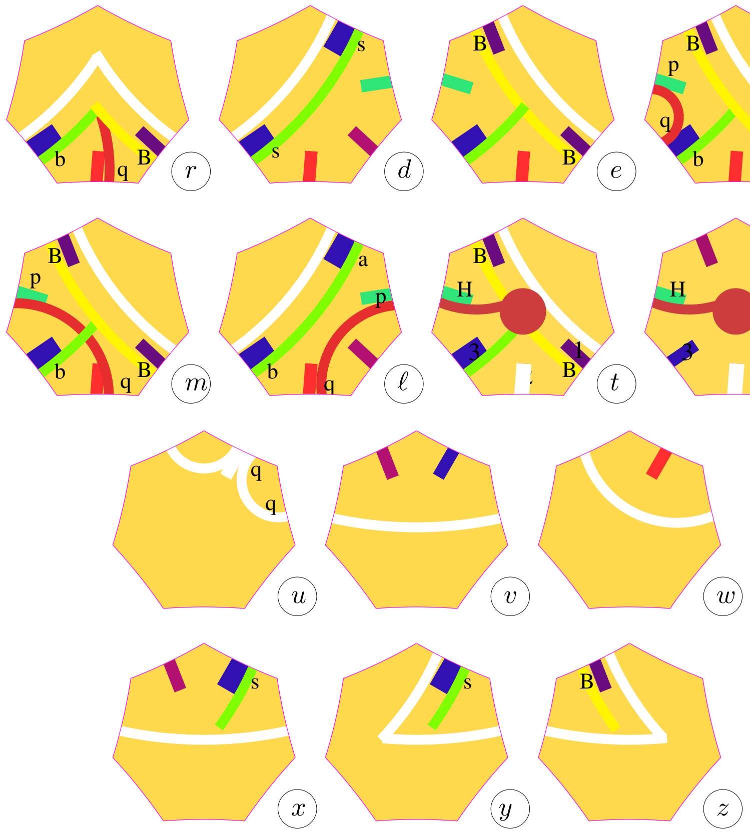


Figure 7 *The tiles for the silver signal.*

5 Conclusion

The finite tiling problem is used in [5] to prove that it is undecidable to know whether a cellular automaton in the Euclidean plane with Moore neighbourhood is surjective. At the present moment, there are two difficulties to transport this proof to the hyperbolic plane. The first one is the use of the classical characterization of surjective cellular automata with cellular automata whose global function is injective when restricted to finite configurations, see [16, 17]. Although Hedlund's theorem can be transported to the hyperbolic plane at the price of an additional condition, see [13], the arguments of Moore's and Myhill's proofs cannot be transported to the hyperbolic plane. The second obstruction lies in the plane-filling property: it is still open whether this holds or not for the hyperbolic plane.

Accordingly, there is still much work to do in these directions.

Another point is the following. In the Euclidean plane, it is not difficult to see that, from the proof of the finite tiling problem, we can prove that periodic tiling problem is also undecidable. Recall that this latter problem consists in asking whether, from a given finite set of prototiles, it is possible or not to tile the plane in a periodic way. This problem was proved undecidable by Yu. Gurevich and I. Koriakov, see, [3]. The present construction cannot be used to prove the undecidability of the periodic tiling problem in the hyperbolic plane. However, the construction of [11, 14, 12], combined with the construction of the present proof, gives a solution which is still not immediate. This is done in [15], where we also discuss the adaptation of the notion of periodicity in the hyperbolic plane, a question which is not that straightforward.

References

1. Berger R., The undecidability of the domino problem, *Memoirs of the American Mathematical Society*, **66**, (1966), 1-72.
2. Chelghoum K., Margenstern M., Martin B., Pecci I., Cellular automata in the hyperbolic plane: proposal for a new environment, *Lecture Notes in Computer Sciences*, (2004), **3305**, 678-687, (proceedings of ACRI'2004, Amsterdam, October, 25-27, 2004).
3. Gurevich Yu., Koriakov I., A remark on Berger's paper on the domino problem, *Siberian Mathematical Journal*, **13**, (1972), 459–463.
4. Hopcroft, J.E., Motwani, R., and Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, Boston/San Francisco/New York, (2001).
5. Kari J., Reversibility and surjectivity problems of cellular automata, *Journal of Computer and System Sciences*, **48**, (1994), 149-182.
6. Kari J., A new undecidability proof of the tiling problem: The tiling problem is undecidable both in the Euclidean and in the hyperbolic plane, *AMS meeting a NC, Special Session on Computational and Combinatorial Aspects of Tilings and Substitutions*, March, 3-4, (2007).
7. Margenstern M., New Tools for Cellular Automata of the Hyperbolic Plane, *Journal of Universal Computer Science* **6**N°12, 1226–1252, (2000)
8. Margenstern M., Cellular Automata in Hyperbolic Spaces, Volume 1, Theory, *OCP*, Philadelphia, to appear (2007).
9. Margenstern M., About the domino problem in the hyperbolic plane from an algorithmic point of view, *arxiv:cs.CG/*060393, (2006), March, 11pp.
10. Margenstern M., About the domino problem in the hyperbolic plane from an algorithmic point of view, *Publications du LITA, N°*2006-101*, Université de Metz*, (2006), 110pp.
11. Margenstern M., About the domino problem in the hyperbolic plane, a new solution, *arxiv:cs.CG/*070196*v*1*,arxiv:cs.CG/*070196*v*2, (2007), January, 60pp.
12. Margenstern M., About the domino problem in the hyperbolic plane, a new solution, *Publications du LITA, N°*2007-102*, Université de Metz*, (2007), 107pp.
13. Margenstern M., On a characterization of cellular automata in tilings of the hyperbolic plane, *arxiv:cs.DM/*0702155*v*1, (2007), February.

14. Margenstern M., A tiling of the hyperbolic plane: the mantilla, with an application to the tiling problem, *AMS meeting at Davidson, NC, Special Session on Computational and Combinatorial Aspects of Tilings and Substitutions*, March, 3-4, (2007).
15. Margenstern M., The periodic domino problem is undecidable in the hyperbolic plane, *arxiv:cs.CG/*0703153*v*1, (2007), March, 12p.
16. Moore E.F., Machine Models of Self-reproduction, *Proceedings of the Symposium in Applied Mathematics*, **14**, (1962), 17-33.
17. Myhill J., The Converse to Moore's Garden-of-Eden Theorem, *Proceedings of the American Mathematical Society*, **14**, (1963), 685-686.
18. Robinson R.M. Undecidability and nonperiodicity for tilings of the plane, *Inventiones Mathematicae*, **12**, (1971), 177-209.
19. Robinson R.M. Undecidable tiling problems in the hyperbolic plane. *Inventiones Mathematicae*, **44**, (1978), 259-264.
20. Turing A.M., On computable real numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, ser. 2, **42**, 230-265, (1936).
21. Wang H. Proving theorems by pattern recognition, Bell System Tech. J. vol. **40** (1961), 1–41.

An Alternative Construction in Symbolic Reachability Analysis of Second Order Pushdown Systems

Anil Seth

CSE Department, I.I.T. Kanpur, Kanpur 208016, INDIA.
seth@cse.iitk.ac.in

Abstract. Recently, it has been shown that for any higher order pushdown system $\mathcal{H}$ and for any regular set $\mathcal{C}$ of configurations, the set $pre^*_{\mathcal{H}}(\mathcal{C})$, is regular. In this paper, we give an alternative proof of this result for second order automata. Our construction of automata for recognizing $pre^*_{\mathcal{H}}(\mathcal{C})$ is explicit. The termination of saturation procedure used is obvious. It gives a better bound on size of the automata recognizing $pre^*_{\mathcal{H}}(\mathcal{C})$ if there is no alternation present in $\mathcal{H}$ and in the automata recognizing $\mathcal{C}$. Using our techniques for two players reachability games on second order pushdown systems, we generalize some result of [2] concerning synthesis of strategies. Analogous to [2], we show two kinds of winning strategies for player 0 and give algorithms to compute them. The first is executable by second order pushdown automata and the second is linear time executable minimum cost strategy.

1 Introduction

Higher order pushdown automata (*hpda*) are a generalization of pushdown automata in that they can have nested stacks, such as stack of stacks. Higher order Push and Pop operations are provided to push a copy of the topmost stack of any order and to pop it. Order of a *hpda* depends on the depth of nested stacks allowed by a *hpda* .

These models were introduced in [7] and were further studied in [9, 8]. Higher order pushdown systems (*hpds*) are *hpda* without any input. A configuration of a *hpds* is a pair of its control state and its stack contents. A *hpds* can be viewed as a transition system on the infinite set of configurations. In the last few years, algorithmic properties of finitely presented infinite graphs have been extensively investigated by verification community, see [5, 11, 6]. Here we survey a few results which are of direct relevance to our work.

In [4], it was shown that for any pushdown system (order-1 *hpds*), and any regular set C of its configurations, $pre^*(C)$, the set of configurations from which an element of C is reachable, is also regular. [4] gave an iterative procedure, called saturation procedure, to construct an automata recognizing $pre^*(C)$, starting from the automata recognizing C. The iteration step adds new edges to the automata but no new states.

In [3], Bouajjani and Meyer, extended this result to higher order context free processes, which are higher order pushdown systems with a single control state. They have introduced a structured way to represent a regular set of configurations of a *hpds* . For example a regular set of order-2 PDS configurations is given by a automata with finitely many states but whose edge labels are ordinary finite automata recognising order-1 stack configurations. Authors of [3] have generalized the saturation procedure to these automata to prove their results. Recently, Hague and Ong [1] have extended results of [3] to higher order pushdown systems. This is a technically non-trivial result. Let A be a order-2 finite automata recognizing a set of order-2 PDS configurations C. The saturation procedure of [1] uses nested recursion involving automata appearing as edge labels of A and the structure of A. As new states are added during this construction termination of the recursion is not immediate and requires a proof. [1] gives details of all this.

In this paper, we give an alternative construction of an automata recognizing $pre^*(C)$ for second order *hpds* . The state set of our automata to compute $pre^*(C)$ is fixed beforehand and remains the same throughout saturation procedure, so termination of saturation procedure is immediate. Further each state has some intuitive meaning and saturation procedure is reduced essentially to order-1 saturation procedure. Our result was arrived at independently of [1].

Unlike, [3, 1], we represent a regular set of order-2 *hpda* configurations by ordinary automata. This is no restriction because as pointed out in [3] the two notions of regularity for representing stack configurations are in fact the same. In fact, given a (deterministic/nondeterministic) finite automata as in [3], one can convert it into a (deterministic/nondeterministic) ordinary automata in polynomial time.

In [1], size of the automata recognising $pre^*(C)$ (and the time to construct it) is double exponential in the size of *hpds* $\mathcal{H}$ (of order-2) and the size of automata $\mathcal{A}$, recognising C. The authors in [1], in fact solve the more general case of 2-player (or alternating) *hpds* with the same bounds. Further, in [1], $\mathcal{A}$, recognising C is also assumed to be alternating. Our bounds match theirs in the general case. We consider the special case when $\mathcal{H}$ is nondeterministic and $\mathcal{A}$ in given as a non-deterministic automata, in this case we show that an automata recognising $pre^*(C)$ can be constructed in time single exponential in the size of *hpds* $\mathcal{H}$ and size of automata $\mathcal{A}$. In case of 2-player (or alternating) *hpds* and nondeterministic $\mathcal{A}$ our construction takes double exponential time which matches the bound in [1], in fact in this case we construct an alternating automata with single exponential states to recognise attractor set of C.

In [2], results of [4] are used to give uniform winning strategies in two player reachability games on order-1 *hpds* . Two kinds of strategies are given in [2], the first is minimum cost positional strategy executable in linear time and the second is computable by a order-1 *hpds* . We generalize these results to order-2 *hpds* reachability games. We show a minimum cost positional strategy executable in linear time and a strategy executable by second order pushdown automata in order-2 *hpds* reachability games.

2 Preliminaries

A higher order pushdown system (*hpds*) $\mathcal{H}$ is a triple (Q, Γ, Δ), where Q is a finite set of control states, Γ is a (finite) stack alphabet and $\Delta \subseteq Q \times \Gamma \times Act$ is transition relation of $\mathcal{H}$. For an order-2 *hpds* , the set of actions is defined as $Act = \{push_1^{q,w}, push_2^q, pop_i^q \mid i \in \{1,2\}, q \in Q, w \in \Gamma^*\}$.

Configurations of a *hpds* are pairs (q, s), where q is a control state of the *hpds* and s is its stack configuration. The set of stack configurations of the ordinary pushdown automata (order-1 *hpds*) denoted by S_1 and the set of stack configurations of an order-2 *hpds* denoted by S_2 are defined as follows.

$S_1 = \{ \ [_1 \ w \ ]_1 \mid w \in \Gamma^*\} \ , \quad S_2 = \{ \ [_2 \ \alpha \ ]_2 \mid \alpha \in (S_1)^*\}$

Throughout this paper we use symbol $[_i$ to denote start of stack of order i and $]_i$ for the end of stack of order i. Let $push_1^w, push_2, pop_1, pop_2$ stand for stateless counterparts of actions $push_1^{q,w}, push_2^q, pop_1^q, pop_2^q$ respectively. These stateless counterparts act on stack configurations as follows.

$push_1^w([_1 \ a_1 \cdots a_l]_1) = [_1 \ wa_2 \cdots a_l]_1,$

$push_1^w([_2 \ s_1 \cdots s_k \ ]_2) = [_2 \ push_1^w(s_1)s_2 \cdots s_k \ ]_2$

$push_2([_2 \ s_1 \cdots s_k]_2) = [_2 \ s_1 s_1 s_2 \cdots s_k]_2$

$pop_1([_2 \ s_1 \cdots s_k]_2) = [_2 \ pop_1(s_1) \cdots s_k]_2,$

$pop_m([_m \ s_1 \cdots s_k]_m) = [_m \ s_2 \cdots s_k]_m$, if $k \geq 1$ else undefined, $m \in \{1, 2\}$.

We also define topmost element, $top(s)$, of a stack configuration s, as below.

$top([_1 \ a_1 \cdots a_l]_1) = a_1 \quad , \quad top([_2 \ s_1 \cdots s_k]_2) = top(s_1)$

Transition relation $\hookrightarrow_{\mathcal{H}}$ is defined on configurations of a *hpds* of order 2 as follows.

$(q', s) \hookrightarrow_{\mathcal{H}} (q, Push_1^w(s))$ if $(q', top(s), Push_1^{q,w}) \in \Delta$

$(q', s) \hookrightarrow_{\mathcal{H}} (q, Push_2(s))$ if $(q', top(s), Push_2^q) \in \Delta$,

$(q', s) \hookrightarrow_{\mathcal{H}} (q, Pop_i(s))$ if $(q', top(s), Pop_i^q) \in \Delta$, $i \in \{1, 2\}$.

Let S be a set of stack configurations of *hpds* $\mathcal{H}$, $pre^*_{\mathcal{H}}(S)$ is defined as $pre^*_{\mathcal{H}}(S) = \{s \mid \exists s' \in S[s \hookrightarrow_{\mathcal{H}} s']\}$. We omit the subscript $\mathcal{H}$ from $\hookrightarrow_{\mathcal{H}}$ whenever it is clear from the context. $\hookrightarrow^*$ stands for the reflexive, transitive closure of $\hookrightarrow$.

We consider regular subsets of *hpds* configurations. Let $\mathcal{H} = (Q, \Gamma, \Delta)$ be a *hpds* of order-2. We define $Q' = \{q' \mid q \in Q\}$. A finite (multi)automata for recognising configurations of $\mathcal{H}$ is given as $\mathcal{A} = (S_{\mathcal{A}}, \Gamma_2, Q', \delta, F)$, where $S_{\mathcal{A}}$ is the set of states of $\mathcal{A}$, $\Gamma_2 = \Gamma \cup \{ \ [_i,]_i \mid 1 \leq i \leq 2\}$ is the input alphabet for $\mathcal{A}$. δ is the transition relation of $\mathcal{A}$, Q' is the set of its initial states and F is the set of final states of $\mathcal{A}$. Multi-automata were first introduced in [4] for order-1 PDS.

For any automata $\mathcal{B}$ and a state q of it, we use the notation $\mathcal{L}(\mathcal{B}, q)$ to denote the set of strings accepted by $\mathcal{B}$ when started in state q. The set of configurations, $C_{\mathcal{A}}$, accepted by multi-automata $\mathcal{A}$ above is given as
$C_{\mathcal{A}} = \mathcal{L}(\mathcal{A}) = \{(q, s) \mid s \in \mathcal{L}(\mathcal{A}, q')\}$.

2.1 Alternating Automata

An alternating automata R is given as (P, Σ, δ, F), where P is a finite set of states Σ is input alphabet and F is the set of final states. Transition function δ of R is given as $\delta : P \times \Sigma \rightarrow B^+(P)$, where $B^+(P)$ is the set of positive boolean formulae (constructed using connectives $\wedge$ and $\vee$ only) over atoms in P. As any function in $B^+(P)$, can be written as a disjunction of conjunctions of elements in P, transition function can also be thought of as a relation on $P \times \Sigma \times 2^P$. In other words, it can be seen as a union of transitions of the form (q, a, S), where $q \in P$, $a \in \Sigma$ and $S \subseteq P$.

A run of an alternating automata on a input is a tree. More efficiently we can represent it as a dag as in [2, 10]. Since we allow ϵ transitions (and later transitions on a string of letters instead of a single letter), all paths through this dag may not be of equal length. We give a slightly modified definition of a run dag useful for our purposes.

A run-dag of automata $\mathcal{B}$ on input w is a rooted dag in which each node is labeled with a state of $\mathcal{B}$ and each edge is labeled with an input symbol or input string including empty string ϵ. If a node is labeled with q then all outgoing edges from q in the dag are labeled by the same string u. Further, if the set of labels of nodes to which these edges are connected is $\{q_{i_1}, \dots, q_{i_k}\}$ then there must be a one step transition $(q, u, \{q_{i_1}, \dots, q_{i_k}\}) \in \delta$ The concatenation of all labels in any maximal path (path which is not contained in a bigger path) through this dag should be a prefix of the string w, such that if it is a proper prefix of w then its terminal node is labeled with a constant *true* or *false*.

A run dag is *accepting* if all its terminal nodes are labeled with 'true' or with final states of $\mathcal{B}$.

We extend the function/relation δ to take its second argument as a string instead of a single letter as follows. $(p, w, S) \in \delta$ if there is a run dag on input w with root labeled as p and its leaves are labeled with either 'true' or with an element in S.

3 Computing pre^* for Hpds of order-2

Let $\mathcal{H} = (Q, \Gamma, \Delta)$ be an order-2 *hpds* . We define $Q' = \{q' \mid q \in Q\}$.
Let $\mathcal{A} = (S_\mathcal{A}, \Gamma_2, Q', \delta_\mathcal{A}, \mathcal{F})$ be a deterministic finite multi-automata with Q' as initial states. Let $\mathcal{A}$ accept a set $\mathcal{C}_\mathcal{A}$ of configurations of $\mathcal{H}$. It is assumed w.l.o.g. that sets Q and $S_\mathcal{A}$ are mutually disjoint.

We design an alternating finite automata $\mathcal{R}_{\mathcal{H},\mathcal{A}}$ to accept $pre^*_\mathcal{H}(\mathcal{C}_\mathcal{A})$.

$\mathcal{R}_{\mathcal{H},\mathcal{A}} = (S_\mathcal{R}, \Gamma_2, \delta_\mathcal{R}, \mathcal{F} \times \{\downarrow\})$

State set of $R_{\mathcal{H},\mathcal{A}}$, denoted $S_\mathcal{R}$, is a disjoint union of several sets as below.

Let $Q'' = \{q'' \mid q \in Q\}$, $\quad U_{Q,\mathcal{A},\downarrow} = Q \cup S_\mathcal{A} \cup \{\downarrow\}$, $\quad U_{\mathcal{A},\downarrow} = S_\mathcal{A} \cup \{\downarrow\}$

[To remember notation U can be thought of as union]

$R_1 = (Q \times U_{Q,\mathcal{A},\downarrow}) \cup (S_\mathcal{A} \times U_{\mathcal{A},\downarrow})$

$R_2 = \{s_w \mid s \in R_1, (w \in \Gamma) \text{ or } (Push_1^{q,w} \in \Delta, \text{ for some } q \in Q)\}$

$S_\mathcal{R} = Q'' \cup R_1 \cup R_2 \cup (Q \times \{2\})$

It will be shown that $w \in \mathcal{L}(\mathcal{R}_{\mathcal{H},\mathcal{A}}, q_i'')$ iff $(q_i'', w) \in pre^*_\mathcal{H}(\mathcal{C}_\mathcal{A})$. We use below $\mathcal{R}$ instead of $\mathcal{R}_{\mathcal{H},\mathcal{A}}$ when no confusion occurs.

We define the transition function $\delta_\mathcal{R}$ as $\bigcup_{i \geq 0} \delta_i$, where δ_i, $i = 0, 1, 2, \dots$ are defined below iteratively. The sequence δ_i, $i = 0, 1, 2, \dots$ is monotone when viewed as a relation on $S_{\mathcal{R}} \times \Gamma_2 \times B^{+}(S_{\mathcal{R}})$. This part of the construction is called saturation procedure.

3.1 Intuitive Idea

The idea behind the construction is to start with the transition group (0). The automata now looks at the top of the stack symbol and sees which moves in *hpds* are possible. The $push_1$ and pop_1 move are handled as in order-1 case. Pop_2 move is simply handled by ignoring the input till the end of current order-1 stack. The $push_2$ operation is to be handled differently as it makes two copies of the current order-1 stack while there is only one copy in the input. To do so, we simultaneously verify the operation on the top stack of the configuration and the stack below. For this we need to know in which state is the top stack popped? This is done by guessing this state. So at a $push_2$ move the computation splits into two threads one thread verifies that the current stack can be popped in some state q and the other thread starts running on the current input with this state. The verifying thread will die on meeting the end of current order-1 stack either successfully or unsuccessfully.

The computation in the two threads is similar in many cases so we use unifying notation for states involving pairs. States $(p, \downarrow)$, through the use of $\downarrow$ indicates main thread which is to continue beyond the current order-1 stack. Other pairs denote threads which check constraints. Note both these type of threads can spawn further threads.

In this way we will be able to keep information about *hpds* configurations that can be reached. At each step, the automata has a choice to explore the next configuration reached or check if the current configuration is in the target set, whose pre^* is being computed. These are the main ideas, the construction below shows how these can be made to work.

3.2 Transitions in δ_0

The transitions of $\mathcal{R}$ below are grouped according to transitions in *hpds* $\mathcal{H}$. To improve readability of triples below, we show $S_{\mathcal{R}} \times \Gamma_2$ part of the triple in lightface and $B^{+}(S_{\mathcal{R}})$ component in boldface.

(0). For $q \in Q$, $(q'', [_2[_1, (q, \downarrow)) \in \delta_0$.

(i). $(\mathbf{q}, \mathbf{a}, \mathbf{pop_1^p}) \in \boldsymbol{\Delta}$ then for $t \in Q \cup S_{\mathcal{A}} \cup \{\downarrow\}$, $((q, t), a, (\mathbf{p}, \mathbf{t})) \in \delta_0$.

(ii). $(\mathbf{q}, \mathbf{a}, \mathbf{push_2^p}) \in \boldsymbol{\Delta}$

(a)

$$\text{then for each } \ t \in Q, \quad ((q, t), a, \bigvee_{\mathbf{q_1} \in \mathbf{Q}} [(\mathbf{p}, \mathbf{q_1})_{\mathbf{a}} \wedge (\mathbf{q_1}, \mathbf{t})_{\mathbf{a}}]) \in \delta_0,$$

(b)

$$\text{and for each } t \in U_{\mathcal{A},\downarrow}, \quad ((q, t), a, \bigvee_{\mathbf{t_1} \in \mathbf{Q} \cup \mathbf{S}_{\mathcal{A}}} [(\mathbf{p}, \mathbf{t_1})_{\mathbf{a}} \wedge (\mathbf{t_1}, \mathbf{t})_{\mathbf{a}}]) \in \delta_0$$

/* See Lemma 1(5-7), for the intuitive meaning of states $(p, q_l)_a$ etc. */

(iii). $(\mathbf{q}, \mathbf{a}, \mathbf{pop_2^p}) \in \mathbf{\Delta}$ then we have the following triples in δ_0

(a) $((q, \downarrow), a, (\mathbf{p}, \mathbf{2}))$,

(b) $((r, 2), b, (\mathbf{r}, \mathbf{2}))$, $r \in Q, b \in \Gamma$ /* skip the current order-1 stack */

(c) $((r, 2),]_1[_1, (\mathbf{r}, \downarrow))$, /* beginning of the next order-1 stack reached */

(d) $((r, 2),]_1]_2, (\delta_{\mathcal{A}}(\mathbf{r'}, [_\mathbf{2}]_\mathbf{2}), \downarrow))$, /* this corresponds to the case when the popped stack was the last one */

(e) $((q, p), a, \mathbf{true})$ /* popping the stack in state q on 'a' satisfies the constraint (q, p) */

(iv). $(\mathbf{q}, \mathbf{a}, \mathbf{push_1^{p,u}}) \in \mathbf{\Delta}$ then for $t \in U_{Q,\mathcal{A},\downarrow}$, $((q, t), a, (\mathbf{p}, \mathbf{t})_\mathbf{u}) \in \delta_0$

(v). Transitions related to $\mathcal{A}$

/* We have the following transitions for simulating automata $\mathcal{A}$ on a guessed *hpds* configuration. */

(a) $(q'', [_2, (\delta_{\mathcal{A}}(\mathbf{q'}, [_\mathbf{2}), \downarrow))$, for $q \in Q$.
/* The guessed configuration is the initial configuration */

(b) $((q, t), \epsilon, (\delta_{\mathcal{A}}(\mathbf{q'}, [_\mathbf{2}[_\mathbf{1}), \mathbf{t}))$, for all $q \in Q, t \in U_{\mathcal{A},\downarrow}$.
/* The automata $\mathcal{R}$ guesses a configuration */

(c) $((t_1, t_2), a, (\delta_{\mathcal{A}}(\mathbf{t_1}, \mathbf{a}), \mathbf{t_2}))$, $\quad for\ (t_1 \in S_{\mathcal{A}}, t_2 = \downarrow, a \in \Gamma \cup \{[_1,]_1\})$
$\quad OR \quad for\ (t_1 \in S_{\mathcal{A}}, t_2 \in S_{\mathcal{A}}, a \in \Gamma)$.
/* Simulates $\mathcal{A}$ in the first component */

(d) For $t_2 \in S_{\mathcal{A}}$,

$$((t_1, t_2),]_1, true) \in \delta_0 \quad \text{if } \delta_{\mathcal{A}}(t_1,]_1) = t_2$$
$$((t_1, t_2),]_1, false) \in \delta_0 \quad \text{otherwise}$$

/* accepts if the constraint is satisfied at the end of current order-1 stack */

(vi). Transitions from states in R_2

$((t_1, t_2)_a, \epsilon, (\delta_{\mathcal{A}}(\mathbf{t_1}, [_\mathbf{1}\mathbf{a}), \mathbf{t_2}))$, for all $t_1 \in S_{\mathcal{A}}, t_2 \in U_{\mathcal{A},\downarrow}, a \in \Gamma$.

3.3 Saturation Step

Saturation process is as follows.

$\delta_{k+1} := \delta_k \cup V$, where

$V = \{((p, t)_x, \epsilon, S) \mid p \in Q, t \in U_{Q,\mathcal{A},\downarrow}, ((p, t), x, S) \in \delta_k, (p, t)_x \in R_2\}$

For each k, $\delta_k \subseteq S_{\mathcal{R}} \times \Gamma_2 \times 2^{S_{\mathcal{R}}}$. As $(\delta_k)_{k \geq 0}$ is a monotonically increasing sequence, the saturation procedure terminates in at most $|S_{\mathcal{R}}| \times |\Gamma_2| \times 2^{|S_{\mathcal{R}}|}$ iterations.

3.4 Correctness of the Construction

The following lemma easily follows from the construction.

Lemma 1. *1. For $t \in S_{\mathcal{A}}$, $w \in \mathcal{L}(\mathcal{R},(t, \downarrow))$ iff $w \in \mathcal{L}(\mathcal{A}, t)$.*

2. For $t_1, t_2 \in S_{\mathcal{A}}$, $w \in \mathcal{L}(\mathcal{R}, (t_1, t_2))$ iff $w = x]_1v$, $x \in \Gamma^*$ and $\delta_{\mathcal{A}}(t_1, x]_1) = t_2$.

3. *$w \in \mathcal{L}(\mathcal{R},(q,2))$ iff $w = x]_1[_1u$ for $x \in \Gamma^*$ and $u \in \mathcal{L}(\mathcal{R},(q,\downarrow))$ or $w = x]_1]_2$ for $x \in \Gamma^*$ and $[_2]_2 \in \mathcal{L}(\mathcal{A},q')$.*
4. *For $q \in Q$, $w \in \mathcal{L}(\mathcal{R},q'')$ iff $w = [_2[_1v$ for some v, and $v \in \mathcal{L}(\mathcal{R},(q,\downarrow))$ or $w \in \mathcal{L}(\mathcal{A},q')$.*
5. *For $r \in Q \times (Q \cup S_\mathcal{A} \cup \{\downarrow\})$ and $r_u \in R_2$, $w \in \mathcal{L}(\mathcal{R},r_u)$ iff $uw \in \mathcal{L}(\mathcal{R},r)$.*
6. *For $t \in S_\mathcal{A}$, $a \in \Gamma$, $w \in \mathcal{L}(\mathcal{R},(t,\downarrow)_a)$ iff $[_1aw \in \mathcal{L}(\mathcal{A},t)$.*
7. *For $(t_1,t_2) \in S_\mathcal{A} \times S_\mathcal{A}$, $a \in \Gamma$, $w \in \mathcal{L}(\mathcal{R},(t_1,t_2)_a)$ iff $w = x]_1v$, $x \in \Gamma^*$ and $\delta_\mathcal{A}(t_1,[_1ax]_1) = t_2$.*

Proof. We omit easy details from this extended abstract. □

The theorem below is at the heart of the correctness proof.

Theorem 1. *Let $\hookrightarrow_*$ be the reachability relation on configurations of hpds $\mathcal{H}$. The following assertions hold for all $p,q \in Q$, $t \in S_\mathcal{A}$, $v,w \in \Gamma_2^*$.*

1. *For $q \in Q$, $w \in \mathcal{L}(\mathcal{R},(q,\downarrow))$ iff there is a $\tau \in \mathcal{L}(\mathcal{A})$ such that $(q,[_2[_1w) \hookrightarrow^* \tau$.*
2. *For $p,q \in Q$, $w \in \mathcal{L}(\mathcal{R},(p,q))$ iff $w = x]_1u$, for some $x \in \Gamma^*$, and $(p,[_2[_1x]_1]_2) \hookrightarrow^* (q,[_2]_2)$.*
3. *For $p \in Q$ and $t \in S_\mathcal{A}$, $w \in \mathcal{L}(\mathcal{R},(p,t))$ iff $w = x]_1u$, for some $x \in \Gamma^*$, and $(p,[_2[_1x]_1]_2) \hookrightarrow^* (r,[_2[_1v]_1]_2)$ and $\delta_\mathcal{A}(r,[_2[_1v]_1) = t$.*

Proof. This is proved in two parts. We omit the details here which may be found in full version. □

Combining Lemma 1 part (4) and Theorem 1 part (1), we immediately have the following corollary.

Corollary 1. *For $q \in Q$, $w \in \mathcal{L}(\mathcal{R},q'')$ iff $w \in pre^*(C_\mathcal{A})$.*

We have presented the construction for deterministic $\mathcal{A}$, however essentially the same construction works for non-deterministic $\mathcal{A}$ also. The only change is that we replace $\delta_\mathcal{A}(r,a)$ by $\vee$ of states instead of a single state. More precisely, transitions (iii.d), (v) and (vi) only need to be modified. We omit easy details.

Note that the situation changes if $\mathcal{A}$ is alternating. In that case, the form of pairs (q,t) is to be replaced by (q,T), where $T \subseteq S_\mathcal{A}$. This is because a path in the simulation of $\mathcal{A}$ is given by a set of states. Consequently the number of states in the automata becomes exponential in $S_\mathcal{A}$.

Corollary 2. *Let $\mathcal{H}$ be a nondeterministic hpds with $|Q|$ states and $|\Delta|$ transitions and let $\mathcal{A}$ be a nondeterministic automaton with $|S_\mathcal{A}|$ states. Then one can effectively construct a finite alternating automata with $O((|Q|+|S_\mathcal{A}|)^2 \cdot |\Delta|)$ states to recognize $pre^*(C_\mathcal{A})$.*

From the above alternating automata recognizing $pre^*(C_\mathcal{A})$, we can get by standard construction a equivalent nondeterministic automata with 2^z states, where z is a polynomial in the size of $\mathcal{H}$ and $\mathcal{A}$. This is one exponential less than what the construction of [1] gives though the construction of [1] is optimal if $\mathcal{H}$ and $\mathcal{A}$ are alternating.

4 Two Player Reachability Game over Hpds

In rest of the paper, we extend our techniques to study two player reachability games over configuration graphs of second order *hpds* . Two player games in general are an important model of reactive computation and have been widely studied in the context of verification and synthesis of finite/infinite state systems over the last few years, see [12, 13].

We first recall some preliminary notions about these games. A game structure can be imposed on the configuration graph of a *hpds* by partitioning the states of the *hpds* into two parts. $\mathcal{H} = (Q_0 \oplus Q_1, \varGamma, \varDelta)$ is a game structure where states in Q_m, $m \in \{0, 1\}$, correspond to player m.

A position in such a game is a configuration of $\mathcal{H}$. A position belongs to player m if control state in this position (configuration) $\in Q_m$. Starting from a initial position π_0 a play proceeds as follows, if π_0 belongs to player-m then player-m chooses a position π_1 such that $\pi_0 \hookrightarrow_{\mathcal{H}} \pi_1$. The play now enters position π_1 and continues this way, at stage i if position π_i belongs to player-m, $m \in \{0, 1\}$, then player-m chooses a position π_{i+1} such that $\pi_i \hookrightarrow_{\mathcal{H}} \pi_{i+1}$. A play is a sequence (possibly infinite) $\pi_0\pi_1 \cdots \pi_i \cdots$ of successive game positions.

In a reachability game for player k, $k \in \{0, 1\}$, a set of game positions (configurations of *hpds*) E is also given. Player-k wins a play in this game if a position $\in E$ is reached during this play otherwise player-$(1 - k)$ wins.

A strategy for player m, $m \in \{0, 1\}$, from position π_0, is a function which associates to each prefix $\pi_0\pi_1 \cdots \pi_i$ of a play, where π_i belongs to player m, a position π_{i+1} such that $\pi_i \hookrightarrow_{\mathcal{H}} \pi_{i+1}$. A strategy σ for player-m from position π is a winning strategy if in any play begining with π where player-m plays according to σ, player-m wins the play. A position π is winning for player-m if there is a winning strategy for player-m from position π. The set of all winning positions of player-m is called winning region or player-m.

It is well known that the winning region of player-k in the reachability game for player-k is is given by attractor of E with respect to k, denoted $Attr_k(E)$ and defined as below.

$$\begin{aligned}
Attr_k^0(E) &= E \\
Attr_k^{j+1}(E) &= \{(q, s) \mid q \in Q_k, \exists \tau \in Attr_k^j(E)[(q, s) \hookrightarrow \tau]\} \\
&\quad \cup \{(q, s) \mid q \in Q_{1-k}, \forall \tau[((q, s) \hookrightarrow \tau) \rightarrow \tau \in Attr_k^j(E)]\} \\
Attr_k(E) &= \textstyle\bigwedge_{j \geq 0} Attr_k^j(E)
\end{aligned}$$

Note that the reachability problem of the previous section can be considered as a special case where all states belong to one player.

5 Regularity of the Attractor Set

Let reachability game for player k be given by game structure
$H = (Q_0 \oplus Q_1, \varGamma, \varDelta)$ and a regular set $\mathcal{L}(\mathcal{A})$ specified by automata $\mathcal{A}$. In this section we show that winning region of player-k in the above game is regular.

We can modify the automata $\mathcal{R}$ of section 3 to $\mathcal{R}^k$ to compute attractor set with respect to player k. We need to replace the states such as (p, t) in section 3 by (p,T), $T \subseteq Q \cup S_{\mathcal{A}}$, as player k may not be able to guarantee to bring the game in specific configuration but only in a set of configurations.

The state set of $\mathcal{R}^k$, $St(R^k)$, is defined as follows.

Let $\mathcal{P}(Z)$ denote the powerset of Z, let $U_{\mathcal{P}(Q,\mathcal{A})} = \mathcal{P}(Q \cup S_{\mathcal{A}})$ and let $U_{\mathcal{P}(Q,\mathcal{A}),\downarrow} = U_{\mathcal{P}(Q,\mathcal{A})} \cup \{\downarrow\}$.

Let $R_1^{(k)} = (Q \cup S_{\mathcal{A}}) \times U_{\mathcal{P}(Q,\mathcal{A}),\downarrow}$

Let $R_2^{(k)} = \{s_w \mid s \in R_1^{(k)}, (w \in \Gamma) \text{ or } (Push_1^{q,w} \in \Delta, \text{ for some } q \in Q)\}$

$St(R^{(k)}) = Q'' \cup R_1^{(k)} \cup R_2^{(k)} \cup (Q \times \{2\})$

Intuitive meaning of state (p,T) is that $\mathcal{R}^k$ accepts a configuration C from (p,T) iff either player k can guarantee that current order-1 stack can be popped in a state $\in T$ (in fact, $T \cap Q$) or it guarantees that the game from configuration C can reach a configuration C' without popping topmost order-1 stack of C and in a run of $\mathcal{A}$ on input C', $\mathcal{A}$ reaches a state $\in T$ (in fact, $T \cap S_{\mathcal{A}}$) at the end of reading topmost order-1 stack of C. This is proved in Theorem 2(ii).

Transition function $\delta_{\mathcal{R}^k}$ is defined using the saturation procedure as in section 3.

5.1 Transitions in δ_0

(0). For $q \in Q$, $(q'', [_2[_1, (q,\downarrow)) \in \delta_0$.

(i). $q \in Q_k$, $(\mathbf{q}, \mathbf{a}, \mathbf{pop_1^p}) \in \mathbf{\Delta}$ then for all $T \in U_{\mathcal{P}(Q,\mathcal{A}),\downarrow}$, $((q,T), a, (\mathbf{p}, \mathbf{T}))$.

(ii). $q \in Q_k$, $(\mathbf{q}, \mathbf{a}, \mathbf{push_2^p}) \in \mathbf{\Delta}$ then for all $T \in U_{\mathcal{P}(Q,\mathcal{A}),\downarrow}$,

$$((q,T), a, \bigvee_{\mathbf{T_1 \subseteq Q \cup S}_{\mathcal{A}}} [(\mathbf{p}, \mathbf{T_1})_{\mathbf{a}} \bigwedge \wedge_{\mathbf{t} \in \mathbf{T_1}} (\mathbf{t}, \mathbf{T})_{\mathbf{a}}])$$

(iii). $q \in Q_k$, $(\mathbf{q}, \mathbf{a}, \mathbf{pop_2^p}) \in \mathbf{\Delta}$ then we have the following triples in δ_0

(a) $((q,\downarrow), a, (\mathbf{p}, \mathbf{2}))$,
(b) $((r,2), b, (\mathbf{r}, \mathbf{2}))$, $r \in Q, b \in \Gamma$
(c) $((r,2),]_1[_1, (\mathbf{r}, \downarrow))$,
(d) $((r,2),]_1]_2, (\delta_{\mathcal{A}}(\mathbf{r'}, [_\mathbf{2}]_\mathbf{2}), \downarrow))$,
(e) $((q,T), a, \mathbf{true})$, if $p \in T$

(iv). $q \in Q_k$, and $(\mathbf{q}, \mathbf{a}, \mathbf{push_1^{p,u}}) \in \mathbf{\Delta}$ then for all $T \in U_{\mathcal{P}(Q,\mathcal{A}),\downarrow}$, $((q,T), a, (\mathbf{p}, \mathbf{T})_{\mathbf{u}})$

(v). Transitions related to $\mathcal{A}$

(a) $(q'', [_2, (\delta_{\mathcal{A}}(\mathbf{q'}, [_\mathbf{2}), \downarrow))$, for $q \in Q$.
(b) $((q,T), \epsilon, (\delta_{\mathcal{A}}(\mathbf{q'}, [_\mathbf{2}[_\mathbf{1}), \mathbf{T}))$, for all $T \in U_{\mathcal{P}(Q,\mathcal{A}),\downarrow}$.
(c) $((t_1,T), a, (\delta_{\mathcal{A}}(\mathbf{t_1}, \mathbf{a}), \mathbf{T}))$, $\quad$ *for* $(t_1 \in S_{\mathcal{A}}, T = \downarrow, a \in \Gamma \cup \{[_1,]_1\})$
$\quad$ *OR* $\quad$ *for* $(t_1 \in S_{\mathcal{A}}, T \in U_{\mathcal{P}(Q,\mathcal{A})}, a \in \Gamma)$.
(d) For $T \in U_{\mathcal{P}(Q,\mathcal{A})}$,

$$\begin{array}{ll} ((t_1,T),]_1, true) \in \delta_0 & \text{if } \delta_{\mathcal{A}}(t_1,]_1) \in T \\ ((t_1,T),]_1, false) \in \delta_0 & \text{otherwise} \end{array}$$

(vi). Transitions from states in R_2
$((t_1,T)_a, \epsilon, (\delta_{\mathcal{A}}(\mathbf{t_1}, [_\mathbf{1}\mathbf{a}), \mathbf{T}))$, for all $t_1 \in S_{\mathcal{A}}, T \in U_{\mathcal{P}(Q,\mathcal{A}),\downarrow}$.

(vii). $\mathbf{q} \in \mathbf{Q_{1-k}}$
We define for $T \in U_{\mathcal{P}(Q,\mathcal{A}),\downarrow}$,
$I_1((q,T),a) = \bigwedge\{(p,T) \mid (q,a,pop_1^p) \in \Delta\}$
$I_2((q,T),a) = \bigwedge\{(q,T)_u \mid (q,a,push_1^{p,u}) \in \Delta\}$
$I_3((q,T),a) = \bigwedge\{\bigvee_{T_1 \in U_{\mathcal{P}(Q,\mathcal{A})}}[(q,T_1)_a \wedge_{t \in T_1} (t,T)_a] \mid (q,a,push_2^p) \in \Delta\}$
$I_4((q,\downarrow),a) = \bigwedge\{(p,2) \mid (q,a,pop_2^p) \in \Delta\}$
For $T \in U_{\mathcal{P}(Q,\mathcal{A})}$,

$$I_4((q,T),a) = \begin{cases} false & \text{if } (q,a,pop_2^p) \in \Delta, \text{ for some } p \notin T \\ true & \text{otherwise} \end{cases}$$

We add the following triples to δ_0, for $T \in U_{\mathcal{P}(Q,\mathcal{A}),\downarrow}$,

$$((q,T),a,\bigwedge_{j=1}^{4} I_j((q,T),a))$$

[Intuitively, I_1, I_2, I_3 and I_4 cover the cases when player $1-k$ chooses moves $pop_1, push_1, push_2$ and pop_2 respectively. In case of pop_2, if player $1-k$ can pop the current stack to reach a state $\notin T$, then constraint of reaching a state in T at the end of current order-1 stack can not be met and $\mathcal{R}^k$ rejects from this state.]

5.2 Saturation Step

Saturation process is as follows.
$\delta_{k+1} := \delta_k \cup \{((p,T)_x, \epsilon, \delta_\mathbf{k}((\mathbf{p},\mathbf{T}),\mathbf{x})) \mid p \in Q, (p,T)_x \in R_2\}$

5.3 Correctness of the Construction

Following is an easy Lemma, analogous to Lemma 1 of section 3.

Lemma 2. *1. For $t \in S_{\mathcal{A}}$, $w \in \mathcal{L}(\mathcal{R}^k,(t,\downarrow))$ iff $w \in \mathcal{L}(\mathcal{A},t)$.*
2. For $t_1 \in S_{\mathcal{A}}$, $T \in U_{\mathcal{P}(Q,\mathcal{A})}$, $w \in \mathcal{L}(\mathcal{R}^k,(t_1,T))$ iff $w = x]_1v$, $x \in \Gamma^*$ and $\delta_{\mathcal{A}}(t_1,x]_1) \in T$.
3. $w \in \mathcal{L}(\mathcal{R}^k,(q,2))$ iff $w = x]_1[_1u$ for $x \in \Gamma^*$ and $u \in \mathcal{L}(\mathcal{R}^k,q)$ or $w = x]_1]_2$ for $x \in \Gamma^*$ and $[_2]_2 \in \mathcal{L}(\mathcal{A},q')$.
4. For $q \in Q$, $w \in \mathcal{L}(\mathcal{R}^k,q'')$ iff $w = [_2[_1v$ for some v, and $v \in \mathcal{L}(\mathcal{R}^k,(q,\downarrow))$ or $w \in \mathcal{L}(\mathcal{A},q')$.
5. For $r \in Q \times (U_{\mathcal{P}(Q,\mathcal{A}),\downarrow})$ and $r_u \in R_2^k$, $w \in \mathcal{L}(\mathcal{R}^k,r_u)$ iff $uw \in \mathcal{L}(\mathcal{R}^k,r)$.
6. For $t \in S_{\mathcal{A}}$, $a \in \Gamma$, $w \in \mathcal{L}(\mathcal{R}^k,(t,\downarrow)_a)$ iff $[_1aw \in \mathcal{L}(\mathcal{A},t)$.
7. For $(t_1,T) \in S_{\mathcal{A}} \times U_{\mathcal{P}(Q,\mathcal{A})}$, $a \in \Gamma$, $w \in \mathcal{L}(\mathcal{R}^k,(t_1,T)_a)$ iff $w = x]_1v$, $x \in \Gamma^*$ and $\delta_{\mathcal{A}}(t_1,[_1ax]_1) \in T$.

Proof. Proof is similar to that of Lemma 1, we omit it. □

To state the main theorem in correctness proof, we first introduce a notation. Let $T \subseteq Q \cup S_{\mathcal{A}}$, we define $M(T) = \{(r, [_2 s]_2) \mid s \in S_1^*, \delta_{\mathcal{A}}(r', [_2 s) \in T\}$. $M(T)$ is the set of configurations of $\mathcal{H}$, on which automata $\mathcal{A}$ has a run which reaches a state in T after seeing the last order-1 stack. Using this notation, we have the following main theorem in the correctness proof which is analogous to Theorem 1 of section 3.

Theorem 2. *The following assertions hold for all $p, q \in Q$, and $T \subseteq Q \cup S_{\mathcal{A}}$.*

1. *$w \in \mathcal{L}(\mathcal{R}^k, (q, \downarrow))$ iff $(q, [_2[_1 w) \in Attr_k(\mathcal{L}(\mathcal{A}))$.*
2. *$w \in \mathcal{L}(\mathcal{R}^k, (p, T))$ iff $w = x]_1 u$, for some $x \in \Gamma^*$, and $(p, [_2[_1 x]_1]_2) \in Attr_k(\{(q, [_2]_2) \mid q \in T\} \cup M(T))$.*

Proof. In the full version. □

Corollary 3. *$w \in \mathcal{L}(\mathcal{R}^k, q'')$ iff $(q, w) \in Attr_k(\mathcal{L}(\mathcal{A}))$.*

Proof. Immediate, using Theorem 2(1.) and Lemma 2(4.). □

6 Strategy Extraction

From the results of section 3 and section 5, it follows that one can decide for an arbitrary configuration whether it is in $pre^*(C)$ or it is in the winning region of player-k by simply checking membership in the corresponding automata. It is a natural question to ask if a configuration u is in $pre^*(C)$, can we also determine the sequence of transitions of $\mathcal{H}$ which starting from u reach a configuration in C or if u is in winning region of player-k then can we determine what is the winning strategy of player-k from u to reach set E.

These questions have relevance to synthesis of transition systems and have been investigated in [2] in the context of PDS. It turns out that accepting runs of the corresponding automata on a configuration encode such strategies. In [2], two kinds of strategies are synthesized for a player to win a reachability game (in its winning region) in a PDS. One of these is executable by a pushdown automata and another is a minimum cost strategy executable in linear time. In this section we generalize these results of [2] to the setting of order-2 *hpds* .

6.1 Strategy Executable by Order-2 Hpda

An order-2 *hpda* executing a strategy is a deterministic order-2 pushdown automata with input and output tapes. It reads the moves of player $1-k$ from the input tape and and outputs the moves of player k on the output tape. A more formal definition can be given as in section 3.3 in [2].

In this section, we show how to design a order-2 *hpda* to execute player-k's winning strategy in the second order *hpds* reachability game.

Theorem 3. *Let a game structure $\mathcal{H}$ for second order hpds and a regular set E be given. One can effectively design a second order hpda B such that for every configuration C in the winning region of player-k, after an appropriate initialization B's stack, B executes the winning strategy of player-k from position C. The initialization of B can be done in linear time in the length of C.*

Proof. We sketch the construction of B. The initialization of B's stack is by an accepting dag of automata computing $Attr_k(E)$ on input C. Let this dag be D. We assume that each node of D has more information than just a state of automata, for example it has information about transition taken at that node. We view D as layered dag, a sequence of set of nodes or transitions. This sequence is initially stored in the stack of B linearly, with root of D as top of B's stack. The control states of B include a subset of $\mathcal{R}^k$'s state and some auxiliary states. Initially B is in state q''. Transition function of B depends on its control state (say z) and topmost symbol (say ϕ) of B's stack, and also on the input tape if the control state of B corresponds to player $1-k$'s state in $\mathcal{H}$. We describe below the action of B in various cases which are grouped according to transitions in *hpds* .

Intuitively, D codes the winning strategy of player k from C. B walks through various paths in D (depending on player $1-k$'s choices). B's state represents state of the node B is currently at, while ϕ represents all nodes at that level of D.

(0) If $z = q^{''}$ and $(q^{''}, [_2[_1, q) \in \phi$, then B pops the topmost element of the stack and enters state q.

(i) If $z = (q, T)$, $q \in Q_k$ and $((q, T), a, (\mathbf{p}, \mathbf{T})) \in \phi$ then B outputs pop_1^p, pops topmost element of its stack and moves to state (p, T).

(ii) $z = (q, T)$, $q \in Q_k$ and $((q, T), a, (\mathbf{p}, \mathbf{T})_\mathbf{u}) \in \phi$ then B outputs $push_1^{p,u}$, pops topmost element of its stack and moves to state $(p, T)_u$.

(iii) $z = (q, T)$, $q \in Q_k$ and $((\mathbf{q}, \mathbf{T}), \mathbf{a}, (\mathbf{p}, \mathbf{T_1})_\mathbf{a} \wedge \bigwedge_{\mathbf{t} \in \mathbf{T_1}} (\mathbf{t}, \mathbf{T})_\mathbf{a}) \in \phi$ for some $T_1 \subseteq Q \cup S_\mathcal{A}$ then B outputs $push_2^p$ and pops the top element from the stack. (The current top of the stack has transitions with source $(p, T_1)_a$ and $(t, T)_a$ for each $t \in T_1$). B remembers the transition α begining with $(p, T_1)_a$ and modifies the current top element of stack to contain transitions with source $(t, T)_a$ only, for $t \in T_1$. (That is B pops the top element and pushes a new one which contains only transitions with source $(t, T)_a$ of the old top element).

Now B does a $push_2$ operation. Deletes the topmost element of the stack and pushes a dag expansion of transition α on the stack and changes its control state to (p, T_1).

(iv) $(\mathbf{q}, \mathbf{a}, \mathbf{pop_2^p}) \in \mathbf{\Delta}$ If $z = q$, $q \in Q_k$ and $(q, \downarrow), a, (\mathbf{p}, \mathbf{2}))\phi$ then B outputs pop_2^p, pops the top element from the stack and changes its control state to $(p, 2)$.

If $z = (r, 2)$ and $((r, 2), b, (\mathbf{r}, \mathbf{2})) \in \phi$ then B does pop_1 and remains in state $(r, 2)$.

If $z = (r, 2)$ and $((r, 2),]_1[_1, (\mathbf{r}, \downarrow)) \in \phi$ then B does pop_1 and enters state $(r, \downarrow)$.

If $z = (r, 2)$ and $((r, 2),]_1]_2, (\delta_{\mathcal{A}}(\mathbf{r'}, [_\mathbf{2}]_\mathbf{2}), \downarrow)) \in \phi$ then B enters the Halt state. If $z = (r, 2)$ and $((q, T), a, \mathbf{true}) \in \phi$ (corresponding to $(\mathbf{q}, \mathbf{a}, \mathbf{pop_2^p}) \in \mathbf{\Delta}$) then B does a pop_2^p operation. In the topmost element of the stack now it examines a transition with source $(p, T)_b$ for $b \in \Gamma$ and enters state $(p, T)_b$. [such a unique $(p, T)_b$ is guaranteed to be there, by action of B in step (ii)].

(v),(vi) Transitions related to $\mathcal{A}$:
If $z = q''$ or $z = (q, t)$ and the first or second transitions of group (v) $\in \phi$ then B enters a Halt state. /* A configuration in the target set is reached */.

(vii) If $z = q$ and $q \in Q_{1-k}$ then B reads the input to find transition say, β. B now finds transitions begining with the target set of β in the top stack symbol and proceeds to take same actions as in one of the earlier cases corresponding to β.

(viii) Transition is added in the saturation step:
Let $z = (p, T)_x$ and $((p, T)_x, \epsilon, S) \in \phi$. S is a conjunction of states. Further let l be the least number such that this transition is in δ_{l+1}.
B pops the topmost element from the stack and pushes a dag rooted at (p, T) for input x whose leaves are states S and has transitions in δ_l only. B changes its control state to (p, T).

This finishes the description of the *hpda* B. □

Note that unlike in [2], the strategy executed by this *hpda* is not constant time. This is because for a single pop_2 operation in *hpds* there are as many transitions (of group (iii).b) in the accepting dag as the number of elements in the order-1 stack. It seems possible to make the strategy constant time by doing a initialization of B's stack in several order-1 stacks each containing the portion of D corresponding to an order-1 stack in C. We have not worked out the details yet.

6.2 Computing Min-Cost Strategy

Let C be a configuration in the winning region of player k in a *hpds* reachability game for player-k. The number of steps (also called cost) in which player k is guaranteed to reach the target set E assuming the worst case behaviour from the opponent is easily seen to be the least j such that $C \in Attr_k^j(E)$. A winning strategy of player k in which he plays at any configuration a move which guarantees him to win the game in *minimum cost from that configuration* is called minimum cost winning strategy of player k.

Analogous to a result of [2], we show that min-cost value for player k from a configuration C can be characterized using accepting dags of $\mathcal{R}^k$ on input C. Further, there is an accepting dag of $\mathcal{R}^k$ on input C which codes a min-cost strategy of player k.

We define a cost labeled dag as a dag D alongwith a labeling by natural number of each node of D. The labeling value of node n, denoted $L(n)$, is defined using the rules below.

(0) Nodes with states in set $S_{\mathcal{A}} \times U_{\mathcal{A},\downarrow}$ or leaves marked 'true' have cost $L(n) = 0$.

(i) Transition rule used at n is $((q,T), a, (\mathbf{p}, \mathbf{T}))$, $q \in Q_k$
[Corresponding to $(q, a, pop_1^p) \in \Delta$]
Let n_1 be the child of n corresponding to $(\mathbf{p}, \mathbf{T})$ then $L(n) = 1 + L(n_1)$.

(ii) Transition rule used at n is $((q,T), a, (\mathbf{p}, \mathbf{T_1})_{\mathbf{a}} \bigwedge \wedge_{\mathbf{t} \in \mathbf{T_1}} (\mathbf{t}, \mathbf{T})_{\mathbf{a}})$, for some $T_1 \subseteq Q \cup S_{\mathcal{A}}$.
[Corresponding to $(q, a, push_2^p) \in \Delta$]
Let $n_0, n_1, \cdots n_r$ be children of n corresponding to $(p, T_1)_a, \{(t,T)_a \mid t \in T_1\}$ respectively. We have $L(n) = 1 + L(n_0) + max\{L(n_i) \mid 1 \leq i \leq r\}$

(iii) Transition rule used at n corresponds to $(\mathbf{q}, \mathbf{a}, \mathbf{pop_2^p}) \in \mathbf{\Delta}$ for $q \in Q_k$

(a) If the rule used is $((q, \downarrow), a, (\mathbf{p}, \mathbf{2}))$ let n_1 be the child of n corresponding to $(\mathbf{p}, \mathbf{2})$. then we have $L(n) = 1 + L(n_1)$.
(b) If the rule used is $((q,T), a, \mathbf{true})$ then we define $L(n) = 1$.
(c) If the rule used is $((r, 2), b, (\mathbf{r}, \mathbf{2}))$ let n_1 be the child of n corresponding to $(\mathbf{r}, \mathbf{2})$. then we define $L(n) = L(n_1)$.
(d) If the rule used is $((r,2),]_1[_1, (\mathbf{r}, \downarrow))$, let n_1 be the child of n corresponding to $(\mathbf{r}, \downarrow)$ then we define $L(n) = L(n_1)$.
If the rule used is $((r,2),]_1]_2, (\delta_{\mathcal{A}}(\mathbf{r'}, [_\mathbf{2}]_\mathbf{2}), \downarrow)$ then we define $L(n) = 0$

(iv) Transition rule used at n is $(\mathbf{q}, \mathbf{a}, \mathbf{push_1^{p,u}}) \in \mathbf{\Delta}$, $q \in Q_k$
[Corresponding to $(\mathbf{q}, \mathbf{a}, \mathbf{push_1^{p,u}}) \in \mathbf{\Delta}$]
Let n_1 be the child of n corresponding to $(\mathbf{p}, \mathbf{T})_{\mathbf{u}}$. We define $L(n) = 1 + L(n_1)$.

(v),(vi) Transitions related to $\mathcal{A}$: If rule used at n is from groups (v) or (vi) then we define $L(n) = 0$.

(vii) Let n correspond to a state $q \in Q_{1-k}$. (the rule used is from group (vii))
In this case n has children corresponding to each rule of Δ applicable at n. Let the cost of choosing these moves be $\{c_1, \cdots, c_r\}$. then $L(n) = max\{c_1, \cdots, c_r\}$.

(viii) Transition used at n is $((p,T)_x, \epsilon, S)$ (introduced in saturation step)
[Which is added during saturation procedure and is in δ_{l+1}.]
Let D_1 be a cost labeled dag on input x for this transition, and using transitions of δ_l only. We define $L(n)$ = value at the root of D_1.
(Note the use of recursion here as labeling inside D_l must satisfy the cost labeling rules. Also note that the choice of dag D_1, is not unique.)

The following theorem is analogous to Theorem 2 of section 5 and is proved in the similar way.

Theorem 4. *Let D be a cost labeled dag for input w with root of D marked with state m and cost with j.*

1. *$(q, [_2[_1 w) \in Attr_k^j(\mathcal{L}(\mathcal{A}))$ iff there is an accepting dag D of $\mathcal{R}^k$ for input w, rooted at $(q, \downarrow)$ and with cost labeled j at its root.*
2. *Let $p \in Q, T \subseteq Q \cup S_{\mathcal{A}}$ and $x \in \Gamma^*$.*
$(p, [_2[_1 x]_1]_2) \in Attr_k^j(\{(q, [_2]_2) \mid q \in T\} \cup M(T))$ iff there is an accepting dag D of $\mathcal{R}^k$ for input $x]_1$, rooted at (p, T) and with cost labeled j at its root.

Proof. It follows by considering cost labeling in the proof of Theorem 2. □

We construct a minimum cost dag of $\mathcal{R}^k$ on C in bottom up manner, considering at each step all possible moves of $\mathcal{R}^k$ and choosing the one leading to minimum cost and satisfying the rules of cost labeled dag. For transitions of the kind $((p, T)_x, \epsilon, S)$, we use the algorithm recursively constructing the dag with transitions of lower levels only. For a fixed $\mathcal{H}, \mathcal{A}$ this can be done in time linear in the length of the configuration. This leads to the following theorem.

Theorem 5. *For a fixed $\mathcal{H}$, $\mathcal{A}$, k, there is a linear time algorithm which takes as its input a configuration of player k and outputs the minimum cost move of player k in the reachability game above if the input configuration is in the winning region of player k.*

7 Conclusion

We have given an alternative construction of finite automata recognizing $pre^*_{\mathcal{H}}(C)$, where C is a regular set of configurations of a second order pushdown system $\mathcal{H}$. Our construction is explicit and simple to understand. It also yields results about strategy synthesis which are a generalization from pushdown reachability games to second order pushdown reachability games. Our approach can be extended to pushdown systems of higher than second order also though some new phenomenon arise there which make the construction more involved. This will be future work.

References

1. Hague, M., Ong, L.: Symbolic backwards-reachability analysis for higher-order pushdown systems. In proc. FoSSaCS, 2007 (Also downloadable from www.comlab.ox.ac.uk/oucl/work/matthew.hague)
2. Cachat, T.: Symbolic strategy synthesis for games on pushdown graphs. In proc. ICALP (2003) 315–333
3. Bouajjani, A., Meyer, A.: Symbolic reachability analysis of higher-order context-free processes. In proc. FSTTCS (2004) LNCS 3328.
4. Bouajjani, A., Esparza, J., Maler O.: Reachability analysis of pushdown automata: Applications to model checking. In proc. Concur (1997) 135-150
5. Walukiewicz, I.: Pushdown processes: games and model checking. Information and computation **164** (2001) 234–263
6. Carayol, A., Wohrle, S.: The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In proc. FSTTCS (2003) 112-123
7. Maslov, A. N.: Multilevel stack automata. In Problems of Information Transmission, 15:1170-1174, 1976.
8. Engelfriet, J.: Iterated stack automata and complexity classes. Information and computation **95** (1991) 21–75
9. Damm, W.: The IO and OI hierarchies. Theoretical Computer Science **20** (1982) 95–208
10. Loding, C., Thomas, W.: Alternating Automata and Logics over Infinite Words, IFIP TCS'00, LNCS 1872, pp. 521-535, 2000.

11. Caucal, D.: On infinite terms having a decidable monadic theory. In proc. MFCS (2002) 165-176.
12. Gradel, E., Thomas, W., Wilke, Th.: Automata, logics, and infinite games, LNCS 2500, Springer, 2002.
13. Kupferman, O., Vardi, M. Y.: An Automata-Theoretic Approach to Reasoning about Infinite-state Systems, In Proc. CAV 2000, LNCS 1885, 2000.

Decidability and Complexity Analysis of Forbidden State Problems for Discrete Event Systems

Hsu-Chun Yen[*]

Dept. of Electrical Engineering, National Taiwan University
Taipei, Taiwan 106, R.O.C.

Abstract. The conventional forbidden state problem for discrete event systems is concerned with the issue of synthesizing a maximally permissive control policy to prevent a discrete event system from reaching any forbidden state during the course of its computation. In this paper, we regard the forbidden state problem as a decision problem, and investigate the decidability/complexity issue of the problem under two new types of control policies, namely, *non-blocking* and *fair* policies, for finite state systems and Petri nets.

1 Introduction

In supervisory control of discrete event systems [7], a central problem, known as the *forbidden state problem*, asks for the design of a supervisor to guide a system with controllable/uncontrollable transitions to avoiding a certain set of *forbidden states* during the course of its computation. At any state, a *state-feedback* or *event-feedback* supervisory control policy is capable of disabling certain controllable transitions, while leaving all the uncontrollable transitions enabled. It is not hard to see that if a control policy avoiding forbidden states exists, then by shutting down all the controllable transitions at any state the goal can certainly be achieved. Traditionally, the forbidden state problem in supervisory control seeks for a control policy that is *maximally permissive* in the sense that the policy enforces the weakest control among all policies avoiding forbidden states (if they exist) [1, 2, 5].

In this paper, we consider the forbidden state problem with respect to two new types of control policies, namely, *non-blocking policy* and *fair policy*, under either the state-feedback or the event-feedback control. In addition, we focus on the decision version of the forbidden state problem, i.e., deciding whether a control policy exists or not under which the system can always be controlled to avoid reaching any forbidden state. Under a non-blocking policy, if in a state from which only controllable transitions are enabled, a control policy is not allowed to 'shut down' the computation by disabling all potentially executable

[*] E-mail: yen@cc.ee.ntu.edu.tw. Research supported in part by *National Science Council* Grant NSC-95-2221-E-002-075 and *Excellent Research Projects of National Taiwan University*, 95R0062-AE00-05.

controllable transitions. Under a fair policy, if without the presence of the controller, a controllable transition is enabled infinitely many times along an infinite computation, then any control action involving the transition must not be denied by the control policy forever. We investigate the forbidden state problem from a complexity/decidability aspect for two types of systems: finite-state systems and systems that can be modelled by Petri nets, which are the two most extensively studied models in the community of discrete event systems, as indicated in many articles (e.g., [1, 7]). Our results are summarized in the following tables:

Finite-state systems	Conventional	Non-blocking	Fair
State-feedback	NL-complete	PTIME-complete	NP-complete
Event-feedback	NL-complete	PTIME-complete	PTIME-complete

Table 1. Complexity results of various forbidden state problems for finite state systems.

Petri nets	Conventional	Non-blocking	Fair
State-feedback	Decidable*	Co-r.e.-complete	Σ_1^1-hard
Event-feedback	Decidable*	Co-r.e.-complete	Σ_1^1-hard

Table 2. Decidability results of various forbidden state problems for Petri nets. (Co-r.e. denotes the complement of recursively enumerable sets; Σ_1^1 denotes the first level of the analytical hierarchy. Note: * assuming the set of forbidden states to be upward-closed.)

The rest of the paper is organized as follows. Section 2 gives the basic definitions of controlled transition systems, non-blocking and fair control policies, as well as various forbidden state problems. Complexity analysis regarding state-feedback and event-feedback forbidden state problems for finite state systems is given in Section 3. Section 4 is concerned with the analysis of various forbidden state problems for Petri nets from a decidability viewpoint.

2 Preliminaries

Let N denote the set of nonnegative integers, and N^k the set of vectors of k nonnegative integers. A *transition system* M is a 4-tuple (S, T, δ, q_0), where S is a (possibly infinite) set of *states*, T is a finite set of *transitions*, $q_0 \in S$ denotes the *initial state*, and $\delta \subseteq (S \times T \times S)$ defines the *transition relation*. For convenience, $(s, t, s') \in \delta$ is also written as $s \xrightarrow{t} s'$, meaning that executing transition t from state s yields a new state s'. We define $en(s) = \{t \mid \exists s' \in S, (s, t, s') \in \delta\}$ to be the set of transitions enabled at state s. Given a sequence of transitions $\sigma{=}t_1 \cdots t_n$, we write $s_0 \overset{t_1 \cdots t_n}{\rightarrow} s_n$ (or $s_0 \xrightarrow{\sigma} s_n$) if $\forall 0 \leq i \leq n-1, s_i \overset{t_{i+1}}{\rightarrow} s_{i+1}$. We let $\xrightarrow{*}$ denote the transitive closure of $\rightarrow$. That is, $s \xrightarrow{*} s'$ if there exists a sequence of transitions $\sigma \in T^*$ such that $s \xrightarrow{\sigma} s'$. For an infinite string $\sigma = t_1 t_2 \cdots t_i \cdots \in T^\omega$, we write $s_0 \xrightarrow{\sigma}$ if $\forall i \geq 1$ $s_0 \xrightarrow{\sigma_i}$, where $\sigma_i = t_1 t_2 \cdots t_i$. The *reachability set* with respect to a start state s and a (finite or infinite) computation σ is $R(M, s, \sigma) = \{s' \mid s \xrightarrow{\sigma'} s', \text{ where } \sigma' \text{ is a prefix of } \sigma\}$, i.e., the set of states encountered along the computation σ. We write $R(M, s) = \bigcup_{\sigma \in T^*} R(M, s, \sigma)$ to denote the *reachability set* of M from state s.

A *controlled transition system* is a transition system $M{=}(S, T, \delta, q_0)$ in which the set of transitions is divided into two disjoint subsets T_c and T_u with $T_c \cap T_u = \emptyset$ and $T_c \cup T_u = T$. Transitions in T_c and T_u are called *controllable* and *uncontrollable* transitions, respectively. Let $\mathcal{C} \subseteq 2^{T_c}$. Each element of $\mathcal{C}$ is called a *control action.* W.l.o.g., we always label the elements of $\mathcal{C}$ as $C_1, \ldots, C_m$ (i.e., $\mathcal{C}{=}\{C_1, \ldots, C_m\}$, for some m), and we also assume that $T_c = \bigcup_{i=1\ldots m} C_i$ throughout the rest of this paper. A *state-feedback control policy* is a mapping $h : S \rightarrow \mathcal{C}$, specifying, for each state $s \in S$, the set $h(s)$ $(\in \mathcal{C})$ which represents the set of controllable transitions enabled under the policy h. An *event-feedback control policy* is a mapping $h : T^* \rightarrow \mathcal{C}$ from the set of possible event histories to $\mathcal{C}$. From the above definitions, it is clear that the control decision under a state-feedback policy depends only on the current state of the system, whereas under the event-feedback policy, the decision also hinders on the history of the computation reaching the current state. With respect to a policy h, a sequence of transitions $\sigma = t_1 \cdots t_n$ is a legitimate computation from state s_0 under the state-feedback (resp., event-feedback) policy if there exist $s_1, \cdots, s_n \in S$ such that $\forall 1 \leq i \leq n$, $s_{i-1} \xrightarrow{t_i} s_i$ and if $t_i \in T_c$, then $t_i \in h(s_i)$ (resp., $t_i \in h(t_1 \cdots t_{i-1})$) (i.e., controllable transition t_i must be enabled under h).

In this paper, we are concerned with the so-called *forbidden state problem* (*FSP*, for short) for controlled transition systems. FST is the problem of, given a controlled transition system $M{=}(S, T, \delta, q_0)$ (where $T = T_u \cup T_c$), a $\mathcal{C} \subseteq 2^{T_c}$, and a set Q of *forbidden states*, determining whether there exists a policy under which all the (finite or infinite) computations σ of M from q_0 never reach a state in Q. Over the years, FSP has been a focal point of research in the community of discrete event dynamic systems with its primary emphasis lying in finding a *maximally permissive* policy (if it exists) (see, e.g., [1, 2, 5]. In this paper, our interest in FSP is on the complexity of deciding whether a forbidden-state-avoidance control policy exists or not for certain finite/infinite systems. In addition to the conventional control policy, we consider two new types of control policies with *non-blocking* and/or *fairness* constraints imposed on the policy, and the control decision is based on state-feedback or event-feedback. Suppose $M{=}(S, T, \delta, q_0)$ $(T = T_c \cup T_u)$ is a controlled transition system and $\mathcal{C}{=}\{C_1, \ldots, C_m\}$ is the set of control actions.

1. *Non-blocking policy*: A policy h is *non-blocking* if for every state s, if $en(s) \subseteq T_c$, then the control policy must enable some control action C with $en(s) \cap C \neq \emptyset$ at state s. In words, if in a state from which only controllable transitions are enabled, a control policy is not allowed to 'shut down' the computation by disabling all potentially executable controllable transitions.

2. *Fair policy*: A policy h is said to be *fair* if it is non-blocking, and for every infinite computation $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \cdots s_{i-1} \xrightarrow{t_i} s_i \xrightarrow{t_{i+1}} \cdots$ and for every $1 \leq j \leq m$, if there exist infinitely many $i_1, i_2, ...$ such that $C_j \cap en(s_{i_l}) \neq \emptyset$ $(\forall l \geq 1)$ then there exist infinitely many $k_1, k_2 \cdots$ such that C_j is chosen by the control policy at states s_{k_l}, $\forall l \geq 1$. In words, if without the presence of the controller, a controllable transition is enabled infinitely many times along an infinite computation, then any control action involving the transition must not be denied by the control policy forever.

Consider the following decision versions of three variants of FSP, namely, the conventional *Forbidden State Problem* (FSP), the *Non-blocking Forbidden State Problem* (NFSP), and the *Fair Forbidden State Problem* (FFSP), under state-feedback and event-feedback control. All three problems ask the following question: Given a controlled transition system $M{=}(S, T, \delta, q_0)$, a $\mathcal{C} \subseteq 2^{T_c}$, and a set Q of forbidden states, deciding whether there exists a (state-feedback or event-feedback) control policy h such that $(\bigcup_{\sigma \textit{ is legal under } h} R(M, q_0, \sigma)) \cap Q = \emptyset$, i.e., none of the (finite or infinite) computations from q_0 under h ever reaches a state in Q. For FSP, h is just a conventional control policy, whereas for NFSP and FFSP, h is required to be *non-blocking* and *fair*, respectively.

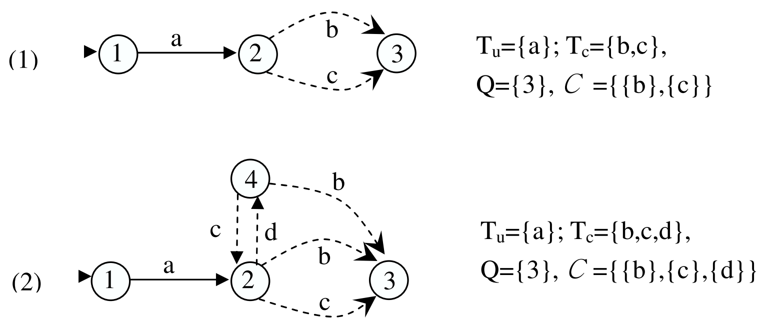


Fig. 1. FSP under the conventional, non-blocking and fair control policies.

To illustrate the disparities among the aforementioned three control policies, consider an example shown in Figure 1. For the system in Figure 1(1), the system is forbidden-state-free under the conventional control policy by letting $h(2) = \emptyset$ (i.e., disabling b and c in state 2, assuming a state-feedback policy). However, turning off b and c in state 2 violates the non-blocking constraint; hence, the system is not forbidden-state-free under the non-blocking policy. Now consider the system in Figure 1(2). If we always choose $h(2) = \{d\}$ and $h(4) = \{c\}$ (yielding an infinite computation $1 \xrightarrow{a} 2 \xrightarrow{d} 4 \xrightarrow{c} 2 \xrightarrow{d} 4 \xrightarrow{c} 2 \cdots$), then the system is forbidden-state-free under the non-blocking policy. However, the above policy is not fair as the control action $\{b\}$ was enabled infinitely many times but was never chosen under the above policy.

As finite state systems and Petri nets are without doubt the two most popular (and important) abstract models in the study of FSP for discrete event systems (see, e.g., [1]), we focus on various FSP for these two models from the complexity/decidability viewpoints in the rest of this paper.

3 Finite state systems

In this section, we study the computational complexity of state-feedback and event-feedback FSP, NFSP, and FFSP for finite state systems.

Theorem 1. *State-feedback and event-feedback NFSP are PTIME-complete for finite state systems.*

Proof. (Sketch) We first show the upper bound, i.e., NFSP is in PTIME. Consider a finite state system $M{=}(S, T, \delta, q_0)$ with $\mathcal{C} = \{C_1, ..., C_m\}$, and let Q the the set of forbidden states.

We construct a sequence of disjoint sets (layers) of states $U_0, U_1, ...$ in the following way:

(1) $U_0 = Q \cup \{q \mid \exists p \in Q, \sigma \in T_u^*, q \xrightarrow{\sigma} p\}$, and
(2) $\forall i \geq 1$, a state $q \notin \bigcup_{0\leq l\leq i-1} U_{i-1}$ is included in U_i iff there exist a state q' such that
   (a) $q \xrightarrow{\sigma} q'$, for some $\sigma \in T_u^*$
   (b) $en(q') \subseteq T_c$, and
   (c) $\forall 1 \leq j \leq m, (C_j \cap en(q') \neq \emptyset) \implies \exists t \in C_j, q' \xrightarrow{t} q''$, for some $q'' \in \bigcup_{0\leq l\leq i-1} U_{i-1}$

Table 3. Procedure NFS

Intuitively speaking, U_i consists of those states each of which can reach some state in a lower layer by a sequence of uncontrollable transitions followed by a controllable transition, regardless of which control action is taken at the end. What (c) says is that regardless of which control action C_i is chosen by the supervisor, a state q'' in $\bigcup_{0\leq l\leq i-1} U_{i-1}$ is always reachable. As the system is of finite-state, the above procedure of constructing the sequences U_0, U_1, ... eventually terminates when no new states can be added. Once this happens, it is reasonably easy to see that from the initial state q_0, there is a non-blocking policy avoiding Q iff $q_0 \in (S - \bigcup_{i\geq 0} U_i)$. As the number of layers is bounded by $|S|$, and each of steps (a), (b) and (c) can be done in polynomial time (for example, (a) involves checking graph reachability), the entire procedure is clearly doable in PTIME. Also note that the above procedure works for both state-feedback and event-feedback policies.

Now we show the PTIME hardness result, which is done by reducing from a well-known PTIME-complete problem, namely, the *path system problem* [4]. Recall that a *path system* is a 4-tuple $\mathcal{P} = (X, R, I, Z)$, where X is a finite set of *nodes*, $I(\subseteq X)$ is the set of *starting nodes*, $Z(\subseteq X)$ is the set of *terminal nodes*, and $R(\subseteq X \times X \times X)$ is the set of *rules*. A node $x \in X$ is said to be *admissible* if either $x \in Z$ or $\exists y, z \in X$ such that $(x, y, z) \in R$ and both y and z are admissible. The path system $\mathcal{P} = (X, R, I, Z)$, is said to have a solution iff there is an admissible node in I.

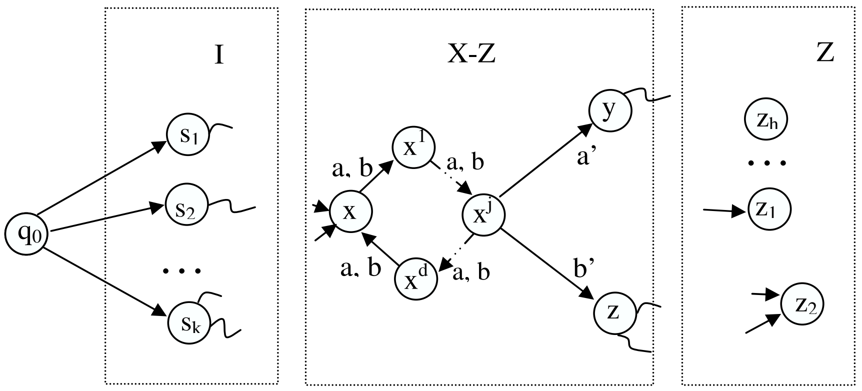


Fig. 2. Simulating an instance of the path system problem.

To show our lower bound, given a path system $\mathcal{P} = (X, R, I, Z)$ we show how to construct a finite state system $M{=}(S, T, \delta, q_0)$ with $\mathcal{C} = \{C_1, ..., C_m\}$, and a forbidden state set $Q \subseteq S$ in such a way that starting from the initial state q_0, M can be controlled under a non-blocking policy to avoid reaching Q iff $\mathcal{P}$ has no solution. Without loss of generality, let $I = \{s_1, ..., s_k\}$, $Z = \{z_1, ..., z_h\}$ and $R = \{r_1, ..., r_d\}$. M is defined to be the following (see Figure 2):

$T = \{a, a', b, b', u\}$, $T_c = \{a, a', b, b'\}$, $T_u = \{u\}$,
$S = X \cup \{q_0\} \cup \{x^1, x^2, ..., x^d \mid \forall x \in (X - Z)\}$,
δ: (Let "$q \in \delta(p, a/b)$" mean that both $\delta(p, a) = q$ and $\delta(p, b) = q$)
(1) $\forall 1 \leq i \leq k, s_i \in \delta(q_0, u)$, (For clarity the label u is not shown in Figure 2)
(2) $\forall x \in (X - Z), \forall j, 1 \leq j \leq d - 1, x^1 \in \delta(x, a/b), x^{j+1} \in \delta(x^j, a/b)$, and $x \in \delta(x^d, a/b)$,
(3) $\forall x \in (X - Z)$, if $r_j = (x, y, z)$ then $y \in \delta(x^j, a')$ and $z \in \delta(x^j, b')$,

Let the set of control actions $\mathcal{C}$ be $\{\{a, a'\}, \{b, b'\}\}$. We also let the set of forbidden states Q be Z. We claim that $\mathcal{P}$ has <u>no</u> solution iff M can be controlled to avoid reaching Q under a non-blocking policy. To show the only-if part, assume that $\mathcal{P}$ has no solution. Let A and B be the sets of nonadmissible and admissible nodes, respectively. Let $A' = A \cup \{x^1, ..., x^d \mid x \in A\}$, and $B' = B \cup \{x^1, ..., x^d \mid x \in (B - Z)\}$.

Clearly, $I \subseteq A'$ since $\mathcal{P}$ has no solution. According to the construction of M, for every node x^j in A' there exists at most one edge that connects x^j to some node in B'. (Otherwise, the corresponding x would be admissible.) Consequently there exists a non-blocking policy that avoids Q. The policy is designed in such a way that if $x^j \xrightarrow{a'} y$ (resp., $x^j \xrightarrow{b'} y$), for some node y in B', then the non-blocking policy h will enable the edge $x^j \xrightarrow{b'} x^{j+1}$ (resp., $x^j \xrightarrow{a'} x^{j+1}$). By doing so, M will not be 'forced' to enter B' under the non-blocking policy. It is also worthy of pointing out that the above is the only situation in which the controller needs to make a 'wise decision.' For the rest, the controller is free to enable a, b or both; hence, h is clearly non-blocking.

Now assume that $\mathcal{P}$ has a solution, i.e., some state, say s' in I is admissible. From the construction, we show that there does not exist a non-blocking on M that always avoids Q. Let H be the 'parse' tree certifies the admissibility of s'. That is, H is a finite labelled binary tree where:

- each node in H is either a leaf or has both a left child and a right child,
- each node is labelled by an element of X,
- each leaf is labelled by an element of Z,
- whenever an internal node is labelled by x and its left and right children are labelled by y and z, respectively, it must be that $(x, y, z) \in R$,
- the root is labelled by s'.

Clearly each node in such a tree must be labelled by an admissible element in X. According to our construction (see (1) and (2)), any non-blocking policy always has a chance to take the computation from q_0 to s' (because of $q_0 \xrightarrow{u} s'$). From s', the above parse tree ensures that some leaf node (corresponding to a state in $Z(= Q)$) to be unavoidable under any non-blocking policy. The above works for both state-feedback and event-feedback policies. □

Theorem 2. *State-feedback FFSP is NP-complete for finite state systems.*

Proof. (Sketch) The problem is clearly solvable in NP. The lower bound proof is done by reducing from the 3SAT. Let $C = \{D_1, ..., D_m\}$ be the set of clauses and $V = \{x_1, ..., x_n\}$ be the set of variables in an instance of 3SAT, where $D_i = \{\alpha_{i1}, \alpha_{i2}, \alpha_{i3}\}$ and $\alpha_{ij} \in \{x, \overline{x} | x \in V\}$ for $1 \leq i \leq m, 1 \leq j \leq 3$. W.l.o.g., we assume that none of the D_i contains both x_j and $\bar{x}_j$, for any j. We construct a finite state system $\mathcal{M}$=(S, T, δ, q_0), a set of control actions $\mathcal{C}$, and a forbidden state set Q in such a way that $\mathcal{M}$ can be controlled under a fair state-feedback policy to avoid Q iff the 3SAT instance is satisfiable. The construction is shown in Figure 3. In addition to states q_0, q_{n+1}, q and q', $\mathcal{M}$ contains modules V_j $(1 \leq j \leq n)$, L_i $(1 \leq i \leq m)$, and X. Module V_j (resp., L_i) corresponds to variable x_j (resp., clause D_i). Now we describe each of the modules and the connecting edges between modules in detail.

- The node set of V_j is $\{q_j, q'_j\} \cup \{q^i_j, \overline{q}^i_j \mid 1 \leq j \leq m+1\}$, $\forall 1 \leq j \leq n$,
- The node set of L_i is $\{c^i_j \mid 1 \leq j \leq n\}$, $\forall 1 \leq i \leq m$,
- The node set of X is $\{z\} \cup \{r_i \mid 1 \leq i \leq m\}$,

– The edge set of $\mathcal{M}$ includes
  1. $\forall 1 \le j \le n$, $q_j \xrightarrow{t} q_j^1$, $q_j \xrightarrow{f} \bar{q}_j^1$, $q_j^{m+1} \to q_j'$, $\bar{q}_j^{m+1} \to q_j'$, $q_j' \to q_{j+1}$,
     if $x_j \in D_i$ then ($q_j^i \to c_j^i$ and $c_j^i \xrightarrow{i} q_j^{i+1}$) else $q_j^i \to q_j^{i+1}$,
     if $\bar{x}_j \in D_i$ then ($\bar{q}_j^i \to c_j^i$ and $c_j^i \xrightarrow{i} \bar{q}_j^{i+1}$) else $\bar{q}_j^i \to \bar{q}_j^{i+1}$
  2. $\forall 1 \le i \le m$ $r_i \to r_{i+1}$, $r_i \xrightarrow{i} z$
  3. $q_0 \to q_1$, $q_{n+1} \to r_1$, $r_m \to q$, $q \xrightarrow{t} q'$, $q' \xrightarrow{f} q_0$.

In the above construction, un-annotated edges correspond to uncontrollable transitions, and each label associated with an annotated edge corresponds to a controllable transition. Let the set of control actions be $\mathcal{C}=\{\{t\}, \{f\}, \{1\}, ..., \{m\}\}$, and z be the only forbidden state.

The crux of the simulation is the following. A truth assignment of the 3SAT instance corresponds to a path from q_0 to q_{n+1} along which, for each V_j, $1 \le j \le n$, the path traverses either the left branch (corresponding to assigning x_j to **false**) or the right branch (corresponding to assigning x_j to **true**). If variable x_j (or $\bar{x}_j$) is in clause D_i, then node c_j^i along with its outgoing controllable transition labelled i will be included in the path. Note that in our construction, a controllable transition labelled i leading $\mathcal{M}$ to the forbidden state z is associated with node r_i, $1 \le 1 \le m$. Since each of such r_i, $1 \le 1 \le m$ is on a loop (from q_0 to q_{n+1} and back) that $\mathcal{M}$ repeats forever, $r_i \xrightarrow{i} z$ is forced to take place unless the control policy already performs i somewhere along the loop. (Recall the definition of a fair control policy.) Consequently, to suppress each of the $r_i \xrightarrow{i} z (1 \le i \le m)$ under a fair control policy, one of the outgoing transitions labelled i must take place in Module L_i, meaning that the truth assignment corresponding to the infinite computation under a fair policy satisfies all clauses. The remaining details are easy. □

Note that in the NP-hardness proof of the above theorem, the construction does not work under the event-feedback policy. To see this, consider state q_j in Figure 3. An event-feedback policy allows the controller to enable t and f at q_j at different points in time, making the simulation of a truth value assignment for x_j to be illegitimate. It turns out that event-feedback FFSP is solvable in PTIME, as our next theorem shows.

To reason about event-feedback FFSP, a graph-theoretic approach is used. We associate a graph $G_{M,\mathcal{C}}$ to a finite state system $M=(S, T, \delta, q_0)$ with $\mathcal{C} = \{C_1, ..., C_m\}$ the set of control actions. The edge-labelled graph $G_{M,\mathcal{C}} = (V, E)$ (with possibly self-loops and parallel edges) is such that

– $V = S$, and
– edge $(p, q)_l$ ($l \in \{0, 1, ..., m\}$ denotes the label of the edge) is in E if $\exists t \in T$ $p \xrightarrow{t} q$, and $t \in C_l$ if $l > 0$; $l = 0$ if $t \in T_u$.

In words, each node of the graph corresponds to a state in the finite state system, and an edge (p, q) is labelled 0 (resp, $l > 0$) if it corresponds to an uncontrollable transition (resp., a controllable transition in control action set C_l). A *strongly*

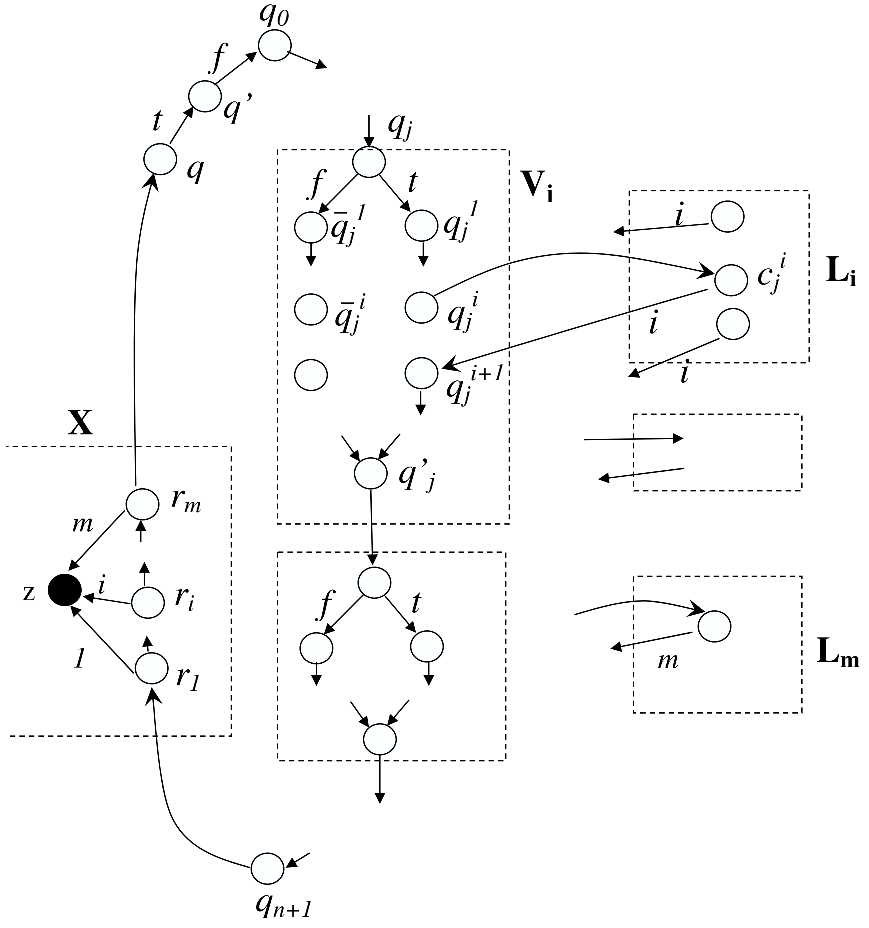


Fig. 3. Simulation of a 3SAT instance.

connected component (SCC) $G' = (V', E')$ $(V' \subseteq V, E' \subseteq E)$ of $G_{M,\mathcal{C}}$ is called a *fair-SCC* if

(C1) $\forall p \in V', \forall q \in V, ((p,q)_0 \in E \Longrightarrow (p,q)_0 \in E')$,
(C2) $\forall p, q \in V', \forall q' \in V, \forall l \in \{0, 1, ..., m\}, (((p,q)_l \in E' \wedge (p,q')_l \in E) \Longrightarrow (p,q')_l \in E')$
(C3) $\forall p \in V', \forall q \in V - V', (((p,q)_l \in E - E') \Longrightarrow \exists p', q' \in V', (p',q')_l \in E')$.

Consider the SCC consisting of nodes p, q, r and their adjacent solid edges in Figure 4. It is not hard to see that Figure 4(A) is a fair-SCC, as it meets all three conditions above. The SCC in Figure 4(B), however, is not a fair-SCC as condition (C1) (requesting $(r,s)_0$ to be in the SCC) is violated. Also Figures

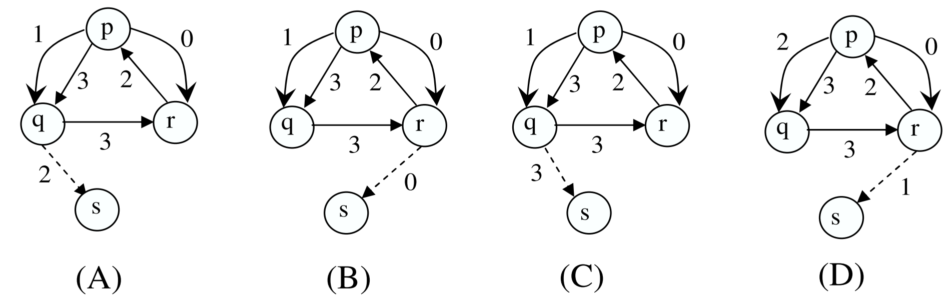


Fig. 4. Examples of fair and unfair SCCs.

4(C) and (D) are not fair-SCCs as conditions (C2) and (C3), respectively, are violated.

Intuitively, a fair-SCC is meant for capturing the set of states that occur infinitely often under a fair event-feedback control policy. Condition (C1) ensures that all the uncontrollable transitions emanating from states in a fair-SCC must also be included in the SCC. Condition (C2) says that at any state in the fair-SCC, outgoing transitions belonging to the same control action set must all be included or all be excluded in the SCC. Condition (C3) is for ensuring the control policy to be fair.

Now we are ready to show the PTIME-completeness result for event-feedback FFSP.

Theorem 3. *Event-feedback FFSP is PTIME-complete for finite state systems.*

Proof. (Sketch) We first show the upper bound. There are two cases that the supervisor can guide the system to avoid reaching any forbidden state. One case is when the computation (possibly a tree structure) under control eventually reaches a dead state (i.e., a state at which no transitions are enabled) which is not a forbidden state. For this case, a breadth-first search approach can clearly be applied to finding such a computation in polynomial time. Now consider the the second case in which the computation under control is infinite. In what follows, we illustrate how such an infinite computation can be found in polynomial time.

Given a finite state system $M{=}(S, T, \delta, q_0)$ with $\mathcal{C} = \{C_1, ..., C_m\}$ as the set of control actions, and $Q \subseteq S$ the set of forbidden states, we remove from $G_{M,\mathcal{C}}$ all the nodes in $\{q \mid q \in Q,\ \text{or}\ \exists p \in Q, \exists \sigma \in (T_u)^*, q \xrightarrow{\sigma} p\}$ and their adjacent edges, and let the resulting graph be $G' = (V', E')$. The idea behind our polynomial time algorithm is to identify all the *maximum* fair-SCCs of G', each of which is associated with an event-feedback fair policy that is capable of avoiding Q once a state in such a fair-SCC is entered. Then by replacing each of such *maximum* fair-SCCs by a dead state (a state without outgoing transitions), then a fair policy avoiding Q in the new graph (system) must be one that eventually guides the system to dead states along all possible computations, which is solvable in polynomial time following our earlier discussion.

Algorithm FSCC

```
(1) Find the set $H = \{G_1, ..., G_k\}$ of maximum strongly connected components
    of $G'$;
(2) If $H = \emptyset$, then halt;
(3) Choose some $\bar{G} \in H$.

    if $\bar{G}$ is a fair-SCC, the output $\bar{G}$ and goto (2), else
    $\bar{G}$ must violate one of Conditions C1, C2, and C3 in the definition of a fair-SCC
        (a) Violating Conditions C1 or C3: find $q$ in $\bar{G}$ such that
            either $(q, p)_0 (\in E)$ is not in $\bar{G}$ (i.e., violating C1) or there is an
            edge $(q, q')_l$ outside $\bar{G}$ and no edge in $\bar{G}$ is labelled $i$ (i.e., violating C3).
            Let $D$ be the subgraph of $\bar{G}$ by removing $q$ and its adjacent edges.
            Decompose $D$ into maximum strongly connected components,
            say $R_1, ..., R_f$, and let $H = (H - \{\bar{G}\}) \cup \{R_1, ..., R_f\}$; goto (2)
        (b) Violating Conditions C2: find nodes $q, q'$ and edge $(q, q')_l$ in $\bar{G}$
            and edge $(q, r)_l$ outside $\bar{G}$. Let $D$ be the subgraph of $\bar{G}$
            by removing all outgoing edges from $q$ labelled $l$.
            If $D$ is strongly connected, then let $H = (H - \{\bar{G}\}) \cup \{D\}$; goto (2),
            else decompose $D$ into maximum strongly connected components,
            say $R_1, ..., R_f$, and let $H = (H - \{\bar{G}\}) \cup \{R_1, ..., R_f\}$; goto (2)
```

Note that constructing the set of maximum strongly connected components and deciding whether a subgraph is a fair-SCC are clearly doable in polynomial time. Each time Algorithm FSCC returns to step (2), at least one edge or node is removed from the graph; hence, the number of iterations of the procedure is polynomial in the size of the graph. As a result, Algorithm FSCC is in PTIME. Once the set of all maximum fair-SCC is identified, the remaining details follow from the discussion in the beginning of this proof.

PTIME-hardness follows from a similar proof of that in Theorem 1. □

Theorem 4. *State-feedback and event-feedback FSP are NL-complete for finite state systems.*

Proof. (Sketch) It suffices to consider the situation when all the controllable transitions are disabled (i.e., the controller enforces the strongest control). In this case, the problem boils down to testing whether any of the forbidden states is reachable from the initial state, which is a problem solvable in nondeterministic logspace. □

4 Petri nets

A *Petri net* is a 3-tuple (P, T, φ), where P is a finite set of *places*, T is a finite set of *transitions*, and φ is a *flow function* $\varphi : (P \times T) \cup (T \times P) \rightarrow N$. Traditionally, a *state* of a PN is called a *marking*, which is a mapping $\mu : P \rightarrow N$ (μ assigns tokens to each place of the net). A transition $t \in T$ is *enabled* at a marking μ

iff $\forall p \in P$, $\varphi(p,t) \leq \mu(p)$. If a transition t is enabled, it may *fire* by removing a token from each input place and putting a token in each output place. We then write $\mu \xrightarrow{t} \mu'$, where $\mu'(p) = \mu(p) - \varphi(p,t) + \varphi(t,p)\ \forall p \in P$. A *marked* PN is a pair $((P,T,\varphi),\mu_0)$, where (P,T,φ) is a PN, and μ_0 is called the *initial marking.* See [6] for more about Petri nets. A marked PN $((P,T,\varphi),\mu_0)$ can be regarded as an infinite-state transition system (N^k, T, δ, μ_0), where $\delta(\mu,t) = \mu'$ if $\mu \xrightarrow{t} \mu'$. A set $U \in N^k$ is said to be *upward-closed* if $\forall x \in U, y \geq x \Longrightarrow y \in U$.

In our subsequent discussion, we consider the forbidden state problem for Petri nets when the set of forbidden states is expressed in Presburger formula.

Theorem 5. *State-feedback and event-feedback NFSP for Petri nets are* co-r.e.-*complete.*

Proof. (Sketch) To show this result, we reduce the halting problem of deterministic 2-counter machines (2-CMs) to the NFSP of Petri nets.

A 2-CM $\mathcal{M}$ is a finite-state machine equipped with 2 counters c_1 and c_2 on which instructions of the following forms can be applied ($i = 1$ or 2)

(Type 1) s: $c_i := c_i + 1$; goto s'
(Type 2) s: if $c_i = 0$ then (goto s') else ($c_i := c_i - 1$; goto s'')

The blank-tape halting problem for 2-CMs is that of given a 2-CM starting with a blank-tape, deciding whether the machine halts or not. As 2-CMs are Turing-equivalent, the halting problem for 2-CMs is undecidable.

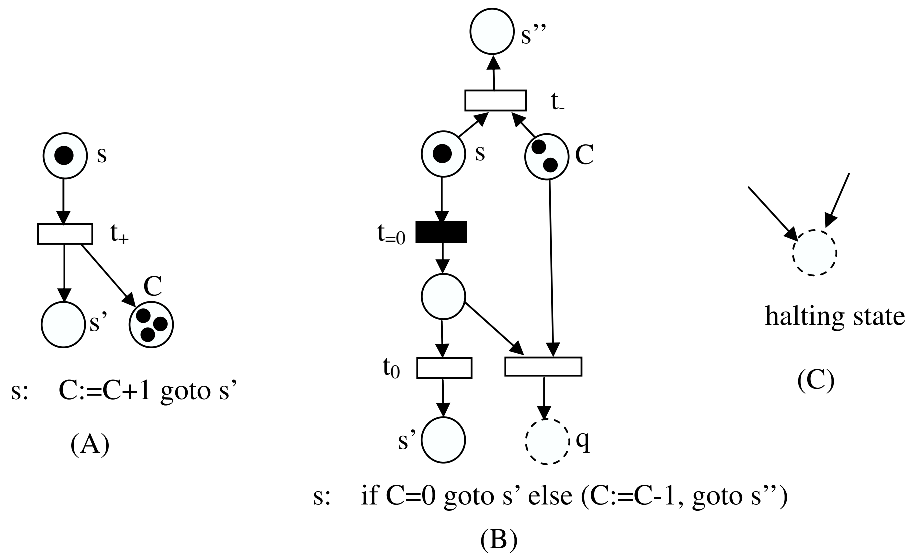


Fig. 5. Simulating a 2-CM.

In what follows, we show how to construct a PN $\mathcal{P}$ (along with a set $\mathcal{C}$ of control actions and a set Q of forbidden states) from a given 2-CM M in such a way that M does not halt on the blank tape iff there exists a non-blocking control policy under which $\mathcal{P}$ always avoids Q. Simulating Type 1 instructions is rather straightforward (see Figure 5(A)). For Type 2 instructions, one needs to know whether the counter is zero or not, and then act accordingly. It is well-known that Petri nets lack test-for-zero capabilities, making them strictly less powerful than Turning machines. With the presence of a non-blocking control policy, we are able to enforce zero-testing implicitly in a way shown in Figure 5(B). In Figure 5(B), $t_{=0}$ is a controllable transition, while the remaining transitions are uncontrollable. A forbidden state corresponds to one of the following two cases: a token is deposited in q or a token is added to the place simulating the halting state of M (see Figure 5(C)). A legitimate simulation of M corresponds to a non-blocking control policy under which $t_{=0}$ is turned off (resp., on) every time a Type 2 instruction is carried out on a non-zero (resp., zero) counter. It should be noted that being non-blocking requires that $t_{=0}$ be turned on when t_{-} is disabled (i.e., counter C is empty), ensuring that the simulation always proceeds if possible. It is then reasonably easy to see that $\mathcal{P}$ has an infinite computation under a non-blocking policy iff M is non-halting. The lower bound follows. To show the upper bound, Procedure NFS in Table 3 (in the proof of Theorem 1) provides a semi-decision procedure for checking whether from a marking the computation eventually reaches a forbidden state under all non-blocking policies. For Petri nets, Procedure NFS, which generates $U_0, U_1, ...$ may not terminate in general. $\square$

Theorem 6. *State-feedback and event-feedback FFSP for Petri nets are Σ_1^1-hard (i.e., hard for the first level of the analytical hierarchy).*

Proof. (Sketch) It was shown in [3] that for Petri nets, the *fair nontermination problem* (NTP^{fair}, i.e., the problem of, given a Petri net, determining whether there exists an infinite computation along which if a transition is enabled infinitely often, then the transition must occur infinitely many times) is Σ_1^1-hard. In what follows, we show how to reduce NTP^{fair} to $FFSP$, thus yielding Σ_1^1-hardness for $FFSP$.

Let Petri net $\mathcal{P}$ be an instance of NTP^{fair}. An instance $\mathcal{P}'$ of $FFSP$ (along with a set of forbidden states) is constructed to simulate $\mathcal{P}$ in such a way that $\mathcal{P}$ has a fair nonterminating computation iff $\mathcal{P}'$ can be controlled under a fair policy to avoid reaching a forbidden state. $\mathcal{P}'$ is constructed from $\mathcal{P}$ with the following modifications:

- $\mathcal{P}'$ contains all the places and transitions of $\mathcal{P}$,
- a new place p and a new transition t with $\varphi(p,t) = 1$ are added; initially, p contains a token,
- for every transition t' in $\mathcal{P}$, an arc from t' to p is added (i.e., $\varphi(t',p) = 1$),
- all transitions in $\mathcal{P}'$ are controllable transitions, and the set of control actions is $\{\{r\} \mid r$ is a transition of $\mathcal{P}'\}$,
- a marking is a forbidden state is p is empty.

In $\mathcal{P}'$, the controller plays the role of nondeterministically "guessing" a nonterminating fair computation that witnesses a solution (if it exists) of NTP^{fair}. If there exists an infinite computation avoiding all the forbidden states under a fair controller, then such an infinite computation (ignoring the additional transition t) is clearly fair with respect to $\mathcal{P}$, and vice versa. Note that as the firing of each of the transitions in $\mathcal{P}$ deposits a token to the newly added place p, it is easy to schedule t to fire infinitely many times without making p empty. On the other hand, if $\mathcal{P}$ does not have an infinite fair computation, then no fair control policy can result in an infinite computation. Hence, a fair controller in this case only leads to finite computations eventually reaching dead markings; hence, p will eventually become empty as the non-blocking constraint requires t to fire when no other transitions are capable of firing. In this case, forbidden states become unavoidable. This completes the proof. □

Theorem 7. *State-feedback and event-feedback FSP for Petri nets are decidable if the set of forbidden states is upward-closed.*

Proof. (Sketch) It is rather obvious that if there exists a conventional control policy under which a system can always be guided to avoid reaching a state in the set Q of forbidden states, then the policy disabling all controllable transitions clearly serves as a witness for such a control policy. After removing all the controllable transitions in a Petri net, a backward reachability analysis (starting from the set Q) can easily generate exactly the set of states from which the controller has no way of avoiding Q. As Q being upward-closed and the underlying system being a Petri net, the backward reachability procedure always terminates. □

Acknowledgments: The author thanks the anonymous referees for their comments that improved the presentation of this paper.

References

1. L. Holloway, and B. Krogh, Controlled Petri Nets: a Tutorial Survey. in *Discrete Event Systems*, Lecture Notes in Control and Information Sciences 199, G. Cohen and J.-P. Quadrat (eds.), pp. 158-168, Springer-Verlag, 1994.
2. L. Holloway, A. Khare, Y. Gong, Computing Bounds for Forbidden State Reachability Functions for Controlled Petri Nets, *IEEE Trans. SMC-A*, 34(2), pp. 219-228, 2004.
3. R. Howell, L. Rosier and H. Yen, A Taxonomy of Fairness and Temporal Logic Problems for Petri Nets, *Theoret. Comput. Sci.*, Vol. 82, pp. 341-372, 1991.
4. N. Jones and W. Laaser. Complete Problems for Deterministic Polynomial Time. *Theoret. Comput. Sci.*, 3:105–117, 1977.
5. B. Krogh, J. Magott, and L. Holloway, On the Complexity of Forbidden State Problems for Controlled Marked Graphs, *Proc. 30th IEEE Conf. on Decision and Control*, 85-91 vol.1, 1991.
6. J. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, Englewood Cliffs, NJ, 1981.
7. P. Ramadge and W. Wonham. The Control of Discrete Event Systems. *Proc. IEEE*, 77(1):81–98, 1989.

Author Index

Workshop on

Tilings and Self-Assembly

7 July 2007, Turku, Finland

Editor:

Jarkko Kari

Workshop on

Tilings and Self-Assembly

July 7, 2007, Turku, Finland

PROGRAM

8:55	Opening
9:00-10:00	*Erik Winfree* (Invited) Algorithmic Self-Assembly of DNA: Theory and Experiment
Coffee	
10:30-11:30	*Matthew Cook* (Invited) Temporal Error Correction in Three Dimensional Asynchronous Lattices
11:30-11:50	*Florent Becker* Self-assembly for tilings of the whole plane
11:50-12:10	*Robert Brijder* and *Hendrik Jan Hoogeboom* Perfectly Quilted Rectangular Snake Tilings
12:10-12:30	*Ville Lukkarila* The 4-way deterministic tiling problem is undecidable
Lunch	
13:30-14:30	*Nadrian Seeman* (Invited) DNA: Not Merely the Secret of Life
14:30-15:30	*Reidun Twarock* (Invited) A Tiling Approach to the Structure and Assembly of Viruses
Coffee	
16:00-17:00	*Nataša Jonoska* (Invited) Computing by Self-assembly of Flexible Tiles
17:00-17:20	*Matteo Pradella* and *Stefano Crespi Reghizzi* SAT-TS: A SAT-based tool to recognize and complete pictures specified by tiling
17:20-17:40	*Christiane Bercoff* Generation of boundary words of rectilinear pseudo-parallelograms

Table of Contents

Algorithmic Self-Assembly of DNA: Theory and Experiment

Erik Winfree

California Institute of Technology
Pasadena, CA 91125, USA

Self-assembly is a fundamental process in the self-organization of biological as well as non-biological structures. Passive self-assembly of molecular units, being driven just by thermodynamic binding energies and the geometrical structure of the molecules, would seem to be the simplest case to study – but it can be remarkably complicated. In fact, in a model of generalized crystal growth abstracted as the self-assembly of Wang tiles, passive self-assembly can be shown to be Turing universal. This leads to a number of theoretical observations: complex shapes that have concise algorithmic descriptions can be self-assembled from a small number of parts; these self-assembled structures can perform error correction during growth and can self-heal after damage; and as a simple form of self-replication, algorithmic crystals could provide an abiological example of Darwinian evolution. In our lab, we have been working to demonstrate these principles experimentally, using molecular Wang tiles constructed from DNA. Several examples will be given.

Temporal Error Correction in Three Dimensional Asynchronous Lattices

Matthew Cook

Institute of Neuroinformatics
Winterthurerstrasse 190
8057 Zurich, Switzerland

Self assembled tilings are useful for creating engineered structures at a very small scale. One of the things we would like to eventually construct is computational structures computing on a lattice. At such a small scale we will be faced with error-prone components and in many settings a lack of global synchronization. This takes us to the realm of fault-tolerant asynchronous cellular automata, where the cells need to locally and actively maintain synchronicity. However, the mechanism for maintaining synchronicity will also be error prone, which can lead to interesting topological defects in the effective space-time sheet. In the search for ways to fix these defects, we find that going to the more complicated case of three dimensions actually makes solving the problem easier, and finally we can reduce the problem to a simple conjecture regarding the smoothness of a diagonally growing surface of a crystal lattice. This is joint work with Erik Winfree and Peter Gacs.

Self-assembly for tilings of the whole plane

Florent Becker

Laboratoire d'Informatique du Paralllisme
UMR 5668 CNRS, INRIA, Universit Lyon 1, NS Lyon
46 Alle d'Italie – 69364 Lyon Cedex 07 – France
Florent.Becker@ens-lyon.fr

Abstract. In this paper, we investigate the ability to use self-assembling system to tile $\mathbb{Z}^2$ with Wang tiles. We say that a self-assembling system is infallible when it covers the whole plane without mismatch (for any sequence of non-deterministic choices), and we look at the set of tiling of $\mathbb{Z}^2$ obtained as limits of its dynamics.
We prove that at temperature 1, only periodic tilings can be assembled, up to equivalence between tiles, but that at temperature 2, much more complex patterns can be assembled. We prove that it is even possible to self-assemble quasi-periodic patterns, namely the famous Robinson tiling. We conjecture that other substitution tilings can be assembled.

Self-assembling tilings are a model of spontaneous growth and organization of structures that was introduced by Winfree in [8], and related to self-organizing polymers of DNA. This very simple model is a refinement of Wang tilings, yet it was proved experimentally that some chemical systems can actually behave like this model[6]. Since then, the problem of assembling finite shapes has been thoroughly studied, and the assembly process for finite shapes starts to be well understood. Assembling a finite shape consists in waiting for the process of self-assembly to stop and say that the set of cells covered by tiles is the assembled shape.

Yet, despite the legacy of Wang tilings, the question of patterns that can be self-assemblied and which cover the whole plane has almost never been studied. That is, given a self-assembling system which does cover the whole plane, we will look at how complex the patterns that appear on the plane can be. In [7], Winfree gives an example of a non trivial pattern which can be self-assemblied: he shows a tile-set which covers a quarter-plane with a Sierpinsky triangle. In this paper, we will show that, as for finite shapes, the *temperature* of self assembly is a crucial parameter.

Definition of the problem

We introduce the notion of *infallible* self-assembly: a self-assembling system is infallible if it covers the whole plane $\mathbb{Z}^2$ without fail: any (finite) pattern can (and will) be extended into a tiling of the whole plane. Their non-determinism has been tamed: we are sure that the whole plane will eventually be covered, but the system still chooses a pattern in a given set, which can even be uncountable. This infallibility would be crucial for the practical applications of such assembly such as nanotechnologies.

At temperature 1, the only patterns that can be self-assembled are periodic tilings with a random (or non-deterministic) component. At temperature 2, we do not have a characterization of the patterns which can be self-assembled, but we show that we can assemble some quasi-periodic but non-periodic tilings. This is a very important step, as it shows that some tilings where local information can be determinant far away can be self-assembled.

Looking at patterns rather than shapes for self-assembly makes sense when studying a number of natural phenomena which take place on a surface rather than building it.

This problem is also intresting on the links between cellular automata and tilings: our self-assembling system can be seen as a cellular automaton. This CA starts from a single cell and spreads a tiling onto the whole plane.

A self-assembling system is *infallible* when any production can be extended to cover the whole plane. This actually means that given any "fair" system for choosing where the next tile will be attached, the system will eventually cover any position in $\mathbb{Z}^2$. We will look at the set of tilings of the plane generated by such systems.

We say that a self-assembling system assembles a set of tilings when, starting from the configuration with only the seed at $(0,0)$, it covers the whole plane with a tiling from $\mathcal{R}$, and if any tiling in $\mathcal{R}$ can be assembled in this way. The function i is an interpretation functions which maps several tiles of $T_{\mathcal{A}}$ to a tiles of Σ: it is often useful to have several tiles in $T_{\mathcal{A}}$ representing the same tile in Σ. This means that the tiles of $T_{\mathcal{A}}$ have a visible part, which is given by i, and a hidden part, which we will use to compute the tiling.

1 Temperature 1

We will now look at temperature 1 self-assembling systems. This class is especially interesting for several reasons. First, it is a natural model in the domain of self-assembling tiling, and also the easiest to implement, for example with chemical means [8,6]. This case also corresponds to the simplest algorithms for answering the question "what tile can be put at (x,y) to extend this finite pattern p": to answer this question, one simply needs to build p, and then, following a path from p to (x,y), put the tiles one after the other.

We will show that this class of tilesets corresponds to tilesets which up to an equivalence relationship between the tiles, are periodic.

The equivalence states that two tiles are equivalent when each can replace the other in a tiling, and the consequences of this replacement don't spread further than one tile:

Definition 1 *Let $\mathcal{R}$ be a set of tilings, two tiles $t_1, t_2 \in \Sigma$ are equivalent ($t_1 \cong t_2$) if, for any $T \in \mathcal{R}$ such that $T(0,0) = t_1$, t' defined by $T'(0,0) = t_2$ and $T'(x,y) = T(x,y)$ when $(x,y) \notin \{(x-1,y)(x+1,y)(x,y-1)(x,y+1)\}$ is in $\mathcal{R}$.*

We extend $\cong$ to tilings as follows: $T \cong T'$ if and only if $\forall (x,y), T(x,y) \cong T'(x,y)$. Thus, to any tiling $T \in \mathcal{R}$, we can associate a tiling in $\mathcal{R}/\cong$, that a is a $\tilde{T} : \mathbb{Z}^2 \to \Sigma/\cong$ defined by: $\tilde{T}(x,y)$ is the equivalence class of $T(x,y)$ modulo $\cong$.

With this notion of equivalence, we can characterize the tilings which can be assembled a temperature 1 self-assembly system.

Theorem 1 *Let $\mathcal{R}$ be a set of tilings which is assembled by a temperature 1 self-assembling system, then $\mathcal{R}/\cong$ consists of a unique equivalence class, which is a periodic tiling of the plane by $\Sigma/\cong$.*

Lemma 1 *Let $\mathcal{R}$ be a set of tilings which is assembled by a temperature 1 deterministic self-assembly system $\mathcal{A}$, $\mathcal{R}$ is reduced to a single element which is periodic.*

Sketch of the proof for the lemma We will build a finite automaton a from $\mathcal{A}$, which will be able to recognize the tiling. As this automaton will be deterministic, the tiling in $\mathcal{R}$ will be unique; as the automaton is finite, the tiling will be periodic.

We will now give the proof of the theorem 1

Proof. Without the hypothesis of determinacy, one can still build a, but as a non-deterministic finite automaton. What we will show is that when we determinize this automaton, we get an automaton whose states are the equivalence classes for $\cong$.

We will build $\bar{a}$, the determinized version of a, using the usual construction. The state $\bar{a}(w)$ is the set of states a can be in after reading w.

We then prove that the states of $\bar{a}$ are exactly the equivalence classes for $\cong$. The quotient of $\mathcal{R}$ by the states of $\bar{a}$ is reduced to a unique periodic tiling, since a is a finite automaton. So $\mathcal{R}/\cong$ is reduced to a unique periodic tiling.

2 Temperature 2 (and beyond)

In [5], Winfree showed that it was possible to self-assemble Sierpinsky's triangle with a temperature 2 tile-set. In fact, the construction can be generalized, and a quarter-plane can be covered with the space-time diagram of any 1-dimensional CA running on a particular set of configurations. This technique was also used in [2]. These configurations can even be made non-deteministic, as well as the CA.

With this construction, a number of patterns, especially self-similar patterns, which appear in the space-time diagrams of cellular automata. One of the most famous examples is Sierpinsky's triangle, which appears as the space-time diagram of the XOR cellular automaton.

Assembling space-time diagrams of 1-dimensional CA does give a wide variety patterns, but these patterns have a strong directionality: the seed plays a central part, and can be seen as the origin of the pattern. It is in fact possible to build much more regular patterns that are not periodic. An interesting properties of tilings is quasi-periodicity.

A tiling T is quasi-periodic if for any pattern of size $n \times n$ that appears in T appears in every window of size $w \times w$ of T for some w.

In [3], Berger showed that for some sets of Wang tiles, there was no possible periodic tiling, and that all their tilings were quasi-periodic. Robinson improved the result by introducing with a much smaller tile-set with that property. This tile-set was used to prove that the tiling problem for Wang tile-sets is undecidable. It also features a strong self-similarity and a hierarchical structure that we are going to use in our construction.

The patterns they can tile the plane with are shown on figure 1. The proof that they are the only possible tilings and that they are quasi-periodic can be found in [1], an annex to [4]. The vertices of the squares of rank $(n + 1)$ are located at the center of the squares of size n. Each of these n-squares is contained in a n-frame, as shown on the figure. This n-frame is the set of tiles which "depend" on that frame.

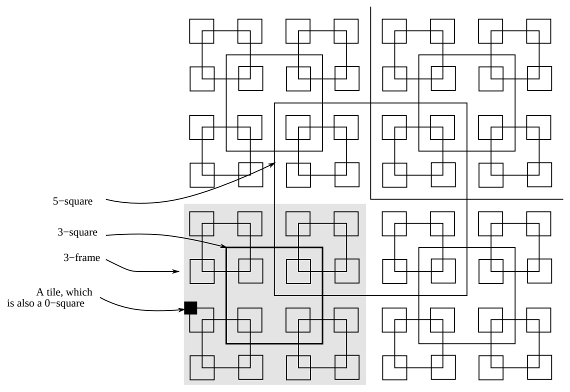


Fig. 1. Robinson tiling

Theorem 2 *There is a self-assembling system S at temperature 2 which assembles the set of Robinson's tilings without infinite line.*

The self-assembly system we are going to use is based on a duplication of each of the tiles and each of the colors of Robinson's tiling, by adding to them some additional informations about what is going on in the construction.

The assembhly of the tiling is going to be made recursively, by building in turn all the n-frames containing $(0, 0)$. For this, we need a self-assembling system which, given a n-frame, is able to build any of the $n + 1$-squares that can contain it. It will use some extra bits of informations put in the colors appearing on the border of the n-frame, and will give the same informations on its own border. We will also show that if c is a configuration containing a n-frame fr then, $\forall c' > c$, $\exists c'' > c'$ which contains a $n + 1$ square containing fr. Then, by proving that we are able to build a first square with the relevant extra bits on its borders, and by induction we will get that S assembles the set of Robinson's tilings.

This result seems not to be too tightly linked to the particular innings of Robinson tiling, and can surely be extended into a general scheme for all substitution tilings.

References

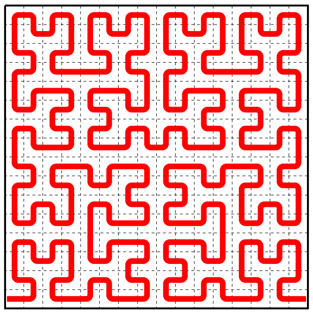
1. Cyril Allauzen and Bruno Durand. Tiling problems.
2. Florent Becker, Ivan Rapaport, and Eric Rémila. Self-assemblying classes of shapes with a minimum number of tiles, and in optimal time. In *FSTTCS*, pages 45–56, 2006.
3. R. Berger. The undecidability of the domino problem. *Memoirs of the american mathematical society*, 1966.
4. Y. Gurevich and E. Gradel. *Classical decision problems*. Springer-Verlag, 1997.

5. Erik Winfree. Paul W.K. Rothemund, Nick Papadakis. Algorithmic self-assembly of dna sierpinski triangles. *PLoS Biology*, 2004.
6. Paul W.K. Rothemund. *Theory and Experiments in Algorithmic Self-Assembly*. PhD thesis, University of Southern California, 2001.
7. Erik Winfree. Simulations of computing by self-assembly.
8. Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, Caltech, 1998.

Perfectly Quilted Rectangular Snake Tilings

Robert Brijder and Hendrik Jan Hoogeboom

LIACS, Leiden University, The Netherlands



1 Introduction

We consider languages consisting of pictures [6], which here are rectangular arrays of symbols, assembled from unit-sized squares carrying one symbol each. The squares should be fitting together by their edges, like in a jig-saw puzzle. However, one abstracts from the familiar curved form of the edges by considering marked edges, requiring that adjacent squares in the assembled rectangle have matching markings on their common edge.

Inspiration for this research was the well-known Hilbert curve as represented in [9], depicted, slightly simplified, above. An old decidability problem on the formation of infinite snake tilings could be solved [1] using an intricate tiling system devised earlier by Kari in the context of cellular automata. The system is built from the usual four-sided tiles, but it is applied in the context of the theory to *snake tilings*, where the tiles are connected as a meandering ribbon, each tile matching only two adjacent tiles (rather than four). So, although the picture of the Hilbert tiling looks like a snake, a continuous line visiting each unit of the resulting square once, this is not the underlying (four-sided) tiling.

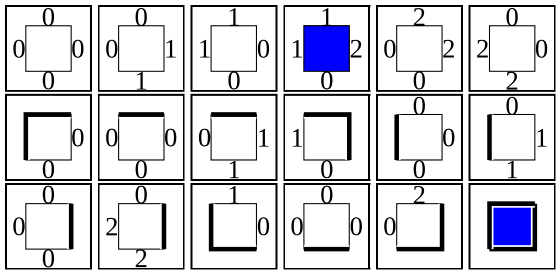
Here we take the mental image of a snake that covers all positions of a rectangle as our definition. Snake tilings basically are one-dimensional descriptions of pictures. Each such description consists of a single string built from the labels of the squares visited and the directions (up, right, down, left) in which the description continues. They can be 'programmed' very much like automata walking over the grid, passing through positions of the picture, and checking their contents. Such a 'program' then is a regular language, states of an automaton corresponding to the markings on the tiles.

We compare the descriptional power of regular snake tilings with the classic notion of finite tiling systems. We will show that, up to a multiplication factor, the requirement that the snake covers all positions of a rectangle exactly once makes them equivalent to tilings with four-sided tiles.

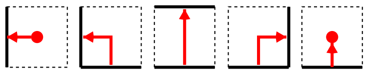
2 Definitions

Let $[m] = \{1, \ldots, m\}$. A *tiling* system is a finite set of unit squares T each having marked edges, together with a designated marking c. A T-*tiling* of a $m \times n$ rectangle is a mapping from $[m] \times [n]$ into T such that adjacent tiles share the marking of their common edge, and such that the outer border of the rectangle is completely marked by c. The latter requirement enables us to fix some properties of the border tiles. Given a mapping $\varphi : T \to \Sigma$, the *picture language* defined by (Σ, T, φ, c) is the set of Σ-labelled rectangles that admit a tiling, i.e., $\{\varphi \circ t \mid t$ is T tiling of a rectangle$\}$.

Example. Consider the following tiling system consisting of 18 tiles. The edges are marked by $\{\ 0, 1, 2, c\ \}$ as indicated in the picture, where the special border marking c is replaced by a thick line. These tiles are forced to form a square of odd side length, with the blue tile taking the middle position. The mapping φ 'removes' the edge markings, leaving squares with a single coloured unit square in the middle.



A T-*snake* on a $m \times n$ rectangle is a pair of mappings σ, τ from $[k]$ into $[m] \times [n]$ and from $[k]$ into T respectively, such that σ is injective, consecutive positions $\sigma(i)$ and $\sigma(i + 1)$ are adjacent in the rectangle, the assigned tiles $\tau(i)$ and $\tau(i + 1)$ share the marking at their common edge, and the borders of the rectangle are marked by c. Tiles in the snake may touch at other places, but there we do not force matching markings (sometimes this is called a *weak* snake). A snake is called *perfectly quilted* if σ is bijective (so $k = mn$). Given a mapping φ as above, a perfectly quilted T-snake defines a picture over Σ.



In order to have linear descriptions of snakes (and consequently linear descriptions of pictures) we add directions to the sequence τ of tiles, or as we do here, we consider *directed tiles* (cf. the picture above and its rotations) which include initial and final tiles. Now a snake is a string of (directed) tiles, and we require that successive tiles match in their directions and the corresponding edge markings. This is a local, hence regular, condition on the strings. As mentioned above we require that the outer border of the rectangle is marked by the special symbol c. ([1])

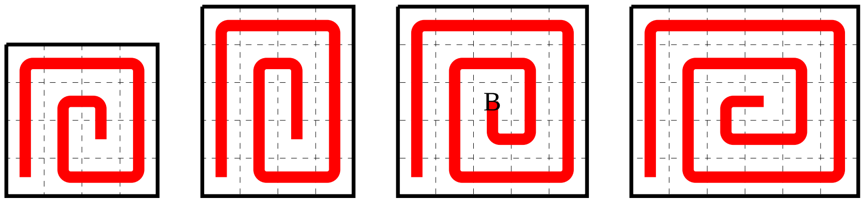
Given a regular language of snakes over a (directed) tiling system T, and a mapping $\varphi : T \to \Sigma$, the perfectly quilted snakes from the language define a picture language

[1] This natural requirement seems not necessary in the arguments of the present paper.

over Σ. Such a language will be called a *perfectly quilted rectangular snake* language, or PQRS picture language for short.

Example. Consider the picture language of all odd sized squares over $\{B, W\}$ such that the middle position is labelled by B. A finite state automaton moving over a grid would accept these by first checking that the input is an odd sized square by moving along the diagonal from the SW-corner to the NE-corner. Then the middle position is (non-deterministically!) checked by moving from the NE-corner to the middle position, verifying there is a B at this position, then making a turn to the SE-corner. If this succeeds, the turning point was the middle position. This is easily transferred to four-sided tiles, cf. the previous example. Note this picture language cannot be accepted by *deterministic* automata walking over grids [2].

Snakes cannot visit the same position twice. However, using perfect quilting it is possible to determine the center by spiraling inwards. In the case of an odd sized square this walk stops at the first step after a turn to the north.



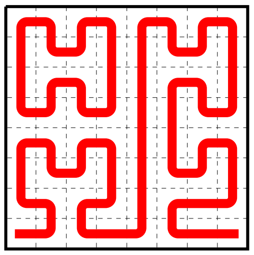
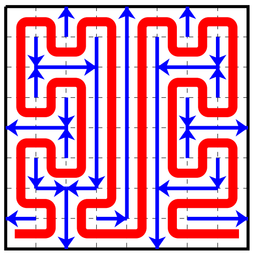
3 Tilings for Snakes

We show that every PQRS picture language can be defined by a four-sided tiling system, but first we argue that this is not immediate. As before we assume the tiles are directed, each tile is superimposed with an arrow indicating the direction of the snake before and after the present tile. This direction can be encoded in the marking of the tile. We eliminate the markings from the other two edges, replacing them by a 'blank' marking (but we keep the special marking for the border of the rectangle). In this way a snake tiling is also a four-sided tiling: the markings on all shared edges match.

Unfortunately not every two-sided tiling of a rectangle with matching arrows defines a perfectly quilting snake for the rectangle. Tiles may form separate circular components, cf. the picture below. We have to force that the arrows form a single path. Recall that picture languages defined by tiling systems are characterized using existential monadic second-order logic [7]. However, the obvious way to define connectedness in monadic second-order logic uses universal quantification over sets of positions in the rectangle. (First define the property of a 'closed' set of tiles, i.e., that it contains for each of its tiles its successor along the snake. To define a path along the snake one additionally requires that the closed set is 'minimal', i.e., there is no smaller set which is also closed.) This is not allowed in the existential part of the logic. At a first glance this makes our task impossible.

Surprisingly, Reinhardt [12] shows that the $\{a, b\}$-labelled pictures in which the b's form a connected area can be defined using tiles, which means that connectedness *can* be defined for pictures without universal quantification. The trick is to ‘grow’ two sets of trees. One set covers the area of b’s, the other set of ‘tentacles’ connects (the bottom-right corner of) each square to one of the outer walls. Separately, neither the b-tree nor the tentacle trees can be guaranteed to be without cycles, but together the one forces the other to be a tree. This method can also be used to define a connected snake, illustrated in the right figure below.

Lemma. Every PQRS picture language can be generated by a (four-sided) tiling system.

4 Undecidability[2]

It is a classical result that emptiness is undecidable for picture languages defined by tiling systems: tiles can be set-up to simulate the computations of a Turing machine (TM). For snakes connectivity problems were investigated [4]; it is undecidable for a given tiling system and two positions if there exists a snake from one position to the other (when the first tile is specified).

The reader will not be surprised that also emptiness for PQRS tilings is undecidable. Again TM computations can be simulated, using mainly the *form* of the snake, and not the labels of the tiles it passes. For this we use techniques like those used for chain code picture languages [10], where strings define simple drawings consisting of line segments.

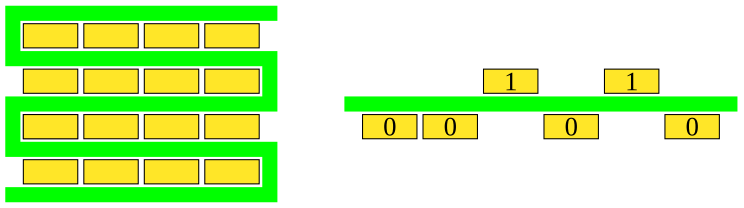
A *chain code picture language* is defined by a (regular) language over the alphabet $D = \{r, d, l, u\}$. These symbols are interpreted as instructions for a plotter which at each step draws a unit segment in the designated direction right, down, left, or up. For instance, d^2urud^2 results in the figure H, when we start in the NW-corner. Clearly snakes and chain code pictures are related, although there are several technical differences. For instance, a segment of a chain code picture may be traced twice (as in the H).

Although emptiness of chain code picture languages is trivially decidable (every string over D defines a picture), many properties of chain code picture languages are undecidable. For instance, whether the language contains a picture without subfigure

[2] The result presented here alternatively follows from the next section. We prefer to keep the construction because of its simplicity. It also shows the deep connections between snake tilings and chain code picture languages.

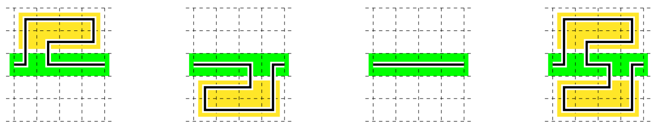
H [3], or whether it contains a self-avoiding picture [13] (interesting in the context of snakes!).

We construct a set of directed tiles which code a computation of a TM if they define a continuous line visiting every position in a rectangular area. The line basically zig-zags, as the green meander below. The horizontal segments code the consecutive configurations of the TM as binary sequences. To code a 0 the snake makes a detour filling a block under the line, for a 1 it fills a block above.



Without further instructions all horizontal segments follow the same pattern of blocks above or under the horizontal line in order to make a perfect quilting pattern (every position covered exactly once).

To accomodate local transformations of the configuration during a computation (as the result of a write action of the TM, or a change of state) we show how to change a 1 into a 0. If a segment of the snake is forced to skip a block (neither traversing the one above nor the one below), this can only yield a perfect quilting if the segment below codes a 1 and the segment above codes a 0 at that position, so both blocks are covered. The basic patterns are shown below: the coding of 1, 0, $1{\longrightarrow}0$, and $0{\longrightarrow}1$ as segments of a snake.



Lemma. It is undecidable whether a directed tiling system defines a nonempty PQRS picture language.

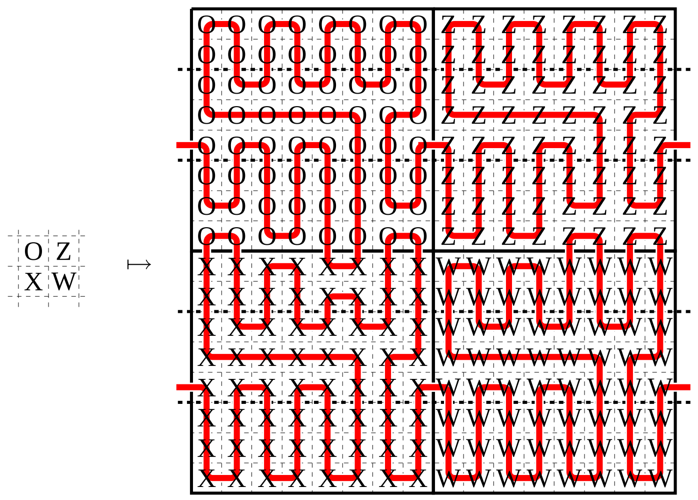
The only surprise in this proof is its complete simplicity. No complicated case analysis is needed, as in [13]. The perfect quilted form of the line simply forces a proper computation. In the construction tiles are 'programmed' to form regular patterns that can copy parts of the curve below, or locally change the configuration to model TM instructions.

5 Weaving Snakes

The tiles defined by snakes are only required to fit along the snake: two edges for each tile. The connectivity of classic tiling systems is greater than that of snakes as they have to fit in four directions. Still we show that every four-sided tiling can be defined using PQRS tilings provided we cheat a little, using tiles larger than unit size. Super-tiles (as explained in [9]) make it possible to code edge colours by humps and bumps. Like in the undecidability construction the snake follows a basic meander, where consecutive rows match according to form (rather than edge marking).

$$\begin{array}{|ccc|}\hline \mathrm{B}&\mathrm{W}&\mathrm{B}\\ \mathrm{B}&\mathrm{W}&\mathrm{W}\\\hline\end{array} \mapsto \begin{array}{|cccccc|}\hline \mathrm{B}&\mathrm{B}&\mathrm{W}&\mathrm{W}&\mathrm{B}&\mathrm{B}\\ \mathrm{B}&\mathrm{B}&\mathrm{W}&\mathrm{W}&\mathrm{B}&\mathrm{B}\\ \mathrm{B}&\mathrm{B}&\mathrm{W}&\mathrm{W}&\mathrm{W}&\mathrm{W}\\ \mathrm{B}&\mathrm{B}&\mathrm{W}&\mathrm{W}&\mathrm{W}&\mathrm{W}\\\hline\end{array}$$

The k-multiplication of the $m \times n$ picture p is the $km \times kn$ picture p' such that the label $p'(i,j)$ equals $p(i/k, j/k)$, where $/$ denotes integer division. The operation is extended to languages. Consider now the k-multiplication of a picture language L defined by a tiling system (for a suitable value k to be determined).



Like for the simulation of a TM in the previous section the main route of the snake is a meander following the super-tiles. This allows to check that the super-tiles fit in one direction. The markings of the edges in the other direction is left as a trail of humbs and bumps. The picture above shows two adjacent 8×8 super-tiles.

All positions in a super-tile have the same label (colour). While meandering along the edge the (small) tiles have to match the current label, but sometimes that of the neighbouring super-tile, when the edge is crossed. This means the form of the trail also codes for the label of the neighbouring tiles. We need a little math here. In a $2k$-size super-tile (with $k \geq 2$) we can code k bumps, with (at least) 2^k possibilities. With c edge markings and t tile labels, the trail should distinguish $c \cdot t^2$ possibilities, which can be obtained by choosing an appropriate k. Note that $4k^2$ unique tiles are necessary to program the walk in such a supertile.

Lemma. For every picture language defined by a four-sided tiling systems there exists a constant k such that its k-multiplication is a PQRS picture language.

Another popular way to define sets of pictures are *substitution tilings*. In general these can be used to obtain aperiodic tilings of the plane by geometric forms, like the Penrose tilings. Here we consider the two-dimensional version of string morphisms, and replace each square in a rectangle by an $k \times \ell$ rectangle which is determined by the label of the original square. In this way a rectangle is changed into a larger rectangle. This process can be iterated, starting from a simple rectangle, to obtain a picture language. (In this way one may generate pleasing fractal pictures, see e.g., Chapter 5.4 of [14] or Chapter 5 in [5].) In the illustration below each symbol is replaced by a particular 2×2 square.

It follows from the work of Mozes [11] that for each substitution tiling has an equivalent four-sided tiling system with 'matching rules', like the one explicitly constructed by Kari.

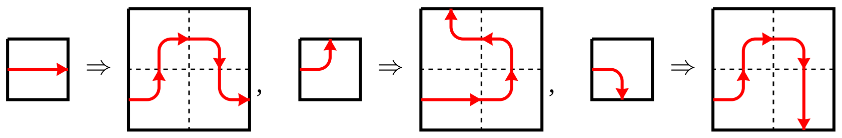
$$\boxed{\mathrm{A}} \Rightarrow \boxed{\begin{array}{cc} \mathrm{A} & \mathrm{C} \\ \mathrm{B} & \mathrm{A} \end{array}} \Rightarrow \boxed{\begin{array}{cccc} \mathrm{A} & \mathrm{C} & \mathrm{D} & \mathrm{C} \\ \mathrm{B} & \mathrm{A} & \mathrm{C} & \mathrm{A} \\ \mathrm{A} & \mathrm{B} & \mathrm{A} & \mathrm{C} \\ \mathrm{B} & \mathrm{D} & \mathrm{B} & \mathrm{A} \end{array}} \Rightarrow \boxed{\begin{array}{cccccccc} \mathrm{A} & \mathrm{C} & \mathrm{D} & \mathrm{C} & \mathrm{D} & \mathrm{C} & \mathrm{D} & \mathrm{C} \\ \mathrm{B} & \mathrm{A} & \mathrm{C} & \mathrm{A} & \mathrm{B} & \mathrm{D} & \mathrm{C} & \mathrm{A} \\ \mathrm{A} & \mathrm{B} & \mathrm{A} & \mathrm{C} & \mathrm{D} & \mathrm{C} & \mathrm{A} & \mathrm{C} \\ \mathrm{B} & \mathrm{D} & \mathrm{B} & \mathrm{A} & \mathrm{C} & \mathrm{A} & \mathrm{B} & \mathrm{A} \\ \mathrm{A} & \mathrm{C} & \mathrm{A} & \mathrm{B} & \mathrm{A} & \mathrm{C} & \mathrm{D} & \mathrm{C} \\ \mathrm{B} & \mathrm{A} & \mathrm{B} & \mathrm{D} & \mathrm{B} & \mathrm{A} & \mathrm{C} & \mathrm{A} \\ \mathrm{A} & \mathrm{B} & \mathrm{D} & \mathrm{C} & \mathrm{A} & \mathrm{B} & \mathrm{A} & \mathrm{C} \\ \mathrm{B} & \mathrm{D} & \mathrm{B} & \mathrm{D} & \mathrm{B} & \mathrm{D} & \mathrm{B} & \mathrm{A} \end{array}}$$

The above construction needs for a multiplication factor k which is large enough to code enough edge types. Now that we consider a substitution h by 2×2 squares, and consider a tiling T defined by h. We may take k of the form 2^ℓ. For a symbol σ let $h^\ell(\sigma)$ be the $2^\ell \times 2^\ell$ tiling which results from ℓ times iterating substitution h starting with a single square labelled by σ. Now in the 2^ℓ-multiplication T' of T we replace each $2^\ell \times 2^\ell$ super-tile labelled by symbol σ by the $2^\ell \times 2^\ell$ tiling $h^\ell(\sigma)$. In this way we obtain a tiling which again is defined by the substitution h. Returning to the snake construction for tiling systems, instead of checking a single label in every position of the super-tile the snake may check the appropriate position of such a super-tile $h^\ell(\sigma)$ (as we can build that image into the 'finite state' the snake tiles).

Lemma. Every picture language defined by a substitution tiling is a PQRS picture language.

6 Back to Hilbert

Also the Hilbert curve we started with can be obtained from a single tile by iterating a 2-dimensional substitution. One needs twelve tiles and rules as in the picture given below, a mapping already implicit in the definition of Hilbert [8].

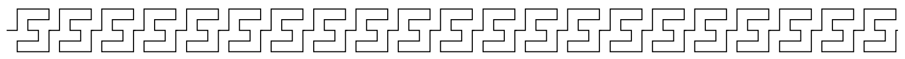


So we have the rather amusing conclusion that the snake-like Hilbert pictures can be defined using a PQRS tiling. This in a sense is a regular one-dimensional description (with additional perfect quilting requirement). However, the PQRS snake tiles do not follow the natural 'flow' of the Hilbert curve! The question that remains is whether the

Hilbert curves can be defined as chain codes (with perfect quilting), so that the curve ‘follows’ its description.

References

1. L.M. Adleman, J. Kari, L. Kari, and D. Reishus: On the Decidability of Self-Assembly of Infinite Ribbons. Proceedings 43rd FOCS (2002) 530–537.
2. M. Blum and C. Hewitt: Automata on a 2-Dimensional Tape. Proceedings 8th SWAT (1967) 155–160.
3. J. Dassow and F. Hinz: Decision Problems and Regular Chain Code Picture Languages. Discrete Appl. Math. 45 (1993) 29–49.
4. Y. Etzion-Petruschka, D. Harel, and D. Myers: On the Solvability of Domino Snake Problems. Theor. Comput. Sci. 131 (1994) 243–269.
5. F. Drewes: *Grammatical Picture Generation*. Springer, 2006.
6. D. Giammarresi, and A. Restivo: Two-Dimensional Languages. Handbook of Formal Languages, Vol. 3 (1997) 215–267.
7. D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas: Monadic Second-Order Logic Over Rectangular Pictures and Recognizability by Tiling Systems. Inf. Comput. 125 (1996) 32–45.
8. D. Hilbert: Ueber die stetige Abbildung einer Linie auf ein Flächenstück. Mathematische Annalen 38 (1891) 459–460.
9. J. Kari: Infinite Snake Tiling Problems. Proceedings Developments in Language Theory 2002, Lecture Notes in Computer Science 2450 (2003) 67–77.
10. H.A. Maurer, G. Rozenberg, and E. Welzl: Using String Languages to Describe Picture Languages. Information and Control 54 (1982) 155–185.
11. S. Mozes: Tilings, Substitution Systems and Dynamical Systems Generated by Them. Journal d’Analyse Mathématique 53 (1989) 139–186.
12. K. Reinhardt: On Some Recognizable Picture-Languages. Proceedings MFCS 1998, Lecture Notes in Computer Science 1450 (1998) 760-770.
13. D. Robilliard and D. Simplot: Undecidability of Existential Properties in Picture Languages. Theoret. Comput. Sci. 233 (2000) 51–74.
14. S. Wolfram: *A New Kind of Science*. Wolfram Media, 2002. http://www.wolframscience.com



The 4-way deterministic tiling problem is undecidable

Ville Lukkarila [⋆]

Department of Mathematics
University of Turku
FIN-20014 Turku
Finland
vinilu@utu.fi

Abstract. It is shown that the tiling problem remains undecidable even when the instances are 4-way deterministic tile sets, i.e. the colors adjacent to any one corner determine the tile within the tile set uniquely.

1 Introduction

A *Wang tile* (or a *tile* in short) is a unit square with colored edges. The edges of a Wang tile are called *north*, *east*, *west* and *south* edges in a natural way. A Wang tile t can be considered also as an ordered 4-tuple $t = (N_t, E_t, W_t, S_t)$ containing the colors in a predefined order. For the given tile t, expressions N_t, E_t, W_t and S_t are used to denote north, east, west and south side colors, respectively. A *Wang tile set* T (or a *tile set* in short) is a finite set containing Wang tiles. A *tiling* is a mapping $f : \mathbb{Z}^2 \to T$, which assigns a unique Wang tile for each integer pair of the plane. A tiling f is said to be *valid* if for every pair $(x, y) \in \mathbb{Z}^2$ the tile $f(x, y) \in T$ matches its neighboring tiles (e.g. the south side of tile $f(x, y)$ has the same color as the north side of tile $f(x, y - 1)$).

A Wang tile set T is said to be *NW-deterministic*, if within the tile set there does not exist two different tiles with the same colors on the north- and west-sides. In general, a Wang tile set is *XY-deterministic*, if the colors of X- and Y-sides uniquely determine a tile in the given Wang tile set. A Wang tile set is *4-way deterministic*, if it is NE-, NW-, SE- and SW-deterministic.

A mapping $f : T_1 \to T_2$ is called a *tile homomorphism* if it respects the colors, i.e. $f(t) = t'$ with $N_{t'} = g(N_t)$, $E_{t'} = g(E_t)$, $W_{t'} = g(W_t)$ and $S_{t'} = g(S_t)$, where g is a mapping from the set of the colors of the tile set T_1 to the set of the colors of the tile set T_2. The *homomorphic image* $f(T)$ of a tile set T is defined in the natural way as the set

$$f(T) = \{f(t) | t \in T\} .$$

A tiling $f : \mathbb{Z}^2 \to T$ is called *periodic* with period (a, b) if $f(x, y) = f(x+a, y+b)$ for all $(x, y) \in \mathbb{Z}^2$ and $(a, b) \neq (0, 0)$. A tile set T is called *aperiodic*, if there exists some tiling with the tile set T, but no tiling with the tile set T is periodic. If the tile set T admits a periodic tiling $f : \mathbb{Z}^2 \to T$ with some period, then it admits also a *doubly periodic* tiling $g : \mathbb{Z}^2 \to T$, that is, there exists such non-zero integers a and b that $g(x, y) = g(x + a, y)$ and $g(x, y) = g(x, y + b)$ for all $(x, y) \in \mathbb{Z}^2$ [1].

⋆ This research has been supported by the Finnish Cultural Foundation.

The following question is referred to as the *tiling problem*: “Given a Wang tile set T, does there exist a valid tiling of the plane?” A tiling $f : \mathbb{Z}^2 \to T$ is said to *contain* a tile $t \in T$, if for some integers $x, y \in \mathbb{Z}$ equation $f(x, y) = t$ holds. The following question is referred to as the *tiling problem with a seed tile*: “Given a Wang tile set T and a tile $t \in T$, does there exist a valid tiling of the plane that contains the tile t?” If the tiling problem with a seed tile was decidable, then the tiling problem would be decidable. Let T be the tile set of the given instance of the tiling problem. Then the answer for the tiling problem is affirmative if, and only if, for some tile $t \in T$ the answer for the tiling problem with a seed tile is affirmative considering the tile set T as the tile set of the instance and the tile t as the seed tile of the instance.

2 The 4-way deterministic tiling problem with a seed tile

It is already known, that the tiling problem is undecidable [1,2]. The tiling problem is known to be undecidable even when restricted to tile sets that are deterministic by one corner [3]. It can be shown, that the tiling problem remains undecidable when restricted to tile sets that are deterministic by two opposite corners [4].

In what follows, it is shown that the tiling problem is undecidable for tile sets that are deterministic by all four corners. The proof is similar to Robinson’s original proof [1], but relies on the 4-way deterministic aperiodic tile set given by Kari and Papasoglu [5].

2.1 Turing machines

Here a Turing machine $\mathcal{M}$ is considered to be a four-tuple $\mathcal{M} = (S, T, \delta, q_0)$, where S is the state set, T is the tape alphabet, δ is the transition function and $q_0 \in S$ is the initial state. No “accept”, “reject” or “halt” states are defined explicitly. The tape of a Turing machine is defined to be two-way infinite and symbol ε is used to denote the empty symbol of the Turing machine. The transition function is a mapping

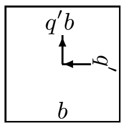
$$\delta : S \times T \to S \times T \times \{L, R\},$$

that is, at every time step the read-write head moves either to the left or to the right. A Turing machine is said to *halt*, if it is in state q reading symbol s and $\delta(q, s)$ in undefined. Transition of the form $\delta(x, y) = (a, b, c)$ can also be written in the form $(x, y) \to (a, b, c)$. The *Turing machine halting problem* is considered to be the following question: “Does the given Turing machine $\mathcal{M}$ halt when started on an empty tape?” The halting problem is known to be undecidable.

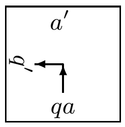
A Turing machine can be represented with a Wang tile set as it has been shown by Robinson [1]. The following tile set is almost the same as the original tile set given by Robinson:

Action tile and merging tiles for a left move Assume that the Turing machine contains move $(q, a) \to (q', a', L)$. Then the tiles in figure 1 are added to the tile set.

Action tile and merging tiles for a right move Assume that the Turing machine contains move $(q, a) \to (q', a', R)$. Then the tiles in figure 2 are added to the tile set.

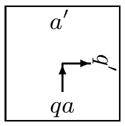


(a) The tile which reads the next symbol b.

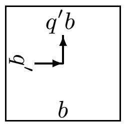


(b) The tile which writes the new symbol a'.

Fig. 1. The tiles representing the read-write head for move $(q, a) \to (q', a', L)$



(a) The tile which writes the new symbol a'.



(b) The tile which reads the next symbol b.

Fig. 2. The tiles representing the read-write head for move $(q, a) \to (q', a', R)$.

Alphabet tiles For every tape alphabet element a, a tile of the form in figure 3 is added to the tile set. This tile represents a single tape cell and its current contents.

Starting tiles To force the Turing machine to start on a blank tape only, the tiles in figure 4 are added to the tile set. One of these tiles (namely, the one in figure 2.1) is chosen to be the seed tile. If the seed tile is contained within a tiling, then a valid tiling will necessarily represent a non-halting Turing machine computation. The tiles in figures 2.1 and 2.1 define the tape to be initially empty. The tiles in figures 2.1 and 2.1 allow the area below the simulation to be correctly tiled always. An example is given in figure 5 on the use of tiles in figure 4.

Following the terminology of Robinson [1], the tiles shown in figures 2.1 and 2.1 are referred to as *action tiles*. The tiles shown in figures 2.1 and 2.1 are called *merging*

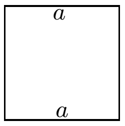
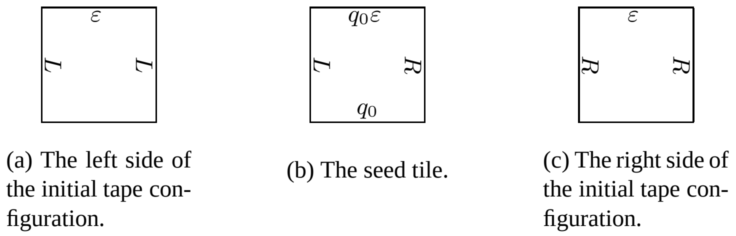


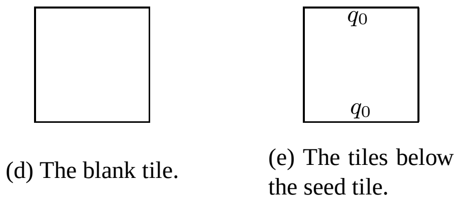
Fig. 3. The tiles to represents the symbols on the tape. Symbol a denotes an arbitrary element of the tape alphabet.



(a) The left side of the initial tape configuration.

(b) The seed tile.

(c) The right side of the initial tape configuration.



(d) The blank tile.

(e) The tiles below the seed tile.

Fig. 4. The tiles that are used to start the Turing machine simulation.

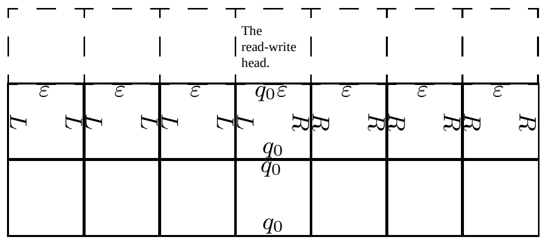


Fig. 5. The pattern representing the initial configuration of the Turing machine computation.

tiles and the tiles of the form in figure 3 are called *alphabet tiles*. The tiles in figure 4 are referred to as *starting tiles*.

Let (q, a) be any preimage pair for which a Turing machine transition $\delta(q, a)$ has not been defined. Then there will be no tile that would have the color qa on its south side. Therefore, if the Turing machine halts, that is, if at some moment of time the read-write head in state q reads symbol a, then the tiling cannot be completed to cover the entire plane in a valid way.

2.2 Representing a Turing machine with a 4-way deterministic tile set

The undecidability of the 4-way determininistic tiling problem with a seed tile would follow directly if the tiles of section 2.1 could be modified to produce a 4-way deterministic tile set.

The original tile set is 4-way deterministic when action tiles and merging tiles are excluded. Therefore, it is sufficient to be able to distinguish two different action tiles or merging tiles from each other and to distinguish action tiles and merging tiles from other tiles.

First, the tile set of Robinson is modified so that the tile set consisting only of action tiles and merging tiles becomes 4-way deterministic. Second, the tile set is modified so that the action tiles and merging tiles can be distinguished from rest of the tiles 4-way deterministically. Formally, these modifications are done by constructing the tile set as a sandwich tile set in four layers as follows:

Layer 1. The tiling representing a Turing machine computation using the tile set construction of Robinson [1].
Layer 2. Horizontal signals identifying the action tile and merging tile pair (at layer 1) occurring on the particular row.
Layer 3. The tiles used to distinguish the action tiles and the merging tiles from alphabet tiles of layer 1 NW- and SW-deterministically.
Layer 4. The tiles used to distinguish the action tiles and the merging tiles from alphabet tiles of layer 1 NE- and SE-deterministically.

Distinguishing different action tiles and merging tiles Let the number of different action tiles and merging tiles of layer 1 be n for the given Turing machine. For simplicity, let the different action tiles and merging tiles be identified with expressions $t_1, \ldots, t_n$.

Let t_k be any action tile or merging tile occurring on layer 1. Assume that integer k identifies the tile uniquely. Then the tile is paired with the tiles of the form in figure 6 with either $i = k$ or $j = k$. If the tile is of the form in figures 2.1 or 2.1, it is required that $i = k$ and $j \neq k$. Otherwise, if the tile is of the form in figures 2.1 or 2.1, it is required that $i \neq k$ and $j = k$. This makes it possible to distinguish any two action tiles or merging tiles from each other just by observing the color of (either) the east side or the west side.

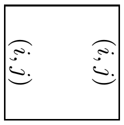
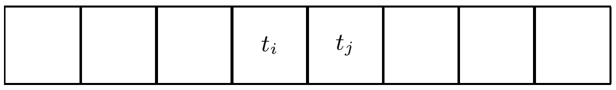


Fig. 6. The tiles of layer 2.

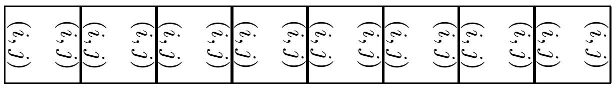
All the other tiles are paired freely with all the tiles of the form in figure 6, where $1 \leq i, j \leq n$. On a valid tiling, layer 2 consists of rows of tiles like the one in figure 2.2 if the underlying tile row contains action tile and move tile pair t_i and t_j as in figure 2.2.

Distinguishing the action tiles and the merging tiles from other tiles

Lemma 1 (J. Kari, 2006). *There exists a 4-way deterministic tile set D which admits a valid tiling of the plane. Furthermore, there exists a valid tiling on which the tiles of*



(a) Layer 1.



(b) Layer 2.

Fig. 7. Associating action tile and move tile pair t_i and t_j with the horizontal signal identifying the tile pair.

a proper subset set $D' \subset D$ are exactly the tiles that are located on a single two-way infinite diagonal line.

Proof (sketch). Due to the details of the construction, the proof is skipped almost entirely.

The aperiodic tile set of Kari and Papasoglu [5] can be modified to produce the desired tile set. Namely, certain kinds of binary signals can be drawn over the aperiodic tiling to locate the "diagonal". These binary signals are started and redirected over certain tiles of the original tile set [5]. The restriction of the signal tile set is 4-way deterministic on each of the tiles of the aperiodic tile set. Because each of the original tiles is paired with all the tiles of a 4-way deterministic tile set, the final tile set is 4-way deterministic. □

Let D be the tile set of lemma 1 which is used to draw a northeast-southwest diagonal line 4-way deterministically. Let $D = D_1 \cup D_2$, where D_1 is the set of tiles used on the diagonal line only and let $D_2 = D \setminus D_1$. Let D^R be the tile set which has been constructed by interchanging the north side colors and south side colors in the tiles of set D. Now D^R can be used to tile a diagonal line in the northwest-southeast direction. Let D_1^R be the set of tiles located on this diagonal pattern and let again $D_2^R = D^R \setminus D_1^R$. Using tile sets D and D^R one can distinguish action tiles and merging tiles from alphabet tiles by pairing the action tiles and the merging tiles of layer 1 with the tiles representing a diagonal line.

The action tiles and the merging tiles 2.1 are paired with the tiles of set D_1 at layer 3 and the action tiles and the merging tiles 2.1 are paired with the tiles of set D_1 at layer 4. In a similar way, the action tiles and the merging tiles 2.1 are paired with the tiles of set D_1^R at layer 3 and the action tiles and the merging tiles 2.1 are paired with the tiles of set D_1^R at layer 4.

Unless otherwise defined above, the tiles that are located on row with a right move (which can be determined by inspecting the colors used on the particular tile at layer 2) are associated with the tiles in D_2 on layers 3 and 4. Similarly, unless otherwise defined, the tiles that are located on row with a left move are associated with the tiles in D_2^R on layers 3 and 4.

The idea of this construction is better seen in figure 8. The left side tiles are identified by the tiles on the diagonal lines on layer 3 and the right side tiles are identified by the tiles on the diagonal lines on layer 4.

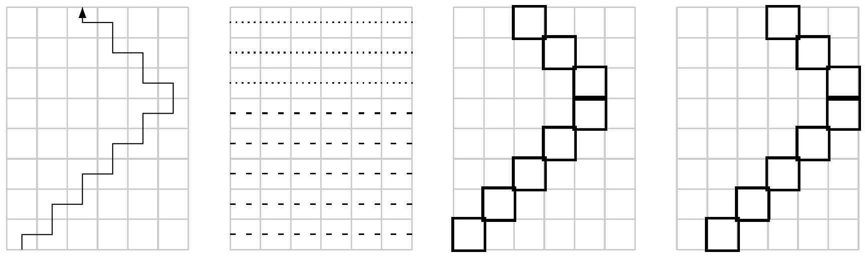

(a) Read-write head movement on layer 1.
(b) Distinguishing different move directions on layer 2.
(c) Simulating a diagonal line on layer 3.
(d) Simulating a diagonal line on layer 4.

Fig. 8. Distinguishing action tiles and merging tiles from alphabet tiles by using diagonal patterns that can be drawn 4-way deterministically.

This construction distinguishes the action tiles and the merging tiles from rest of the tiles. The color used on layer 2 can be used to determine which tile set, either D or D^R, has been used on layers 3 and 4. It needs to be noted that $D \cup D^R$ may not be 4-way deterministic and thus it is important (on layer 2) to distinguish moves with different directions. Furthermore, if layer 1 has been tiled in a valid way, also layers 3 and 4 can be tiled correctly. This follows from the fact that a row of tiles of D can always be followed by a row of tiles D^R with matching colors since D^R is only a "reflected" version of D and D admits a valid tiling.

It was possible to construct a 4-way deterministic tile set which can be mapped homomorphically onto Robinson's original tile set. Hence, the following theorem holds:

Theorem 1. *The tiling problem with a seed tile remains undecidable when restricted to tile sets that are 4-way deterministic.*

3 The 4-way deterministic tiling problem

A finite row of tiles on the tiling by the aperiodic tile set is called *free* if it runs from a horizontal border of a red square to another horizontal border of the same square without intersecting any smaller red squares. A *free* column is defined in a similar manner.

An instance of the tiling problem with a seed tile can be converted to an instance of the tiling problem essentially the same way as was done by Robinson [1]. The instance will be constructed so that the tiling problem with a seed tile will be "simulated" on the intersections of free rows and columns.

The new instance (i.e. the tile set) is constructed in six layers for a given tile set T and a seed tile $t \in T$. The rough outline of layers is the following:

Layer 1. The tiling forced by the aperiodic tile set of Kari and Papasoglu.
Layer 2. The tiles to identify free areas.
Layer 3. A tiling simulating a tiling by the given tile set T.
Layer 4. The tiles to force a copy of the seed tile $t \in T$ to be located at the center of every red square of layer 1
Layer 5. The tiles to forward the colors of the tile set T at layer 3 from a free area border to a free area border.
Layer 6. The tiles to forward the colors of the tile set T at layer 3 from a red border to a red border.

Theorem 2. *The tiling problem is undecidable even when restricted to tile sets that are 4-way deterministic.*

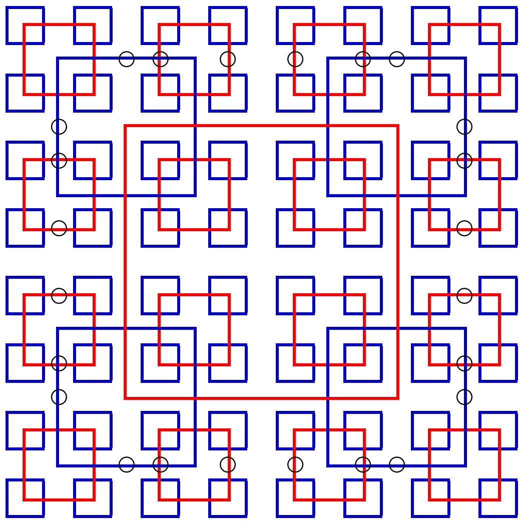
Proof (sketch). It is possible to divide the plane into squares of increasing size using the tile set of Kari and Papasoglu. The squares are colored either red or blue. No borders of two squares of the same color can coincide.

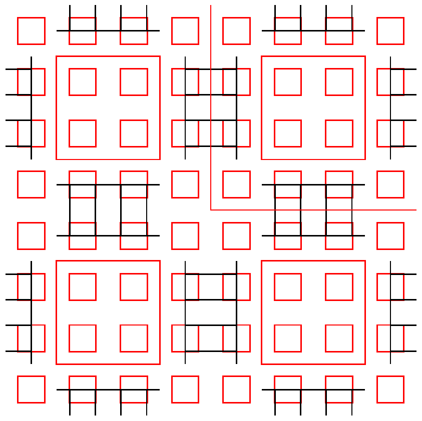
The free rows and the free columns can be located 4-way deterministically. This can be seen from figure 9. First, certain tiles (shown in figure 3) in the aperiodic tiling can be identified. These tiles are used to draw signals which are used to locate the corner points of the free rows and columns. To be precise, the signals are drawn to identify the end points of the non-free rows and column as shown in figure 3.

The simulation of the tiling problem with a seed tile can be done the same way as in the original proof [1]. A finite area of a tiling by the original tile set is simulated on the free areas within the red squares. The size of the free area inside a red square square is directly proportional to the size of the red square [1]. The area consisting of disjoint free areas can be considered as a single continuous square even in the 4-way deterministic case.

One copy of the seed tile can be forced to be located at the center of the simulation area with a simple 4-way deterministic construction. Therefore, a tiling by the original tile set is forced to be simulated at arbitrarily long distances from the seed tile to any direction.

The plane is tiled correctly if, and only if, on every red square the free area is tiled correctly using the original tile set. Any valid tiling by the original tile set can be simulated using the new tile set without a tiling error. This is seen by transferring the outermost colors between the red squares by choosing the outermost colors nondeterministically. □


(a) The special points in the tiling to construct line patterns that are used to identify the free rows and columns.


(b) The free rows and columns can be distinguished from the non-free areas by drawing certain binary signals between the locations shown in figure 3.

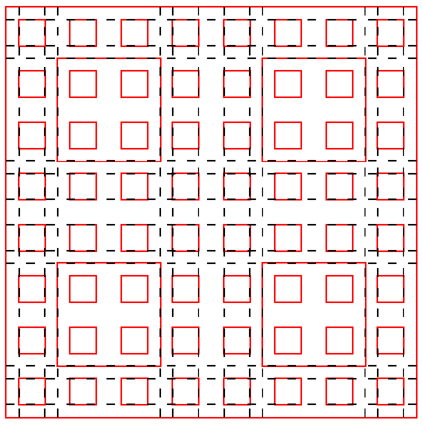

(c) Using the signals shown in figure 3, borders can be drawn for the free areas.

Fig. 9. Determining the free areas 4-way deterministically. First, the end points of the free rows and columns are determined. Second, the borders are drawn between the end points.

4 Acknowledgements

The author is grateful to Professor J. Kari for guidance, useful discussions and advices. The financial support by the Finnish Cultural Foundation for this research is gratefully acknowledged.

References

1. Robinson, R.M.: Undecidability and nonperiodicity for tilings of the plane. Inventiones Mathematicae **12** (1971) 177–209
2. Berger, R.: The undecidability of the domino problem. Mem. Amer. Math. Soc. **66** (1966) 1–72
3. Kari, J.: The nilpotency problem of one-dimensional cellular automata. SIAM Journal on Computing **21** (1992) 571–586
4. Lukkarila, V.: On the tiling problem and reversible cellular automata. Technical Report 788, TUCS (2006)
5. Kari, J., Papasoglu, P.: Deterministic aperiodic tile sets. Geometric and functional analysis **9** (1999) 353–369

DNA: Not Merely the Secret of Life

Nadrian C. Seeman

Department of Chemistry, New York University, New York, NY 10003, USA
ned.seeman@nyu.edu

DNA is well-known as the genetic material of living organisms. Its most prominent feature is that it contains information that enables it to replicate itself. This information is contained in the well-known Watson-Crick base pairing interactions, adenine with thymine and guanine with cytosine. The double helical structure that results from this complementarity is well-known, and is indeed a cultural icon for our era. However relevant the double helix is to genetics, it is a limiting feature when the goal is to prepare more complex chemical species, because the helix axis is linear, in the sense that it is unbranched. Consequently, the only structures that can be obtained by combining molecules with linear helix axes are longer linear molecules or perhaps cyclic species, including knots and catenanes. This problem can be solved by using the notion of reciprocal exchange, which leads to branched species. The topologies of these species are readily programmed through sequence selection; in many cases, it is also possible to program their structures. Branched species can be connected to one another using the same interactions that genetic engineers use to produce their constructs, cohesion by molecules tailed in complementary single-stranded overhangs, known as 'sticky ends.' Such sticky-ended cohesion is used to produce N-connected objects and lattices.

Structural DNA nanotechnology is based on using stable branched DNA motifs with 3-12 arms, or related structures, such as double crossover (DX), triple crossover (TX), and paranemic crossover (PX) motifs. We have been working since the early 1980's to combine these DNA motifs to produce target species. From branched junctions, we have used ligation to construct DNA stick-polyhedra and topological targets, such as Borromean rings.

Nanorobotics are key to the success of nanotechnology. PX DNA has been used to produce a robust sequence-dependent device that changes states by varied hybridization topology. We have used this device to make a translational device that prototypes the simplest features of the ribosome. A protein-activated device that can be used to measure the ability of the protein to do work, and a bipedal walker have both been built. Recently, we have extended the 2-state device to become a 3-state device.

A central goal of DNA nanotechnology is the self-assembly of periodic matter. We have constructed 2-dimensional DNA arrays from many different motifs. We can produce specific designed patterns visible in the AFM from DX and TX molecules. We can change the patterns by changing the components, and by modification after assembly. Recently, we have used DNA scaffolding to organize active DNA components, as well as other materials. Active DNA components include DNAzymes and DNA nanomechanical devices; both are active when incorporated in 2D DNA lattices. Multi-tile DNA arrays have also been used to organize gold nanoparticles in specific arrangements.

The key structural challenge in the area is the extension of the 2D results obtained so far to 3D systems with a high degree of ordering. Several 3-space spanning motifs have been produced that can produce 2D arrays in each of the three directions containing a pair of the vectors that span the 3D space. Crystals with dimensions as large as a millimeter, ordered to 10 resolution (as determined by X-ray diffraction) have been produced. Ultimately, we expect to produce high resolution crystals of DNA host lattices with heterologous guests, leading to bio-macromolecular systems amenable to diffraction analysis.

A Tiling Approach to the Structure and Assembly of Viruses

Reidun Twarock

Departments of Mathematics and Biology
University of York
Heslington
York YO10 5DD, England

Viruses have a protein shell that encapsulates and hence provides protection for the viral genome. It has long been recognised that viral capsids are highly symmetrical, and that the organisation of the proteins in the capsids follows icosahedral symmetry. In this talk we discuss how the protein stoichiometry of viral capsids can be modelled in terms of tilings with icosahedral symmetry (see, for example, [1], [2]). These tilings encode the locations both of the capsid proteins and of the intersubunit bonds between them, and they hence lend themselves as a basis for the construction of virus assembly models. Such models assist in the development of new anti-viral strategies, for example, by suggesting mechanisms to misdirect assembly.

For large classes of viruses capsid assembly does not depend on the presence of the viral genome. For example, in vitro experiments based on the major capsid protein VP1 of Simian Virus 40 show that recombinant VP1 self-assemble into different types of protein containers, tubular structures and sheets depending on the experimental conditions (e.g. pH value and ion concentrations) [3]. From a mathematical point of view one can therefore model this assembly process as tile assembly [4,5]: Tiles representing clusters of capsid proteins attach to each other and gradually build up to a tiling that encodes the topology of the proteins and inter-subunit bonds in the complete capsid [1,2].

In those cases where the genomic material is important for assembly (for example bacteriophage MS2 [6]), these tile assembly models need to include boundary conditions implied by the interaction of the capsid proteins and the genomic material. We show here that such boundary conditions can be included into the models via a new symmetry principle that encodes all material boundaries in simple viruses [7,8]. Moreover, we discuss how the new symmetry principle may lead to tile assembly with three-dimensional, rather than the previously used schematic two-dimensional, building blocks.

References

1. R. Twarock (2004) A tiling approach to virus capsid assembly explaining a structural puzzle in virology. J. Theor. Biol. 226, 477.
2. T.Keef & R.Twarock (2006) A new series of polyhedra as blueprints for viral capsids in the family of Papovaviridae, q-bio.BM/0512047, submitted.
3. S. Kanesashi, S. et al (2003) Simian virus 40 VP1 capsid protein forms polymorphic assemblies in vitro. J. Gen. Virol. 84, 1899.

4. T.Keef, A.Taormina & R.Twarock (2005) Assembly Models for Papovaviridae based on Tiling Theory. Phys. Biol. 2, 175.
5. T. Keef, C. Micheletti & R.Twarock (2006) Master equation approach to the assembly of viral capsids. J. Theor. Biol. 242, 713.
6. P. Stockley et al. (2006) A simple, RNA-Mediated Allosteric Switch Controls the Pathway to Formation of a T=3 Capsid, submitted.
7. T. Keef, N. Grayson, S. Severini & R. Twarock (2007) Self-Assembly of Viral Capsids via a Hamiltonian Paths Approach: The Case of Bacteriophage MS2, in progress.
8. T. Keef, K. Toropova, N. Ranson, P.G. Stockley & R. Twarock (2007) Unmasking the consequences of icosahedral symmetry in a simple virus particle, submitted.

Computing by Self-assembly of Flexible Tiles

Nataša Jonoska

Department of Mathematics,
University of South Florida, Tampa FL, USA

DNA molecules have been assembled in rigid DX and TX molecules, arrayed in assemblies similar to Wang tiles, as well as flexible branched junction molecules with flexible arms that have been used in assemblies of arbitrary graphs. We present a theoretical model for self-assembling tiles with flexible branches motivated by DNA branched junction molecules. We encode an instance of a "problem" as a pot of such tiles, and a "solution" as an assembled complete complex without any free sticky ends (called ports), whose number of tiles is within predefined bounds. We develop an algebraic representation of this self-assembly process and use it to prove that this model of self-assembly precisely captures NP-computability when the number of tiles in the minimal complete complexes is bounded by a polynomial. We also show relationship between both models of rigid and flexible tiles and show how to simulate computations obtained from (bounded) rigid tile self-assembly by corresponding assemblies of flexible tiles.

SAT-TS: a SAT-based tool to recognize and complete pictures specified by tiling

Matteo Pradella and Stefano Crespi Reghizzi

CNR IEIIT–MI and DEI – Politecnico di Milano
Piazza Leonardo da Vinci, 32,
I-20133 Milano, Italy
e-mail: {pradella, crespi}@elet.polimi.it

Extended Abstract

Syntactic methods (see for instance [1],[2]) have been often considered for performing pattern analysis and recognition, by formally specifying the class of pictures of interest. Pictures or patterns can be specified by different methods, such as grammars or automata. A sample of approaches can be found in [3], including for instance [4], where isometric array grammars are considered for efficient syntactic pattern recognition and picture generation. An alternative, theoretically sound, yet practically unexplored, approach is to use tiling: in the crudest form a specified set of small, say two by two, tiles is listed, which can cover the intended class of pictures. A picture is recognized if, and only if, it can be covered with tiles from the listed set. To overcome the limitations of such rudimentary method, a more flexible formalism, called *Tiling Systems* (TS) has been studied by theoreticians (see e.g. [5], [6], [7]). *Wang Tiles* [8] are an equivalent variant of the formalism, which uses a more traditional concept of tiling where tiles are placed side by side. With TS the picture is obtained by first covering it with tiles drawn from a listed set of two by two tiles, then by performing a pixel by pixel mapping. Tiling Systems are a powerful technique: the corresponding pictures can be recognized by non-deterministic cellular automata, which orderly scan the diagonals [9]. Such abstract machines are more powerful then the four ways automata of [10]. However TS definitions are hard to write and error-prone for non elementary pictures. Moreover the NP-complete computational complexity of picture recognition has until now blocked any attempt to realistic experimentation and application of TS, in spite of a large amount of theoretical work.

Our work is concerned with a practical experimentation of tiling systems/Wang tiles in conjunction with a new approach for performing pattern recognition and image generation or completion, based on powerful logical tools, the SAT-solvers, whose task is to find Boolean values which make a propositional formula true. We have implemented a recognizer/generator for TS defined pictures in a very attractive, unconventional way, by transforming the tiling problem into a Boolean satisfiability one, then using an efficient off-the-shelf SAT-solver. The tool is invaluable to assist in writing picture specifications, is fast enough to experiment on reasonably sized samples, and has the bonus of being able to complete a partial picture, by assigning to unknown pixels some values which satisfy the picture specification. Therefore, SAT-TS can be also applied to image reconstruction or noise elimination, by parsing a picture where some pixels are tagged as unknown.

A first, very simple and unoptimized prototype of the tool was presented at the ESF workshop *Advances on Two-dimensional language theory*, held in Salerno, Italy, May 3-5, 2006. We intend to show here the more solid and efficient new version, together with several examples and applications, such as the set of geographical maps which can be colored with three colors, and various classes of nested patterns and connected paths. SAT-TS is open source and freely available.[1]

The Encoding

We briefly present here the main ideas and principles for encoding tiling systems into a SAT-problem. The actual encoding implemented in SAT-TS is a Conjunctive Normal Form optimized variant of the one presented next.

Consider a Tiling System $T = (\Sigma, \Gamma, \Theta, \pi)$, where Σ is the input alphabet, Γ is the local alphabet, Θ is the set of tiles on Γ, and $\pi : \Gamma \to \Sigma$ the projection.

Essentially, given an input picture $p \in \Sigma^{*,*}$, i.e. a picture made of symbols taken from Σ, the parsing problem consists in finding a picture $q \in \Gamma^{*,*}$, having the same size as p, such that:

1. its projection coincides with p, i.e. $\pi(q) = p$;
2. its tiling is compatible with Θ, i.e. every 2×2 sub-picture of q is in Θ.

If both conditions are true, then, and only then, $p \in \mathcal{L}(T)$. Clearly, q is not necessarily unique. Notice that this is an instance of typical inverse mathematical problems, which are often computationally challenging.

Now, to encode the problem into SAT, we represent the pixels of the picture q as SAT's propositional variables. In practice, this means that the statement $q(i, j) = a$ (i.e. pixel (i, j) of q contains the symbol a), becomes a propositional variable of the SAT problem.

To fully exploit the SAT encoding, we also accept partial input pictures. This means that some of p's pixels may be left unspecified (conventionally marked by a "don't care" symbol '?'). With a slight abuse of notation, we say that the inverse projection of a "don't care" symbol in p is Γ, i.e. $\pi^{-1}(?) = \Gamma$. Informally, this means that we do not know anything about that pixel, so any symbol of the tile alphabet could be in q at that position.

The encoding consists of expressing the afore mentioned Conditions 1) and 2), as propositional logic formulas.

Condition 1) states that q must be "compatible" with p, i.e. such that $\pi(q) = p$: [2]

$$F_1 := \bigwedge_{(i,j) \in [(1,1)..|p|]} \left(\underset{a \in \pi^{-1}(p(i,j))}{\mathrm{OnlyOne}} \left(q(i,j) = a \right) \right)$$

[1] http://www.elet.polimi.it/upload/pradella/

[2] For conciseness, we introduce the $\mathrm{OnlyOne}$ Boolean function, with any number of arguments. Informally, $\mathrm{OnlyOne}$ is true if, and only if, exactly one of its arguments is true. E.g. $\mathrm{OnlyOne}\,(A, B, C) \iff (A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C)$.

F_1 depends only on p and on the projection π. The first AND is used to span the whole picture, while the inner $\mathrm{OnlyOne}$ operator is used to check that one and only one value taken from the alphabet Γ is assigned to q at a given position.

Condition 2) considers the tile set Θ: to accept p, every tile used in q must be a member of Θ.

$$F_2 := \bigwedge_{(i,j)\in[(1,1)..|p|]} \bigvee_{t\in\Theta} \bigwedge_{h,k\in[0,1]} \begin{pmatrix} q(i+h, j+k) \\ = \\ t(h+1, k+1) \end{pmatrix}$$

As in the previous formula, the first AND spans the whole picture. Then, the inner OR states that one of the tiles in Θ must be present at a given position.

The TS-recognition problem is then encoded as the propositional formula $F_1 \wedge F_2$.

References

1. Eiichi Tanaka. Theoretical aspects of syntactic pattern recognition. *Pattern Recognition*, 28(7):1053–1061, 1995.
2. K. S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, Englewoods Cliffs, 1982.
3. Nikolaos G. Bourbakis. Special issue on languages for image processing and pattern recognition. *Pattern Recognition*, 32(2):253, 1999.
4. Kenichi Morita and Katsunobu Imai. Uniquely parsable array grammars for generating and parsing connected patterns. *Pattern Recognition*, 32(2):269–276, 1999.
5. Dora Giammarresi and Antonio Restivo. Two-Dimensional Languages. In Arto Salomaa and Grzegorz Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, pages 215–267. Springer-Verlag, Berlin, 1997.
6. R. Siromoney, K.G. Subramanian, V.R. Dare, and D.G. Thomas. Some results on picture languages. *Pattern Recognition*, 32(2):295–304, 1999.
7. Alessandra Cherubini, Stefano Crespi Reghizzi, Matteo Pradella, and Pierluigi San Pietro. Picture Languages: Tiling Systems versus Tile Rewriting Grammars. *Theoretical Computer Science*, 356(1-2):90–103, 2006.
8. C. Allauzen and B. Durand. Tiling problems. In E. Borger and E. Gradel, editors, *The classical decision problem*. Springer-Verlag, 1997.
9. Katsushi Inoue and Akira Nakamura. Some properties of two-dimensional on-line tessellation acceptors. *Information Sciences*, 13:95–121, 1977.
10. M. Blum and C. Hewitt. Automata on a two-dimensional tape. In *IEEE Symposium on Switching and Automata Theory*, pages 155–160, 1967.

Generation of contour words of rectilinear pseudo-parallelograms

Christiane Bercoff[*]

c.bercoff@univ-cezanne.fr

Abstract. In this paper, the 5-tiles, simply connected subsets of $\mathbb{R}^2$ whose the pattern is a rectilinear pseudo-parallelogram (an extended definition of Beauquier-Nivat), are defined from a rewriting system on the contour word of its pattern. From our definition, 5-tiles can to make the regular tilings of the plane by translation of one given pattern.

1 Introduction

Let us consider the lattice $\mathbb{Z} \times \mathbb{Z}$. A *cell* is a unit square whose vertices have integer coordinates. A (directed) polygonal line is a path on the lattice; it is said to be *simple* if no edge appears twice and it does not cross itself. Two cells are said to be *adjacent* if they have a common edge and to be *contiguous* if they have a single common point. The *boundary* of two contiguous cells is a simple polygonal line passing around their contact point, that's why the contact point of two contiguous cells is said be an *avoidable point*. A *rectilinear tile* is a finite union of adjacent or contiguous cells whose its boundary is a *unique* polygonal line. The boundary of a rectilinear tile is clockwise oriented from an origin.

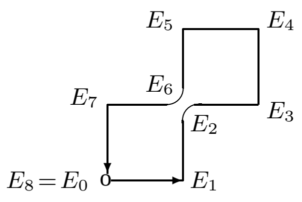


Fig. 1. This polygonal line is the boundary of the rectilinear tile built with two contiguous cells. $E_2 = E_6$ is an avoidable point.

Let T be a rectilinear tile and let A, B be two points on its boundary ∂T. The directed polygonal line from A to B along ∂T will be denoted by $[AB]$ and its *length* will be denoted by $\big|\,[AB]\,\big|$. If A belongs to the boundary ∂T we denote A' the point such as $\big|\,[AA']\,\big| = \big|\,[A'A]\,\big| = \frac{1}{2}\,\big|\partial T\big|$ A' is called the *symmetric* of A *for* ∂T.

[*] The author is supported by LATP, 39 rue Jolliot Curie, 13453 Marseille Cedex 13 - France

Definition 1. *(extended definition of* BEAUQUIER-NIVAT *[1] valid for rectilinear tile)*
A rectilinear tile is a pseudo-parallelogram $ABA'B'$ if there are two points $A, B \in \partial T$ such that $B \in [A\,A']$ with A' symmetric of A, $B' \in [A'\,A]$ with B' symmetric of B and

$$\partial T = [A\,B] \cup [B\,A'] \cup [A'\,B'] \cup [B'\,A]$$

where $[B'A']$ is the translation of vector $\overrightarrow{AB'}$ of $[BA]$ and
$[AB']$ is the translation of vector $\overrightarrow{BA}$ of $[BA']$

We consider the free group on $< x, y >$ where x represents a right step, y a up step and the inverse $\overline{x}$ (resp. $\overline{y}$) a left step (resp. a down step). A *reduced word* is a word on $\Sigma = \{x, y, \overline{x}, \overline{y}\}$ where all cancellations are done (namely each occurrrence of $a\overline{a}, a \in \Sigma$ is replaced by the empty word ε).
If $s \in \Sigma$, $\overline{s}$ denotes the *inverse word* of s i.e.

$$\overline{s} = \overline{s_1 s_2 \ldots s_n} = \overline{s_n}\,\overline{s_{n-1}} \ldots \overline{s_1}.$$

If T is a rectilinear tile, ω_T denotes the *contour word* that is a reduced word on Σ which codes the sequence of moves along the boundary, starting from an origin. The starting point is not meaningful. Thus, the contour word ω_T is a *cyclic* word on Σ^*.

Our study takes the following property as one's starting point.

Proposition 1. *The contour word ω of the rectilinear pseudo-parallelogram $ABA'B'$ admits the* **exact factorization**

$$\omega = \alpha\,\beta\,\overline{\alpha}\,\overline{\beta}$$

where $\alpha = \omega_{[A\,B]}, \quad \beta = \omega_{[B\,A']}, \quad \overline{\alpha} = \omega_{[A'\,B']}, \quad \overline{\beta} = \omega_{[B'\,A]}.$

But, all word on Σ^* as form $\alpha\,\beta\,\overline{\alpha}\,\overline{\beta}$ **is not the contour word of rectilinear tile** because the *associated path*, denoted by $P(\alpha\,\beta\,\overline{\alpha}\,\overline{\beta})$, can cross itself : for example, if $\alpha = x\,y$ and $\beta = y\,x$, then the associated path to word $\alpha\,\beta\,\overline{\alpha}\,\overline{\beta} = x\,y\,y\,x\,\overline{y}\,\overline{x}\,\overline{x}\,\overline{y}$ is not simple.

2 Main results

Let T a rectilinear tile. The image of T in the translation of vector u will be denoted by $T(u)$ and $T(u)$ is called an *instance* of T.

Definition 2. *A 5-tuile of pattern* **q** *is a rectilinear tile make up the pseudo-parallelogram* **q** *surrounded with the four instances* **q**(τ),**q**(ν), **q**$(-\tau)$, **q**$(-\nu)$ *without gap or overlapping between them.*

In other words, the rectilinear pseudo-parallelogram **q** tiles the 5-tile of pattern **q**.
The circular sequence $\big($**q**(τ),**q**(ν),**q**$(-\tau)$, **q**$(-\nu)\big)$ is an *exact* 4-*surrounding* in sense of Beauquier-Nivat[1].

We prove the first result.

Proposition 2. *Let* **q** *be a rectilinear tile. The following two assertions are equivalent:*

1. *there exists two independent vectors of translation τ and ν with integer coordinates such that $\big(\mathbf{q}(\tau),\, \mathbf{q}(\nu),\, \mathbf{q}(-\tau),\, \mathbf{q}(-\nu)\big)$ is an exact 4-surrounding of* **q**.
2. *there exists $A, B, A', B' \in \partial\,\mathbf{q}$ where A' (resp. B') is the symmetric of A (resp. of B) such that*

$$\begin{array}{ll} \textit{if} & [AB] = \mathbf{q} \cap \mathbf{q}(\tau) \textit{ and } [BA'] = \mathbf{q} \cap \mathbf{q}(\nu) \\ \textit{then} & [A'B'] = \mathbf{q} \cap \mathbf{q}(-\tau) \textit{ and } [B'A] = \mathbf{q} \cap \mathbf{q}(-\nu) \end{array}$$

Thus, the points $A_0 = \mathbf{q} \cap \mathbf{q}(\tau) \cap \mathbf{q}(-\nu)$, $B_0 = \mathbf{q} \cap \mathbf{q}(\tau) \cap \mathbf{q}(\nu)$, $A'_0 = \mathbf{q} \cap \mathbf{q}(\nu) \cap \mathbf{q}(-\tau)$ and $B'_0 = \mathbf{q} \cap \mathbf{q}(-\tau) \cap \mathbf{q}(-\nu)$ define a *bichromatic colouring* of ∂**q**.

We state a particular case of theorem 4.2 of Beauquier-Nivat [1].

Theorem 1. *A rectilinear tile* **q** *have at least one exact 4-surrounding if and only if* **q** *is a rectilinear pseudo-parallelogram.*

Now, we can build some 5-tile.

Theorem 2. *Let* **q** *be a rectilinear tile. The three assertions are equivalent:*

1. *the boundary ∂**q** have at least one bichromatic colouring*
2. *there exists two words α and β such that $\omega_{\mathbf{q}} = \alpha\,\beta\,\overline{\alpha}\,\overline{\beta}$*
3. *there exists two independent vectors of translation τ and ν with integer coordinates such that*

$$T = \mathbf{q} \cup \mathbf{q}(\tau) \cup \mathbf{q}(\nu) \cup \mathbf{q}(-\tau) \cup \mathbf{q}(-\nu)$$

is a rectilinear pseudo-parallelogram

Since a 5-tile of pattern **q** is a rectilinear pseudo-parallelogram, it can be using as pattern for the 5-tile, called 5^2**q**-tile, which is tiled with 25 instances of **q**. And so on . . .

In the following, we introduce a rewriting system which generates, from derivation of an exact factorization of contour word of a given rectilinear pseudo-parallelogram **q**, all the contour words of 5-tuile of pattern 5^n**q** for every integer $n > 0$.

Definition 3. *Let $\mathcal{A} = \{\alpha, \beta, \overline{\alpha}, \overline{\beta}\}$ be an alphabet where α and β are words on Σ^* such that the path $P(\alpha\,\beta\,\overline{\alpha}\,\overline{\beta})$ is simple. Let $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ be a rewriting system on $\mathcal{A}$ where*

$$\mathcal{R}_1 = \{\, \alpha \to \alpha\,\beta\,\alpha \;;\; \beta \to \beta\,\overline{\alpha}\,\beta \;;\; \overline{\alpha} \to \overline{\alpha}\,\overline{\beta}\,\overline{\alpha} \;;\; \overline{\beta} \to \overline{\beta}\,\alpha\,\overline{\beta} \,\}$$
$$\mathcal{R}_2 = \{\, \alpha \to \alpha\,\overline{\beta}\,\alpha \;;\; \beta \to \beta\,\alpha\,\beta \;;\; \overline{\alpha} \to \alpha\,\overline{\beta}\,\overline{\alpha} \;;\; \overline{\beta} \to \overline{\beta}\,\overline{\alpha}\,\overline{\beta} \,\}$$

Proposition 3. *Let ω_1 and ω_2 two words which derive from $\alpha\,\beta\,\overline{\alpha}\,\overline{\beta}$ in the system $\mathcal{R}$*

$$\alpha\,\beta\,\overline{\alpha}\,\overline{\beta} \xrightarrow{\mathcal{R}_1} \omega_1 \;\; \textit{and} \;\; \alpha\,\beta\,\overline{\alpha}\,\overline{\beta} \xrightarrow{\mathcal{R}_2} \omega_2$$

Then

1. *ω_1 and ω_2 are conjugate : there exists $u_1, u_2 \in \mathcal{A}^*$ such that $\omega_1 = u_1\,u_2$ and $\omega_2 = u_2\,u_1$*

2. *ω_1 and ω_2 have at least one exact factorization in $\mathcal{A}^*$:*
   there exists $\alpha_1, \beta_1, \alpha_2, \beta_2 \in \mathcal{A}^*$ such that $\alpha_1\, \beta_1\, \overline{\alpha}_1\, \overline{\beta}_1 = \omega_1$ and $\omega_2 = \alpha_2\, \beta_2\, \overline{\alpha}_2\, \overline{\beta}_2$

The sequence $(\mathcal{R}_1^{k_1},\, \mathcal{R}_2^{k_2},\, \ldots,\, \mathcal{R}_j^{k_i})$ with $k_1+k_2+\cdots+k_i = n$ and $j = \begin{cases} 1 & \text{if } i \text{ is odd} \\ 2 & \text{if } i \text{ is even} \end{cases}$
is a n-**dérivation** of rewriting system $\mathcal{R}$.

Now, we prove a combinatoric version of theorem 2.

Theorem 3. *Let* **q** *be a rectilinear pseudo-parallelogram and let $\omega_{\mathbf{q}} = \alpha\, \beta\, \overline{\alpha}\, \overline{\beta}$ be an exact factorization of its contour word. For every integer $n > 0$ the word obtained from a n-dérivation of $\omega_{\mathbf{q}}$ is a contour word of 5^n***q***-tile.*

The following result is an explicit form of theorem's Dekking [4] about the contour word of k-composition defined by iterated morphism on Σ^*. Here, the given result is both :

more particular, because it deals with only the 5-tiles

more general, because the morphism is on the set of words of Σ^*

Theorem 4. *Let ω_∞ the word obtained by transitive fermeture of system $\mathcal{R}$ on a factorization $\alpha\, \beta\, \overline{\alpha}\, \overline{\beta}$ of contour word of the rectilinear pseudo-parallelogram* **q** *i.e.*

$$\alpha\, \beta\, \overline{\alpha}\, \overline{\beta} \quad \overset{*}{\underset{\mathcal{R}}{\Longrightarrow}} \omega_\infty$$

Then, the polygonal path $P(\omega_\infty)$ is simple and its fractal Hausdorff dimension is defined by

$$\dim P(\omega_\infty) = 2\, \frac{\log 3}{\log 5}$$

The subset of lattice limited by the polygonal path $P(\omega_\infty)$ is said be *rep*-5 *fractile* and $1/5$ is the *contraction factor*.

To draw a boundary of 5^n**q**-tile, we use an algorithm which determine at each iteration $i, 0 \le i \le n$ the points A_i, B_i, A'_i, B'_i of the bichromatic colouring of its boundary. Since the rules $\mathcal{R}_1$ and $\mathcal{R}_2$ preserves the exact factorization and the conjugaison (see proposition 3), we have two possible colourings of the boundary of a given 5-tile.

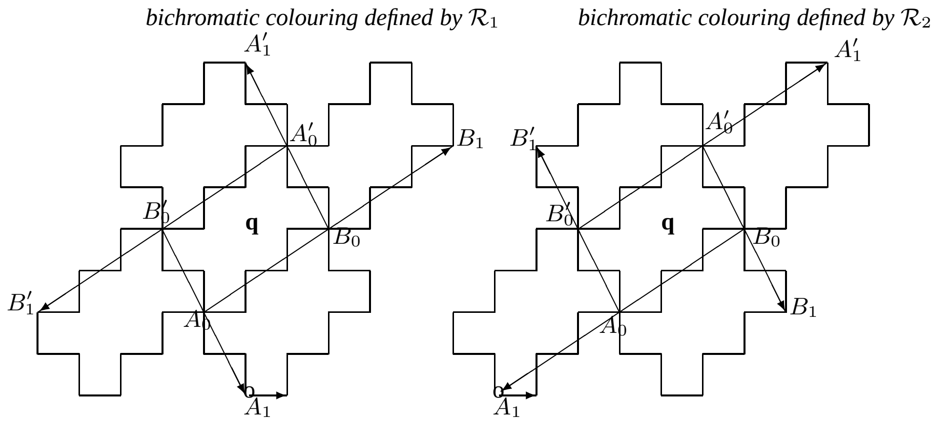

bichromatic colouring defined by $\mathcal{R}_1$
bichromatic colouring defined by $\mathcal{R}_2$
A'_1
A'_0
B_1
B'_0
q
B_0
B'_1
A_0
0
A_1
A'_1
A'_0
B'_1
B'_0
q
B_0
B_1
A_0
0
A_1

Fig. 2. Relative position between the bichromatic colouring points of $\partial\mathbf{q}$ and the bichromatic colouring points of the 5**q**-tile

The bichromatic colouring points A_i, B_i, A'_i, B'_i are iterated defined as following. Let f be the morphism : $(\Sigma, .) \to (\mathbb{Z}\times\mathbb{Z}, +)$ defined by

$$f(x)=(1,0) \text{ et } f(y)=(0,1) \text{ and } f(\overline{s})=-f(s) \text{ and } f(s\,t)=f(s)+f(t) \ \ \forall s,t\in\Sigma^*$$

Algorithm. *Let* **q** *be a rectilinear pseudo-parallelogram which the contour word is* $\alpha\,\beta\,\overline{\alpha}\,\overline{\beta}$*. If* A_0 *is the origin on the boundary* $\partial\mathbf{q}$*, then* $B_0 = f(\alpha)$*,* $B'_0 = f(\beta)$ *and* $A'_0 = f(\alpha) + f(\beta)$*. If* $\tau_0 = \overrightarrow{B'_0A_0}$ *and* $\nu_0 = \overrightarrow{A_0B_0}$ *then*
for $i > 0$,

$$A_{i+1} = A_i + \tau_i,\ B_{i+1} = B_i + \nu_i,\ A'_{i+1} = A'_i - \tau_i,\ B'_{i+1} = B'_i - \nu_i \ \text{ using } \mathcal{R}_1$$
$$A_{i+1} = A_i - \nu_i,\ B_{i+1} = B_i + \tau_i,\ A'_{i+1} = A'_i + \nu_i,\ B'_{i+1} = B'_i - \tau_i \ \text{ using } \mathcal{R}_2$$
$$\tau_{i+1} = A_i - B'_i \text{ and } \nu_{i+1} = B_i - A_i$$

Example. Let **q** be a rectilinear pseudo-parallelogram which the contour word is $\alpha\,\beta\,\overline{\alpha}\,\overline{\beta}$ where $\alpha = x\,y\,x\,\overline{y}\,x$ and $\beta = y\,x\,y\,\overline{x}\,\overline{y}\,\overline{x}\,y\,y$. The 5^2**q**-tile which the contour word is $\mathcal{R}_1\mathcal{R}_1(\alpha\,\beta\,\overline{\alpha}\,\overline{\beta})$ is defined by means :

	A	B	A'	B'	τ	ν
	$(0,0)$	$f(\alpha)$	$f(\alpha)+f(\beta)$	$f(\beta)$	$A-B'$	$B-A$
initialisation	$(0,0)$	$(3,0)$	$(2,3)$	$(-1,3)$	$(1,-3)$	$(3,0)$
$\mathcal{R}_1$	$A+\tau$	$B+\nu$	$A'-\tau$	$B'-\nu$		
iteration 1	$(1,-3)$	$(6,0)$	$(1,6)$	$(-4,3)$	$(5,-6)$	$(5,3)$
iteration 2	$(6,-9)$	$(11,3)$	$(-4,12)$	$(-9,0)$	$(15,-9)$	$(5,12)$

Now, starting from $A_2 = (6, -9)$, we can draw the following 5^2**q**-tile which the contour word is

$$\mathcal{R}_1\mathcal{R}_1(\alpha\,\beta\,\overline{\alpha}\,\overline{\beta}) = \alpha\,\beta\,\alpha\,\beta\,\overline{\alpha}\,\beta\,\alpha\,\beta\,\alpha \quad \beta\,\overline{\alpha}\,\beta\,\overline{\alpha}\,\overline{\beta}\,\overline{\alpha}\,\beta\,\overline{\alpha}\,\beta \quad \overline{\alpha}\,\overline{\beta}\,\overline{\alpha}\,\overline{\beta}\,\alpha\,\overline{\beta}\,\overline{\alpha}\,\overline{\beta}\,\overline{\alpha} \quad \overline{\beta}\,\alpha\,\overline{\beta}\,\alpha\,\overline{\beta}\,\alpha\,\overline{\beta}\,\alpha\,\overline{\beta}$$

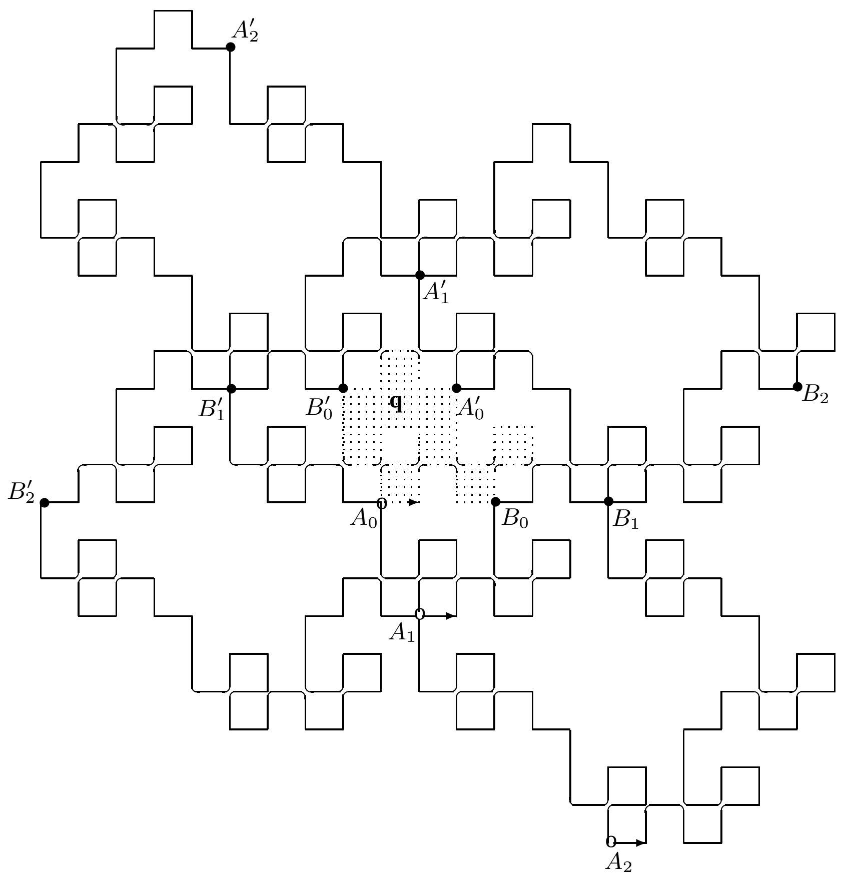


Fig. 3. The 5^2**q**-tile associated to word $\mathcal{R}_1\mathcal{R}_1(\alpha\,\beta\,\overline{\alpha}\,\overline{\beta})$ where $\alpha = x\,y\,x\,\overline{y}\,x$ and $\beta = y\,x\,y\,\overline{x}\,\overline{y}\,\overline{x}\,y\,y$.

Remarks.

We have obtained other results which are in the prepublication [2]. In particular, we exhibit a family of rectilinear column-convex tiles which are all pseudo-parallelograms. Because this tiles are defined with only a part of their contour word (encoding on $\mathbb{N}$), we can enumerate them. In another paper [3] (unpublished up to then), we also consider the rectilinear pseudo-hexagons which are 7-tiles.

References

1. D. Beauquier et M. Nivat, Tiling the plane with one tile, *Prépublication* LITP 90-11, (1990).
2. C. Bercoff, Génération et énumération de mots de contour de tuiles rectilignes pavantes, *Prépublication* LATP 06-11, (2006).
3. C. Bercoff, Generation and enumeration of contour words of tiling rectilinear shapes, *Preprint*, (2007).
4. F.M. Dekking, Replicating Superfigures and Endomorphisms of Free Groups, *J. Comb. Theory, Ser. A* **32**, (1982) 315-320.

ISBN 978-952-12-1921-4
ISSN 1239-1905