Computational Complexity of Some Equivalence-Checking Problems

Zdeněk Sawa

Habilitation Thesis
Faculty of Electrical Engineering and Computer Science
VŠB–Technical University of Ostrava
2012

Abstract

This habilitation thesis gives an overview of five papers of the author in the area of verification, in particular, in the study of computational complexity of equivalence checking problems and related areas.

The first of these papers, Sawa, Jančar: *Equivalences on Finite-State Systems are PTIME-hard* (2005), shows that deciding any relation between bisimulation equivalence (bisimilarity) and trace preorder is PTIME-hard for finite-state systems that are presented explicitly (i.e., as a list of states and transitions).

The second paper, Sawa, Jančar: *Hardness of equivalence checking for composed finite-state systems* (2009), gives a proof that deciding any relation between bisimulation equivalence and trace preorder is EXPTIME-hard for systems composed of interacting finite-state components. Further hardness results are shown for special cases of acyclic systems.

The third paper, Jančar, Kot, Sawa: *Complexity of deciding bisimilarity between normed BPA and normed BPP* (2010), presents a polynomial-time algorithm deciding bisimilarity between a normed BPA process and a normed BPP process, with running time $O(n^7)$.

The forth paper, Fröschle, Jančar, Lasota, Sawa: *Non-Interleaving Bisimulation Equivalences on Basic Parallel Processes* (2010), describes polynomial time algorithms for deciding hereditary history preserving bisimilarity (in $O(n^3 \log n)$) and history preserving bisimilarity (in $O(n^6)$) on BPP processes.

The fifth paper, Jančar, Sawa: *A note on emptiness for alternating finite automata with a one-letter alphabet* (2007) gives a short direct self-contained proof of PSPACE-hardness of the emptiness problem for alternating finite automata with a singleton alphabet, which is much simpler than the original proof by Holtzer (1995) relying on a series of reductions from several papers.

Contents

1 Introduction

Many software and hardware systems are used in critical applications where bugs can have very serious consequences such as loss of important data, financial loss, damage of physical facilities, or even loss of human lives. Examples of such systems are software and hardware systems used in nuclear plants, aircrafts, automobiles, in traffic control systems, etc. Other examples of complicated systems, where bugs can have fatal consequences, are microprocessors, operating systems, and implementations of different communication protocols.

Such systems often consist of many parts running in parallel and communicating with each other. The practice shows that standard techniques such as simulation and testing are often not sufficient to ensure correctness of a design of such systems, because they can show presence of bugs but they can not guarantee their absence. This is due to fact that by simulation and testing we can usually explore only a small part of all potential runs of tested systems and only a small part of states that can be reached by these runs.

Alternative approach, which complements simulation and testing, is *formal verification*. The aim of the verification is to provide a formal proof that a given system has given required properties. Typical properties that can be verified are, e.g., absence of a deadlock, fairness of an allocation of resources, etc. In particular, the goal of formal verification is to ensure that the required properties are satisfied in *all* possible runs of a system.

Construction of such correctness proofs by hand can be very tedious and error-prone, so the idea is to construct tools that will allow to automate the formal verification as much as possible. One of approaches is *theorem proving*, which requires some cooperation with a user, which specifies propositions that must be proved, invariants that should hold, etc., and a tool (a theorem prover) helps to perform routine steps of a proof, to check the correctness of the steps, etc.

More automatic approaches are *model checking* and *equivalence checking*. In model checking we have some (model of a) system and a required property specified in a form of a formula of some temporal logic (e.g., LTL or CTL). The aim is to check whether the system satisfies the given formula.

In equivalence checking we have two systems, where usually one of them represents a *specification* of a required behaviour and the other an *implementation*. The goal is to check whether a behaviour of both systems is equivalent in some sense, or if a behaviour of the implementation is in some sense contained in a behaviour of the specification. In the former case, the equivalence of behaviours of systems is formally defined by some *behavioural equivalence*, in the latter case, the containment of behaviour of one system in a behaviour of some other system is defined by some *behavioural preorder*. There exist a plethora of behavioural equivalences and preorders that were defined in literature.

One of the main obstacles of more widespread use of tools for model checking and equivalence checking is a high computational complexity (or even undecidability) of the corresponding problems on systems that appear in practice. It is well known that both model checking and equivalence checking are undecidable on systems, which are Turing powerful. Even on finite-state systems where these problems often can be solved at least in principle, it is possible that a given problem can not be solved in practice due to so called *state explosion*, i.e., the situation where an algorithm must check a number of states, which is exponential with respect to a size of a presentation of a given system, so the amount of a required time is enormous even for systems of a moderate size.

This motivates a research of computational complexity and decidability of model checking and equivalence checking for different types of systems. One of goals of this research is to clarify where are the bounds of what can be done by efficient algorithms, and also to find efficient algorithm for different problems where it is possible.

This thesis gives an overview of (some of) results of the author in the study of the computational complexity of equivalence checking problems. In particular, the thesis presents results from five articles, where the author participated, and that were published in well respected journals:

- Zdeněk Sawa, Petr Jančar: *Behavioural Equivalences on Finite-State Systems are PTIME-hard*, *Computing and Informatics*, Volume 24, Issue 5, pp. 513–528, Slovak Academy of Sciences, Institute of Informatics, 2005. [5]
- Zdeněk Sawa, Petr Jančar: *Hardness of equivalence checking for composed finite-state systems*, *Acta Informatica*, Volume 46, Issue 3, pp. 169–191, Springer, 2009. [3]
- Petr Jančar, Martin Kot, Zdeněk Sawa: *Complexity of deciding bisimilarity between normed BPA and normed BPP*, *Information and Computation*, Special Issue: 19th International Conference on Concurrency Theory (CONCUR 2008), Volume 208, Issue 10, pp. 1193–1205, Elsevier, 2010. [2]
- Sibylle Fröschle, Petr Jančar, Sławomir Lasota,Zdeněk Sawa: *Non-Interleaving Bisimulation Equivalences on Basic Parallel Processes*, *Information and Computation*, Volume 208, Issue 1, pp. 42–62, Elsevier, 2010. [1]
- Petr Jančar, Zdeněk Sawa: *A note on emptiness for alternating finite automata with a one-letter alphabet*, *Information Processing Letters*, Volume 104, Issue 5, pp. 164–167, Elsevier, 2007. [4]

The first of these papers [5] deals with the complexity of equivalence checking of finite-state systems that are presented explicitly, i.e., by an explicit list of states and transitions. It is shown that deciding any relation between bisimulation equivalence and trace preorder is PTIME-hard. This paper is discussed in more detail in Section 3.

The second paper [3] generalizes the result of [5] to finite-state systems composed of many components running in parallel that communicate with each other. It is proved that deciding any relation between bisimilarity and trace preorder is EXPTIME-hard on a wide variety of such systems, in particular, on systems where components communicate by synchronizing on shared actions and where some actions can be hidden, i.e., replaced by a special "invisible" action τ. This approves the conjecture of Rabinovich [70] that proved PSPACE-hardness of this problem and conjectured its EXPTIME-hardness. The paper [3] also contains some additional results dealing with some special cases of the described problem, for example some results dealing with systems where transition systems of components are acyclic. The paper also presents an overview of known complexity results on this type of systems. The paper [3] is described in Section 4.

The third paper [2] shows that bisimulation equivalence between two types of infinite state systems – normed Basic Process Algebra (nBPA) and normed Basic Parallel Processes (nBPP) can be decided in a polynomial time. The paper presents an algorithm with time complexity $O(n^7)$ for this problem. A preliminary version of the paper appeared on conference Concur 2008 as [7], where it won the best paper award. The paper [2] is described in Section 5.

The fourth paper [1] deals with deciding of so called non-interleaving equivalences on Basic Parallel Processes (BPP). Non-interleaving equivalences are equivalences that distinguish between performing some actions in parallel and performing them sequentially in an arbitrary order (i.e., the interleaving of actions), so these equivalences can be used to model systems composed of processes running in parallel more faithfully. Two main results of [1] are two polynomial time algorithms. The first of these algorithms decides hereditary history-preserving bisimilarity (hhp-b) on BPP and its time complexity is $O(n^3 \log n)$, and the second decides history-preserving bisimilarity (hp-b) on BPP with time complexity $O(n^6)$. Many other non-interleaving equivalences coincide with hp-b on BPP, so the latter algorithm can be also used to decide these equivalences on BPP. Paper [1] also clarifies definitions of hhp-b and hp-b on BPP. The results from [1] are described in more detail in Section 6.

The fifth paper [4] presents a result that does not belong directly to the area of automatic verification but rather to automata theory. It is an alternative proof of PSPACE-hardness of deciding emptiness of a language accepted by an alternating finite automaton with a one-letter alphabet, which was originally shown by Holzer [44]. However, the proof presented in [4] is much simpler than the original Holzer's proof in [44]. The PSPACE-hardness of this automata theoretic problem can be used to show PSPACE-hardness of some equivalence checking and model checking problems, in particular, some problems dealing with one-counter automata. The article [4] is discussed in Section 7.

The thesis is organized as follows. Section 2 presents some basic definitions. It describes behavioural equivalences and preorders and different types of labelled transition systems. Sections 3–7 describe in more detail the above mentioned five articles. Each of these sections corresponds to one of these articles. It gives an overview of known related results (the state of the art) for the corresponding article, shortly states the main results of the article, and then describes these results in more detail and with appropriate technical definitions. Sometimes it also presents main ideas of proofs.

The thesis also contains copies of the discussed five journal articles and a list of publications of the author, together with a list of citations.

2 Definitions

This section gives an overview of definitions of some notions and notation used in the following sections. In particular, Subsection 2.1 recalls some standard mathematical notation used in the rest of the thesis, Subsection 2.2 presents the standard notion of a labelled transition system. Subsection 2.3 then gives a short overview of definitions of some behavioural equivalences and preorders, in particular of bisimulation equivalence (bisimilarity), simulation equivalence and preorder, and trace equivalence and preorder.

Subsections 2.4 and 2.5 describe several particular types of labelled transition systems discussed in the following subsections. Subsection 2.4 describes composed finite-state systems; results concerning this type of systems are described in detail in Section 4. Subsection 2.5 describes several types of infinite-state systems that can be defined as natural subclasses of so called Process Rewrite Systems (PRS). In particular, Basic Process Algebra (BPA) and Basic Parallel Processes (BPP) are described in this subsection. Results concerning these types of systems are discussed in detail in Sections 5 and 6.

Subsection 2.6 recalls definitions of some complexity classes and of types of reductions between problems used in the rest of the thesis.

2.1 Used Notation

$\mathbb{N}$ denotes the set of natural numbers $\{0, 1, 2, \ldots\}$.

Given a set A, $|A|$ denotes the cardinality of A, $\mathcal{P}(A)$ denotes the set of all subsets of A, and A^* denotes the set of finite sequences of elements of A. For $w \in A^*$, $|w|$ denotes the length of w. Symbol ε denotes the empty sequence, i.e., the only sequence such that $|\varepsilon| = 0$.

2.2 Labelled Transition Systems

A *labelled transition system* (an *LTS* for short) is a tuple $\mathcal{T} = (S, \mathcal{A}, \longrightarrow)$ where S is a set of *states*, Act is a finite set of *actions*, and $\longrightarrow \subseteq S \times Act \times S$ is the *transition relation.*

A tuple $(s, a, s') \in \longrightarrow$ (where $s, s' \in S$ and $a \in Act$) is called a *transition.* Instead of $(s, a, s') \in \longrightarrow$ we usually write $s \xrightarrow{a} s'$. We also use notation $s \xrightarrow{w} s'$ for finite sequences of actions $w \in Act^*$; for $w = a_1 a_2 \ldots a_n$ (where $a_i \in Act$) this means that there are some states $s_0, s_1, \ldots, s_n$ such that $s_0 = s$, $s_n = s'$, and $s_{i-1} \xrightarrow{a_i} s_i$ for each $1 \leq i \leq n$.

2.3 Behavioural Equivalences and Preorders

Assume a fixed LTS $\mathcal{T} = (S, Act, \longrightarrow)$.

For a state $s \in S$, we define the set of its *traces* as

$$Traces(s) = \{w \in Act^* \mid \exists s' \in S : s \xrightarrow{w} s'\}\,.$$

States s, t are in the *trace preorder*, written $s \sqsubseteq_{tr} t$, iff $Traces(s) \subseteq Traces(t)$, and they are *trace equivalent*, written $s \equiv_{tr} t$ iff $s \sqsubseteq_{tr} t$ and $t \sqsubseteq_{tr} s$ (i.e., iff $Traces(s) = Traces(t)$).

A relation $\mathcal{R} \subseteq S \times S$ is a *simulation* if for each $(s_1, s_2) \in \mathcal{R}$ and each $a \in Act$ we have:

- $\forall t_1 \in S : s_1 \xrightarrow{a} t_1 \Rightarrow (\exists t_2 \in S : s_2 \xrightarrow{a} t_2 \wedge (t_1, t_2) \in \mathcal{R})$.

A symmetric simulation is called *bisimulation*, i.e., a relation $\mathcal{R} \subseteq S \times S$ is a bisimulation if for each $(s_1, s_2) \in \mathcal{R}$ and each $a \in Act$ we have:

- $\forall t_1 \in S : s_1 \xrightarrow{a} t_1 \Rightarrow (\exists t_2 \in S : s_2 \xrightarrow{a} t_2 \wedge (t_1, t_2) \in \mathcal{R})$, and
- $\forall t_2 \in S : s_2 \xrightarrow{a} t_2 \Rightarrow (\exists t_1 \in S : s_1 \xrightarrow{a} t_1 \wedge (t_1, t_2) \in \mathcal{R})$.

We say that a transition $s_1 \xrightarrow{a} t_1$ is *matched* by $s_2 \xrightarrow{a} t_2$, resp. $s_2 \xrightarrow{a} t_2$ by $s_1 \xrightarrow{a} t_1$, if $(t_1, t_2) \in \mathcal{R}$.

States s, t are in the *simulation preorder*, written $s \sqsubseteq_{sim} t$, iff there is a simulation $\mathcal{R}$ such that $(s, t) \in \mathcal{R}$, and they are *simulation equivalent*, written $s \equiv_{sim} t$, iff $s \sqsubseteq_{sim} t$ and $t \sqsubseteq_{sim} s$. States s, t are *bisimulation equivalent* (or *bisimilar*), written $s \sim t$, iff there is some bisimulation $\mathcal{R}$ such that $(s, t) \in \mathcal{R}$. The relation $\sim$ is called *bisimulation equivalence* or *bisimilarity* [69, 66].

Note that the union of bisimulations is a bisimulation; thus bisimilarity $\sim$ is the maximal bisimulation (the union of all bisimulations). It is easy to check that $\sim$ is an equivalence relation.

It is useful to recall an alternative definition of bisimilarity based on games (cf. for example [77]). The *bisimulation game* on a given LTS $(S, A, \longrightarrow)$ is played by two players — *Attacker* and *Defender*. The *positions* in the game are pairs $(s_1, s_2) \in S \times S$. In a position (s_1, s_2), Attacker chooses $i \in \{1, 2\}$ and a transition from s_i, say $s_i \xrightarrow{a} t_i$; Defender must respond by choosing some transition with the same label a from the other component of the pair (s_1, s_2), i.e., a transition $s_{3-i} \xrightarrow{a} t_{3-i}$. The play then continues from the position (t_1, t_2). If one of the players gets stuck (i.e., there is no appropriate transition), then the other player wins. If the play continues forever, then Defender wins. The *simulation game* is the bisimulation game where Attacker is always obliged to choose $i = 1$ (i.e., to play on the left-hand side).

Generally speaking, a *strategy* for a player P in a game is a (partial) function that determines a concrete move of player P for each sequence $m_1, m_2, \ldots, m_k$ of moves played so far after which it is P's turn. A strategy is a *winning strategy* of P if player P wins each play when he/she uses the strategy. A strategy is *memory-less (positional)* if each prescribed move depends only on the current position, not on the whole sequence of moves played so far.

Proposition 2.1 ([77]) *In the bisimulation game starting from position (s_1, s_2):*

1. *Defender has a (memory-less) winning strategy iff s_1, s_2 are bisimilar,*
2. *Attacker has a (memory-less) winning strategy iff s_1, s_2 are not bisimilar.*

Any bisimulation (simulation) containing (s_1, s_2) naturally provides a winning strategy for Defender in the bisimulation (simulation) game starting from (s_1, s_2).

All above mentioned relations were defined only on pairs of states belonging to *the same* LTS but they can be also used for relating states from different LTSs, since the states of two different LTSs can be naturally viewed as the states of their disjoint union.

There were many other behavioural equivalences and preorders defined in the literature. The most prominent of them were organized by van Glabbeek [79] into so called *linear time – branching time spectrum*, see Figure 1. An arrow in this diagram going from relation X to relation Y means that X is a finer relation than Y, i.e., $(s_1, s_2) \in X$ implies $(s_1, s_2) \in Y$.

Note that in this spectrum, bisimilarity is the finest relation and trace preorder is the coarsest, and all other relation are somewhere between them.

Some results that will be presented in the following sections hold for all relations *between* some pair of relations, for example between bisimilarity and trace preorder. By saying that a relation Z on the states of LTSs is *between* relations X and Y (which are also relations on the states of the LTSs), we mean that $(s_1, s_2) \in X$ implies $(s_1, s_2) \in Z$, and $(s_1, s_2) \in Z$ implies $(s_1, s_2) \in Y$.

2.4 Composed Finite-State Systems

Systems are often presented as sets (parallel compositions) of communicating agents; the global state space of such a *composed system* is usually exponential in the size of the system presentation. This phenomenon is known as "state explosion", which is the main challenge in the design of efficient algorithms for verification of such systems.

2.4.1 Operations on Labelled Transition Systems

There are several possible ways how to define systems composed of components running in parallel and communicating with each other. Here, we describe operations of parallel composition and hiding as defined for example in [43], which were also used in [70].

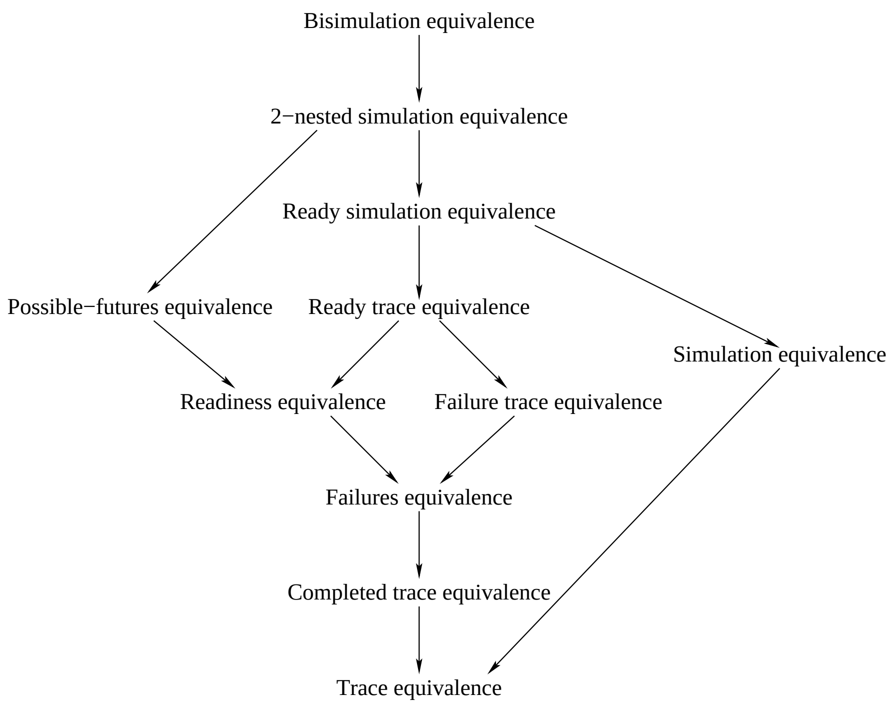
Bisimulation equivalence
2−nested simulation equivalence
Ready simulation equivalence
Possible−futures equivalence
Ready trace equivalence
Simulation equivalence
Readiness equivalence
Failure trace equivalence
Failures equivalence
Completed trace equivalence
Trace equivalence

Figure 1: The linear time – branching time spectrum

We reserve symbol τ for denoting a special action; it serves for renaming the usual (visible) actions whose identity we want to hide.

Remark. For the above defined behavioural relations, this τ is taken as a normal action (so "bisimilarity" means "strong bisimilarity", etc.).

Given an LTS $\mathcal{T} = (S, Act \cup \{\tau\}, \longrightarrow)$, $\tau \notin Act$, and $\mathcal{B} \subseteq Act$, by

$$hide\ \mathcal{B}\ in\ \mathcal{T}$$

we denote the LTS $\mathcal{T}_1 = (S, (Act - \mathcal{B}) \cup \{\tau\}, \longrightarrow_1)$ where $s \xrightarrow{a}_1 s'$ in $\mathcal{T}_1$ iff

- either $a \in (Act - \mathcal{B}) \cup \{\tau\}$ and $s \xrightarrow{a} s'$ in $\mathcal{T}$,
- or $a = \tau$ and $s \xrightarrow{b} s'$ in $\mathcal{T}$ for some $b \in \mathcal{B}$.

The *parallel composition* of LTSs $\mathcal{T}_1 = (S_1, Act_1 \cup \{\tau\}, \longrightarrow_1)$, $\mathcal{T}_2 = (S_2, Act_2 \cup \{\tau\}, \longrightarrow_2)$ (where $\tau \notin (Act_1 \cup Act_2)$) is the LTS

$$\mathcal{T}_1 \parallel \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2 \cup \{\tau\}, \longrightarrow)$$

representing the product of $\mathcal{T}_1$ and $\mathcal{T}_2$ in which $\mathcal{T}_1$ and $\mathcal{T}_2$ synchronize on the shared non-τ actions, i.e., where $(s_1, s_2) \xrightarrow{a} (s_1', s_2')$ iff

- either $a \in Act_1 \cup Act_2$ and for each $i \in \{1,2\}$ we have: if $a \in Act_i$ then $s_i \xrightarrow{a}_i s'_i$ and otherwise $s'_i = s_i$,
- or $a = \tau$ and $s_i \xrightarrow{\tau}_i s'_i$ and $s_{3-i} = s'_{3-i}$ for some $i \in \{1,2\}$.

Hence if $a \in Act_1 \cap Act_2$ then both components perform a-transitions simultaneously; otherwise just one component does.

It is obvious that $\|$ is associative and commutative with respect to isomorphism, and we freely write

$$\mathcal{T} = \mathcal{T}_1 \parallel \mathcal{T}_2 \parallel \cdots \parallel \mathcal{T}_n$$

for the parallel composition of $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$.

2.4.2 Models PC and PCH

In this subsection we consider only *finite* labelled transition systems, where the sets of states and actions are finite. By an *explicit finite-state system*, denoted FS, we mean a finite LTS together with its presentation which lists all states, actions and transitions. The size $|\mathcal{T}|$ of FS $\mathcal{T} = (S, Act, \longrightarrow)$ is $|S| + |Act| + |\longrightarrow|$.

By a system presented as *parallel composition without hiding*, denoted PC, we mean a presentation (of an LTS) in the form

$$\mathcal{T} = \mathcal{T}_1 \parallel \mathcal{T}_2 \parallel \cdots \parallel \mathcal{T}_n$$

where $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ are explicit finite-state systems, called *components* of PC $\mathcal{T}$; we also assume that their action alphabets do not contain τ. The size of PC $\mathcal{T}$ is $|\mathcal{T}| = |\mathcal{T}_1| + |\mathcal{T}_2| + \ldots + |\mathcal{T}_n|$. A state of $\mathcal{T}$, called a *global state*, is $E = (s_1, s_2, \ldots, s_n)$ where $s_1, s_2, \ldots, s_n$ are states of the components $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ respectively. We note that the number of (global) states of $\mathcal{T}$ can be exponential in $|\mathcal{T}|$.

Suppose that $Act_1, Act_2, \ldots, Act_n$ are action alphabets of $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ and note that if $E \xrightarrow{a} E'$ then *all* components $\mathcal{T}_i$ such that $a \in Act_i$ must perform an a-transition. We say that such components *participate* in the transition.

It can be easily shown (see [70]) that any system constructed from explicit finite-state systems by a finite number of applications of parallel composition and hiding can be transformed into an isomorphic system (of the same size) in the form

$$\textit{hide } \mathcal{B} \textit{ in } (\mathcal{T}_1 \parallel \mathcal{T}_2 \parallel \cdots \parallel \mathcal{T}_n)$$

where $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ are explicit finite-state systems. (We can use the fact that $(\textit{hide } \mathcal{B} \textit{ in } \mathcal{T}) \parallel \mathcal{T}'$ is isomorphic to $\textit{hide } \mathcal{B} \textit{ in } (\mathcal{T} \parallel \mathcal{T}')$ provided that actions of $\mathcal{T}'$ do not occur in $\mathcal{B}$, which can be ensured by renaming of actions in $\mathcal{B}$ with fresh names when necessary.) We call a system presented in the specified form a *parallel composition with hiding*, denoted PCH, and we define its size as

$|\mathcal{T}_1| + |\mathcal{T}_2| + \cdots + |\mathcal{T}_n| + |\mathcal{B}|$. PC is the special case of PCH where $\mathcal{B} = \emptyset$ and where components do not have τ actions.

A PCH $\mathcal{T}$ is *acyclic* iff (the graph of) every component of $\mathcal{T}$ is acyclic.

2.5 Process Rewrite Systems

Process Rewrite Systems (PRS) defined by Mayr in [64] provide a unified view of many formalisms presented in the following sections.

Process rewrite systems are defined as follows. Let $\mathit{Act} = \{a, b, c, \ldots\}$ be a countably infinite set of atomic actions and $\mathit{Var} = \{X, Y, Z, \ldots\}$ be a countably infinite set of process variables. Process terms are defined by the following abstract grammar

$$P \;::=\; \varepsilon \;\mid\; X \;\mid\; P_1.P_2 \;\mid\; P_1 \parallel P_2$$

where ε is the empty term, X is a process variable, and where '.' denotes sequential composition and '$\parallel$' parallel composition. Sequential composition is associative and parallel composition is associative and commutative. We always work with equivalence classes of terms modulo associativity of sequential composition and modulo associativity and commutativity of parallel composition. We also define that $\varepsilon.P = P.\varepsilon = P$ and $P \parallel \varepsilon = P$.

Process rewrite system is a finite set of rules Δ containing rules of the form $t_1 \xrightarrow{a} t_2$ where t_1 and t_2 are process terms and $a \in \mathit{Act}$ is an atomic action. Let $\mathit{Var}(\Delta)$ be the set of process variables occurring in Δ and let $\mathit{Act}(\Delta)$ be the set of atomic actions occurring in Δ.

Process rewrite system Δ produces a corresponding labelled transition system $(S, \mathit{Act}', \longrightarrow)$ where S is the set of process terms that contain only variables from $\mathit{Var}(\Delta)$, $\mathit{Act}' = \mathit{Act}(\Delta)$, and the transition relation is the smallest relation satisfying the following inference rules where t_1, t_2, t_1', t_2' are process terms:

$$\frac{(t_1 \xrightarrow{a} t_2) \in \Delta}{t_1 \xrightarrow{a} t_2}$$

$$\frac{t_1 \xrightarrow{a} t_1'}{t_1.t_2 \xrightarrow{a} t_1'.t_2} \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1 \parallel t_2 \xrightarrow{a} t_1' \parallel t_2} \qquad \frac{t_2 \xrightarrow{a} t_2'}{t_1 \parallel t_2 \xrightarrow{a} t_1 \parallel t_2'}$$

Note that $\mathit{Var}(\Delta)$ and $\mathit{Act}(\Delta)$ are finite. Since Δ is finite, the generated labelled transition system is finitely branching, which means that the branching-degree is finite in every state, however, it can be be arbitrarily high, i.e., it is possible that there is no finite constant depending only on Δ bounding the branching-degree in all states.

It is worth mentioning that process rewrite systems are not Turing powerful because for example the reachability problem is decidable for them [64].

Note also that there is no operator for non-deterministic choice ('+'), because nondeterminism can be encoded in the set of rules Δ which can contain more rules with the same term on the left side.

There can be defined different types of subclasses of process rewrite systems. At first we distinguish four classes of process terms:

1 – terms consisting of a single process variable (e.g., X),

$\mathcal{S}$ – terms consisting of ε, a single variable, or a sequential composition of process variables (e.g., $X.Y.Z$),

$\mathcal{P}$ – terms consisting of ε, a single variable, or a parallel composition of process variables (e.g., $X \parallel Y \parallel Z$),

$\mathcal{G}$ – any process terms without any restriction (e.g., $(X \parallel Y).Z$).

Obviously $1 \subsetneq \mathcal{S}$, $1 \subsetneq \mathcal{P}$, $\mathcal{S} \subsetneq \mathcal{G}$, and $\mathcal{P} \subsetneq \mathcal{G}$. Classes $\mathcal{S}$ and $\mathcal{P}$ are incomparable and $\mathcal{S} \cap \mathcal{P} = 1 \cup \{\varepsilon\}$.

Let $\alpha, \beta \in \{1, \mathcal{S}, \mathcal{P}, \mathcal{G}\}$ be classes of process terms such that $\alpha \subseteq \beta$. A system of type (α, β)-PRS is defined as a finite set of rules Δ where in every rewrite rule $(l \xrightarrow{a} r) \in \Delta$ the term l is from class α and $l \neq \varepsilon$ and the term r is from class β (and r can be ε).

The hierarchy of (α, β)-PRS models is depicted in Figure 2. Each model in the hierarchy has a name shown also in the figure and many of these (α, β)-PRS correspond to well-known classes of infinite state systems studied in the literature. A line from a higher model to a lower model means that the higher model is more general than the lower one. It is known that the hierarchy is strict with respect to bisimilarity [64].

The classes of process rewrite systems correspond to the following following formalisms:

FS – finite-state systems,

BPA – Basic Process Algebra [17], also called *context-free processes*,

BPP – Basic Parallel Processes [24],

PDA – Pushdown Automata, also called *pushdown processes* or *pushdown systems*,

PA – Process Algebra [14],

PN – Petri nets,

PRS – Process Rewrite Systems.

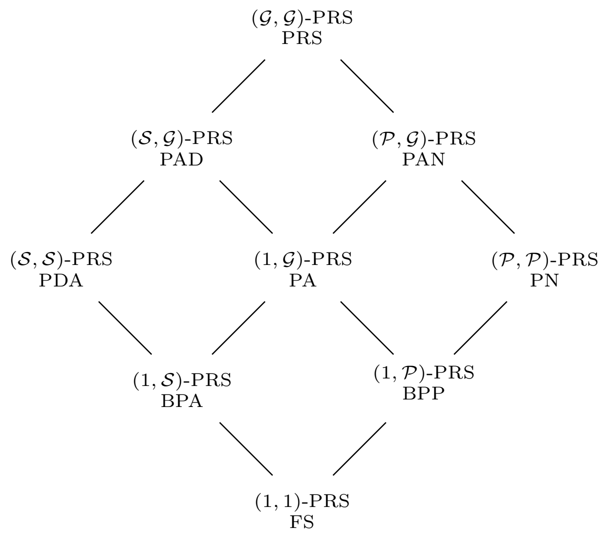
($\mathcal{G},\mathcal{G}$)-PRS
PRS
($\mathcal{S},\mathcal{G}$)-PRS
PAD
($\mathcal{P},\mathcal{G}$)-PRS
PAN
($\mathcal{S},\mathcal{S}$)-PRS
PDA
($1,\mathcal{G}$)-PRS
PA
($\mathcal{P},\mathcal{P}$)-PRS
PN
($1,\mathcal{S}$)-PRS
BPA
($1,\mathcal{P}$)-PRS
BPP
($1,1$)-PRS
FS

Figure 2: Hierarchy of process rewrite systems

Class PAD was introduced in [64] as the "smallest" common generalization of classes PDA and PA. Similarly class PAN was introduced in [63] as the "smallest" common generalization of classes PA and PN.

The following subsections describe classes BPA and BPP in more detail, and describe alternative (and more standard) definitions of processes in these classes, which are sometimes more appropriate when discussing some results.

2.5.1 Basic Process Algebra (BPA)

A *BPA system*, or *BPA* for short, can be viewed as a context-free grammar in Greibach normal form. Formally it is a triple $\Sigma = (V_\Sigma, Act_\Sigma, \Gamma_\Sigma)$, where V_Σ is a finite set of *variables* (nonterminals), Act_Σ is a finite set of *actions* (terminals) and $\Gamma_\Sigma \subseteq V_\Sigma \times Act_\Sigma \times V_\Sigma^*$ is a finite set of *rewrite rules*. We often use V, Act, Γ without subscripts when the underlying BPA is clear from context. We also write $X \xrightarrow{a} \alpha$ instead of $(X, a, \alpha) \in \Gamma$. A *BPA process* is a pair (α, Σ) where Σ is a BPA system and $\alpha \in V^*$; we write just α when Σ is clear from context. A BPA Σ gives rise to the LTS $\mathcal{S}_\Sigma = (V^*, Act, \longrightarrow)$ where $\longrightarrow$ is induced from the rewrite rules by the following (deduction) rule: if $X \xrightarrow{a} \alpha$ then $X\beta \xrightarrow{a} \alpha\beta$ for every $\beta \in V^*$.

2.5.2 Basic Parallel Processes (BPP)

We recall the standard definition of the class *Basic Parallel Processes* (BPP).

Given a set *Act* of atomic *actions*, usually denoted by $a, b, \ldots$, and a set *Var* of process *variables*, ranged over by $X, Y, \ldots$, the class of *BPP expressions* over *Act* and *Var* is defined by the following abstract grammar:

$$E \ ::= \ \mathbf{0} \ \mid \ X \ \mid \ a.E \ \mid \ E+E \ \mid \ E \parallel E$$

where $\mathbf{0}$ denotes the empty process, X stands for a process variable, and $a.\_$, $\_+\_$, $\_\parallel\_$ denote the operations of *action prefix* (for each $a \in \mathit{Act}$), *nondeterministic choice*, and *parallel composition*, respectively.

A *BPP system* Δ, also called a *BPP definition*, with a finite set of actions $\mathit{Act}(\Delta)$ and a finite set of variables $\mathit{Var}(\Delta) = \{X_1, X_2, \ldots, X_k\}$, is a finite family of (possibly recursive) equations:

$$\Delta = \{X_i \overset{\text{def}}{=} E_i \mid 1 \le i \le k\}$$

where each E_i is a BPP expression over $\mathit{Act}(\Delta)$ and $\mathit{Var}(\Delta)$. We stipulate that each occurrence of a variable in E_i is *guarded*, i.e., within the scope of an action prefix. (This guarantees that the transition system induced by the rules below is finitely branching.)

A *BPP process* is a pair (E, Δ) where Δ is a BPP system and E is a BPP expression over $\mathit{Act}(\Delta)$ and $\mathit{Var}(\Delta)$. When Δ is clear from context, we often write just E instead of (E, Δ), and *Act* and *Var* instead of $\mathit{Act}(\Delta)$ and $\mathit{Var}(\Delta)$, respectively.

Any BPP system Δ can be viewed as representing the (possibly infinite) LTS $\mathit{LTS}(\Delta)$, where the processes (E, Δ) are viewed as the states and where the transition relation is induced by the following SOS (structural operational semantics) rules:

$$\frac{}{a.E \xrightarrow{a} E} \qquad \frac{E \xrightarrow{a} E'}{E+F \xrightarrow{a} E'} \qquad \frac{F \xrightarrow{a} F'}{E+F \xrightarrow{a} F'}$$

$$\frac{E \xrightarrow{a} E'}{E \parallel F \xrightarrow{a} E' \parallel F} \qquad \frac{F \xrightarrow{a} F'}{E \parallel F \xrightarrow{a} E \parallel F'} \tag{1}$$

$$\frac{E \xrightarrow{a} E'}{X \xrightarrow{a} E'} \ \ ((X \overset{\text{def}}{=} E) \in \Delta)$$

Sometimes, it is convenient to consider BPP systems in the *normal form*, similar to the definition of BPA above. A BPP system in the normal form is a triple $\Delta = (V_\Delta, \mathit{Act}_\Delta, \Gamma_\Delta)$ defined almost the same way as BPA system above, except that the deduction rule for the associated LTS $\mathcal{S}_\Delta$ is different: if $X \xrightarrow{a} \alpha$ then $\gamma X \delta \xrightarrow{a} \gamma \alpha \delta$ for any $\gamma, \delta \in V^*$ (thus *any* occurrence of a variable can be rewritten, not just the first one). It is easy to observe that BPP processes α, β with the same Parikh image (i.e., containing the same number of occurrences of each variable) are bisimilar. Hence BPP processes can be read modulo commutativity of concatenation and interpreted as multisets of variables. It is well known folklore that a BPP system can be transformed to this normal form in polynomial time.

This also suggests to identify a BPP system Δ with a *BPP net*, a labelled Petri net in which each place corresponds to a variable and each transition corresponds to a rewrite rule (and thus has a unique input place).

Formally, a *BPP net* is a tuple $\Delta = (P_\Delta, \mathit{Tr}_\Delta, \textsc{pre}_\Delta, F_\Delta, \mathit{Act}_\Delta, l_\Delta)$ where P_Δ is a finite set of *places* (variables), Tr_Δ is a finite set of *transitions*, $\textsc{pre}_\Delta : \mathit{Tr}_\Delta \to P_\Delta$ is a function assigning an input place to each transition, $F_\Delta : (\mathit{Tr}_\Delta \times P_\Delta) \to \mathbb{N}$ is a *flow function*, Act_Δ is a finite set of *actions*, and $l_\Delta : \mathit{Tr}_\Delta \to \mathit{Act}_\Delta$ is a *labelling function*. We will use $P, \mathit{Tr}, \textsc{pre}, F, \mathit{Act}, l$ if the underlying BPP net is clear from context. We note that a transition $t \in \mathit{Tr}$ can be viewed as the rewrite rule $p \xrightarrow{a} \alpha$ where $\textsc{pre}(t) = p$ and $F(t, p')$ is the number of occurrences of p' in α, for each $p' \in P$.

A BPP process is thus, in fact, a *marking*, i.e. a function $M : P \to \mathbb{N}$ which associates a finite number of *tokens* to each place.

A transition t is *enabled* at marking M if $M(\textsc{pre}(t)) \geq 1$. An enabled transition t may fire from M, producing a marking M' defined by

$$M'(p) = \begin{cases} M(p) - 1 + F(t,p) & \text{if } p = \textsc{pre}(t) \\ M(p) + F(t,p) & \text{otherwise} \end{cases} .$$

This is denoted by $M \xrightarrow{t} M'$; the notation is extended to $M \xrightarrow{\sigma} M'$ for sequences $\sigma \in T^*$. We write $M \xrightarrow{\sigma}$ if $M \xrightarrow{\sigma} M'$ for some M'.

In the above sense, a BPP Δ gives rise to the LTS $\mathcal{S}_\Delta = (\mathcal{M}_\Delta, \mathit{Act}, \longrightarrow)$ where $\mathcal{M}_\Delta = \mathbb{N}^P$ is the set of all markings (of the respective BPP net), and $M \xrightarrow{a} M'$ iff there is some $t \in \mathit{Tr}$ such that $l(t) = a$ and $M \xrightarrow{t} M'$.

We often use symbols $\alpha, \beta, \ldots$ for both BPA processes and BPP processes, and $M_1, M_2, \ldots$ only for the latter.

We say that a BPA system Σ (a BPP net Δ) is *normed* iff $\alpha \longrightarrow^* \varepsilon$ for each state α of $\mathcal{S}_\Sigma$ ($\mathcal{S}_\Delta$). We use nBPA (nBPP) for normed BPA (normed BPP).

2.6 Complexity Classes

Some standard definitions from complexity theory are recalled in this subsection.

Most of complexity classes discussed in this thesis are standard and well known – LOGSPACE, PTIME, NP, PSPACE, EXPTIME, EXPSPACE. The class DP is probably less known. It is defined as follows: a problem P is in DP iff there are problems $P_1 \in$ NP and $P_2 \in$ coNP such the set of positive instances of P is the intersection of the sets of positive instances of P_1 and P_2.

When we talk about hardness or completeness of a problem, we mean hardness or completeness under logspace reductions, unless stated otherwise, i.e., when $\mathcal{C}$ is some arbitrary complexity class (e.g., PTIME, NP, PSPACE, $\mathsf{EXPTIME}$, etc.), a *problem* P is *$\mathcal{C}$-hard* iff each problem from $\mathcal{C}$ can be reduced to P by a logspace reduction. A *problem* P is *$\mathcal{C}$-complete* iff P is $\mathcal{C}$-hard and $P \in \mathcal{C}$.

In particular, a problem P is PTIME-hard if for each problem $P' \in \mathsf{PTIME}$ there is a logspace reduction from P' to P. Recall that if a problem P is PTIME-hard then it is unlikely that there exists an efficient parallel algorithm deciding P, unless $\mathsf{PTIME} = \mathsf{NC}$, where NC is the class of problems that can be solved by a parallel algorithm in polylogarithmic time (i.e., with time complexity $O(\log^k n)$ where k is a constant) using a polynomial number of processors. It is known that $\mathsf{LOGSPACE} \subseteq \mathsf{NC} \subseteq \mathsf{PTIME}$. It is an open question if any of these inclusions is proper, however, it holds for each PTIME-complete problem P that $P \notin \mathsf{NC}$ unless $\mathsf{NC} = \mathsf{PTIME}$. (See e.g. [37] for further details.)

When proving some hardness results, some reductions described in these proofs are in fact reductions to the *complements* of discussed problems. When *deterministic* complexity classes (such as PTIME, PSPACE, $\mathsf{EXPTIME}$) are considered, this distinction is not important, as follows from the following simple observation.

Observation 2.2 *A problem P is* PTIME*-hard (*PSPACE*-hard,* $\mathsf{EXPTIME}$*-hard) iff the complement of P is* PTIME*-hard (*PSPACE*-hard,* $\mathsf{EXPTIME}$*-hard).*

3 Finite-State Systems

This section gives an overview of results concerning deciding of behavioural equivalences on (explicitly given) finite-state state systems, which were published in [5]:

- Zdeněk Sawa, Petr Jančar: *Behavioural Equivalences on Finite-State Systems are PTIME-hard*, *Computing and Informatics*, Volume 24, Issue 5, pp. 513–528, Slovak Academy of Sciences, Institute of Informatics, 2005.

3.1 Motivation and State of the Art

Finite labelled transition systems can be viewed as classical nondeterministic finite automata (NFA) with all states defined as accepting. It is well known in classical language theory that checking language inclusion or equivalence for NFA is PSPACE-complete (cf., e.g., [45]), and this result can be easily applied also to the case where all states of an automaton are accepting. This implies PSPACE-hardness of deciding *trace inclusion* (*trace preorder*) and *trace equivalence* on finite-state systems.

On the other hand, there are polynomial-time algorithms for finite-state systems in the simulation-like part of the spectrum, for example there is an algorithm for deciding bisimilarity with running time $O(n \log n)$ [68]; we can refer, e.g., to [25] for a survey.

Balcázar, Gabarró and Sántha [15] showed that the problem of checking bisimilarity is PTIME-hard. Their paper shows a logspace reduction from (a special version of) the boolean circuit value problem, which is well-known to be PTIME-complete. Their reduction handles just bisimilarity; in particular, it does not show PTIME-hardness of other "simulation-like" equivalences (which are known to be in PTIME as well).

3.2 Summary of the Results

Paper [5] shows that deciding an *arbitrary relation* which subsumes bisimulation equivalence and is subsumed by trace preorder is PTIME-hard even for *acyclic* finite state systems:

Theorem 3.1 *For any relation $\mathcal{R}$ between bisimilarity and trace preorder, the following problem is* PTIME*-hard:*

Instance*: A finite acyclic labelled transition system and two of its states, p and q.*

Question*: Is $p\mathcal{R}q$?*

This result substantially extends the result of [15]. The value of this extension is primarily relevant for the problems in PTIME, i.e., in the simulation-like part of the spectrum, which can also comprise “not yet discovered” equivalences; of course, PTIME-hardness trivially follows for problems which are known to be PSPACE-hard (in the trace-like part). Notice also that problems considering only acyclic systems are in general easier than general case; for example deciding trace equivalence is still coNP-complete for acyclic finite state systems.

The proof in [5] proceeds by presenting a logspace reduction from the circuit value problem (in fact, it uses a less constrained version of circuit value problem than [15]).

Remark. A preliminary version of [5] appeared in [10].

3.3 Outline of the Proof

Theorem 3.1 is proved by a logspace reduction from the problem of monotone boolean circuit value, MCVP for short.

To define MCVP, we need some definitions. A *monotone boolean circuit* is a directed acyclic graph in which the nodes (also called *gates*) are either of in-degree zero (*input* gates) or of in-degree 2; there is exactly one node of out-degree zero (the *output* gate), otherwise the out-degree is not constrained. Each non-input gate is labelled either by conjunction $\wedge$ or by disjunction $\vee$. (No negation is used; thus we have monotonicity.) An *input of the circuit* is an assignment of boolean values (values from the set $\{0,1\}$) to input gates. Given an input, the circuit computes as expected: the (output) value of an input gate is given by the input, the value of a gate labelled with $\wedge$ (resp. $\vee$) is computed as conjunction (resp. disjunction) of values of its predecessors. The *output value of the circuit* is the value of the output gate.

See Figure 3 for an example of a monotone boolean circuit with an input; the corresponding computed values are also depicted.

The MCVP problem is defined as follows:

INSTANCE: A monotone boolean circuit with an input.

QUESTION: Is the output value 1 ?

The problem is well-known to be PTIME-complete (cf. e.g. [37]). We also recall that if P_1 is PTIME-hard and P_1 is logspace reducible to P_2 then P_2 is PTIME-hard as well.

We sketch a logspace algorithm which, given an instance of MCVP, constructs an LTS with two designated states p, q so that:

- if the output value of the circuit is 1 then $p \sim q$, and

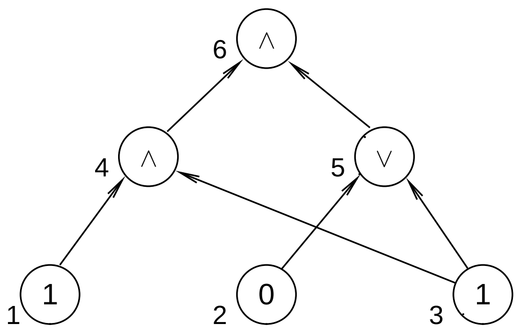

Figure 3: A monotone boolean circuit with computed values on gates (the gates are topologically ordered)

- if the output value is 0 then $\mathit{Traces}(p) \not\subseteq \mathit{Traces}(q)$.

So for any relation $\mathcal{R}$ between bisimilarity and trace preorder we have $p\mathcal{R}q$ iff the output value is 1. This will immediately imply Theorem 3.1.

Let us fix an instance of MCVP where the set of gates is $V = \{1, 2, \dots, n\}$. For technical reasons we assume that the gates are *topologically ordered*, i.e., for any edge (i, j) we have $i < j$; hence n is the output gate. (This assumption does not affect PTIME-completeness of MCVP, as can be seen from [37].) See Figure 3 for an example of topologically ordered gates.

We use function $t : V \longrightarrow \{0, 1, \wedge, \vee\}$ for denoting the types of gates (in the given instance of MCVP):

$$t(i) = \begin{cases} 0 & \text{if } i \text{ is an input gate with value } 0 \\ 1 & \text{if } i \text{ is an input gate with value } 1 \\ \wedge & \text{if } i \text{ is a gate labelled with } \wedge \\ \vee & \text{if } i \text{ is a gate labelled with } \vee \end{cases}$$

For every non-input gate i, we denote its (first and second) predecessors by $f(i), s(i)$ so that $f(i) < s(i)$ (and $s(i) < i$ due to the topological order). Let

$$v_i \in \{0, 1\}$$

denote the actual (computed) value on gate i, i.e.,

- if i is an input gate then $v_i = t(i)$,
- if i is a non-input gate, then v_i is computed from $v_{f(i)}$ and $v_{s(i)}$ using the operation indicated by $t(i)$.

For later use, we define inductively the words $W_i \in \{0, 1\}^*$, $i = 0, 1, \dots, n$:

$$W_0 = \varepsilon, W_{i+1} = v_{i+1} W_i \tag{2}$$

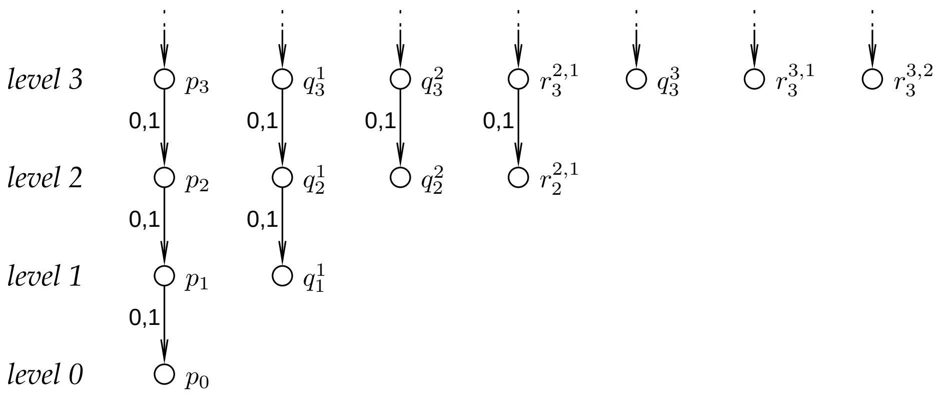

Figure 4: The states of Δ organized into levels

Thus W_n is the sequence of the actual (computed) values for all gates in the reversed order wrt the given topological order; W_i is the suffix of W_n of length i.

For concreteness, we can assume that the given input instance of MCVP is of the form

$$1 : d(1);\ 2 : d(2);\ \ldots;\ n : d(n)$$

where $d(i) = t(i)$ if $t(i) \in \{0, 1\}$ and $d(i)$ stands for $\langle t(i), f(i), s(i)\rangle$ if $t(i) \in \{\wedge, \vee\}$.

Given this instance of MCVP, we show a construction of LTS $\Delta = (S, Act, \longrightarrow)$, where $Act = \{0, 1\}$ and S is the union of the following sets:

- $\{p_i \mid 0 \le i \le n\}$,
- $\{q_i^j \mid 1 \le j \le i \le n\}$,
- $\{r_i^{j,k} \mid 1 \le k < j \le i \le n\}$.

We organize states in S into *levels*. *Level* i (where $0 \le i \le n$) contains all states with the same lower index i, i.e.,

$$\{p_i\} \cup Q_i$$

where

$$Q_i = \{q_i^j \mid 1 \le j \le i\} \cup \{r_i^{j,k} \mid 1 \le k < j \le i\}$$

as depicted in Figure 4 (already with *some* transitions).

For ease of presentation, we call q_i^j *successful* if $v_j = 1$, and *unsuccessful* if $v_j = 0$; similarly, state $r_i^{j,k}$ is *successful* if *both* $v_j = 1$ and $v_k = 1$, and is *unsuccessful* otherwise. We denote the set of all successful (unsuccessful) states on level i by $Succ_i$ ($Unsucc_i$). Hence

$$Q_i = Succ_i \cup Unsucc_i$$

Further, let I denote the identity relation $I = \{(s, s) \mid s \in S\}$.

The way we shall construct the transition relation of LTS Δ will guarantee the following property. (By W_i we refer to (2).)

CRUCIAL PROPERTY:

- the set
$$I \cup \{(p_i, q) \mid 0 \le i \le n, q \in \mathit{Succ}_i\}$$
is a bisimulation (therefore $p_i \sim q$ for each $q \in \mathit{Succ}_i$) ; and

- if $q \in \mathit{Unsucc}_i$ then
$$W_i \in \mathit{Traces}(p_i) \setminus \mathit{Traces}(q)$$
(therefore $\mathit{Traces}(p_i) \not\subseteq \mathit{Traces}(q)$) .

Hence p_n and q_n^n can serve as the two announced distinguished states, with the required property that

- $p_n \sim q_n^n$ if $v_n = 1$ (i.e., if q_n^n is successful), and
- $\mathit{Traces}(p_n) \not\subseteq \mathit{Traces}(q_n^n)$ otherwise.

Transitions of Δ are constructed in such a way that transitions going from states on level i go only to states on level $i - 1$, which ensures that the constructed LTS Δ will be acyclic.

Level i naturally corresponds to gate i and the actual transitions going from level i to level $i - 1$ depend just on $t(i)$ and $f(i)$, $s(i)$ (when $t(i) \in \{\wedge, \vee\}$). See [5] for technical details of the construction of transitions and the proof of correctness of the whole construction.

As an illustration of the construction, see Figure 5 for the LTS constructed for the boolean circuit depicted in Figure 3. (For better clarity, only states that are reachable from states p_6 and q_6^6 are depicted.)

We now sketch why the described reduction can be performed in logarithmic space. The algorithm performing the reduction has an instance of MCVP as its input, and outputs the states and transitions of LTS Δ in a systematic stepwise manner. The algorithm can be bound to use only a fixed number of variables (such as i, j, k) with values no bigger than n (the number of gates); these values can be stored in $O(\log n)$ bits (and the size of the input is clearly bigger than n).

Remark. LTS Δ contains $O(n^3)$ states and also $O(n^3)$ transitions (the number of possible transitions leaving a given state is bounded, independently on n). It can be easily proved by induction that a state of the form $r_i^{j,k}$ can be reachable from p_n or q_n^n only if there is $i' \in V$, such that $t(i') = \wedge$, $s(i') = j$ and $f(i') = k$. There is at most $O(n)$ pairs j, k, where such i' exists, and it is possible to test

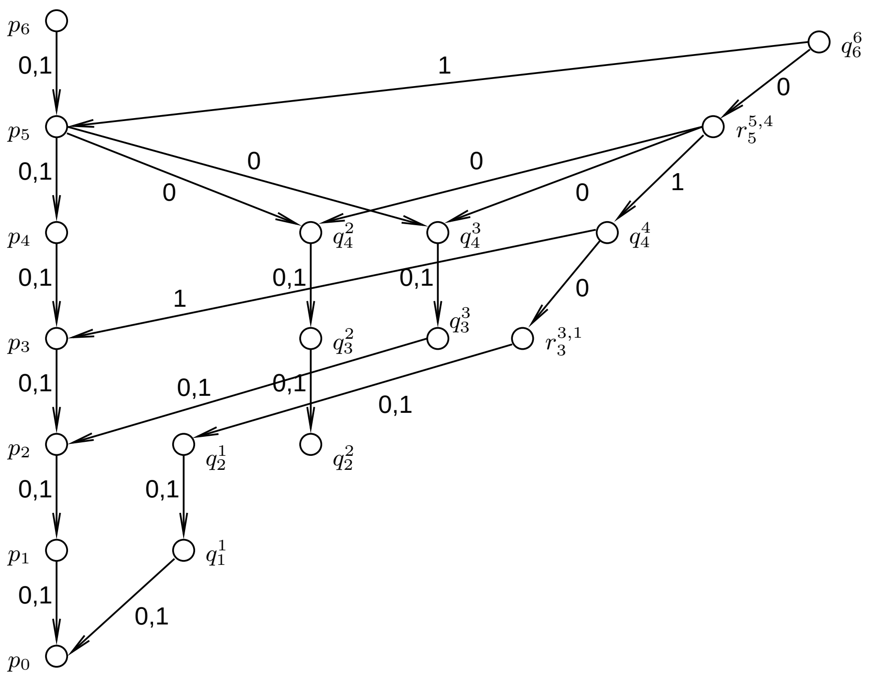

Figure 5: An example of the constructed LTS

for given j, k the existence of such i' in a logarithmic space. So we could add to Δ only those states $r_i^{j,k}$, where such i' exists. This way we can reduce the number of states (and transitions) to $O(n^2)$.

4 Composed Finite-State Systems

This section describes results concerning deciding behavioural equivalences on systems that are composed of communicating finite-state components. These results were published in [3]:

- Zdeněk Sawa, Petr Jančar: *Hardness of equivalence checking for composed finite-state systems*, *Acta Informatica*, Volume 46, Issue 3, pp. 169–191, Springer, 2009.

4.1 Motivation and State of the Art

When we consider *composed finite-state systems*, i.e., systems that are presented as sets (parallel compositions) of communicating finite-state components, the global state space of such composed systems is usually exponential in the size of the system presentation. This phenomenon is known as the "state explosion" and it is the main challenge in the design of efficient algorithms for verification of such systems, since the straightforward approach, where the global state space is explicitly constructed and the behavioural relation is decided on the resulting system, requires exponential space.

It is natural to ask if some more efficient algorithms exist in special cases. For example, Groote and Moller [39] have shown that if the components of the system can perform actions independently and there is no communication between them then the bisimulation equivalence (and some other equivalences that satisfy certain axioms) can be decided in polynomial time. As one may expect, the problem becomes harder when communication between components is allowed. Rabinovich [70] considered a general model of composed systems which can be called *Parallel Composition with Hiding* (PCH); the components (are forced to) synchronize via shared actions, and the identity of some actions can be "hidden", which refers to replacing with a special action τ. Rabinovich proved only PSPACE-hardness in general, though he showed EXPSPACE-completeness for trace equivalence and mentioned EXPTIME-completeness for bisimilarity. He formulated the EXPTIME-hardness conjecture for PCH and any relation between bisimilarity and trace preorder.

Laroussinie and Schnoebelen [58] approved the EXPTIME-hardness conjecture for all relations between bisimilarity and simulation preorder, even for composed systems with no hiding, denoted PC. It is hardly possible to extend EXPTIME-hardness for PC to all relations in the spectrum-range since trace equivalence is in PSPACE for this model, as was proved in [73]; see also [78] for results for other types of "trace-like" equivalences and composed systems. But it is also not clear how the result of [58] could be extended using hiding.

4.2 Overview of the Results

In [3], Rabinovich's conjecture was approved in the whole, deriving EXPTIME-hardness for PCH and any relation between bisimilarity and trace preorder. (A preliminary version of this proof appeared in [9].)

Another results in [3] concern with acyclic PC and PCH, which were also considered in [70]. A new alternative (simpler) reduction is provided that has improved the lower bound (of NP-hardness and coNP-hardness) to DP-hardness. Also PSPACE-hardness for acyclic PC and any relation between bisimilarity and simulation preorder has been derived. The paper [3] also provides an overview of known relevant results concerning equivalence checking on composed finite-state systems.

4.3 Outline of the Proof for (General) PCH

The EXPTIME-hardness proof is achieved by a reduction from the acceptance problem of alternating LBAs (linear bounded automata), denoted ALBA-ACCEPT. An important ingredient of the proof is a new proof of PTIME-hardness for *explicit* finite-state systems (given by listing the whole state space) and any relation between bisimilarity and trace preorder; this is achieved by a reduction from AGP, a problem on alternating graphs. The same result was already shown in [5] even for acyclic finite-state systems, as described in Section 3; however it is not clear, how the construction from [5] could be generalized to prove EXPTIME-hardness, and so the use of the new alternative construction was necessary in [3].

For describing the behaviour of the composed system arising by the reduction from an instance of ALBA-ACCEPT in the EXPTIME-hardness proof, it was also useful to introduce a "reactive" version of linear bounded automata as an intermediate model.

These reactive LBAs can be easily modeled not only by PCH but also by other models of composed systems, like, e.g., labelled 1-safe Petri nets; the EXPTIME-hardness result is thus carried over to them as well.

Let C be a fixed type (class) of systems (such as FS, PC, or PCH) and $\mathcal{R}$ some fixed binary relation between bisimilarity and trace preorder (i.e., $\sim\subseteq \mathcal{R} \subseteq\sqsubseteq_{tr}$) defined on states of LTSs. By C-REL$_{\mathcal{R}}$ we denote the following problem:

INSTANCE: Presentation of an LTS $\mathcal{T}$ of type C, with two distinguished states s and s'.

QUESTION: Is $(s, s') \in \mathcal{R}$?

The problems considered in [3] are FS-REL$_{\mathcal{R}}$, PC-REL$_{\mathcal{R}}$, PCH-REL$_{\mathcal{R}}$, APC-REL$_{\mathcal{R}}$, APCH-REL$_{\mathcal{R}}$. In APC-REL$_{\mathcal{R}}$ and APCH-REL$_{\mathcal{R}}$ (compared to PC-REL$_{\mathcal{R}}$ and PCH-REL$_{\mathcal{R}}$) the instances are restricted to acyclic systems.

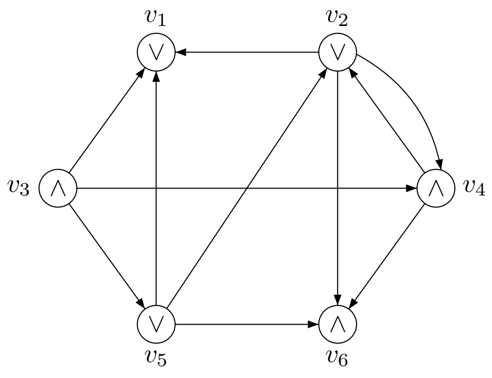

Figure 6: Alternating graph

It is also worth to note that the systems constructed in the reductions are in a special form: a PCH $\mathcal{T}$ is *centralized* iff it has a special *control component* $\mathcal{T}_c$ such that at most two components participate in each transition and $\mathcal{T}_c$ is always one of them.

Being inspired by the acceptance problem for ALBA (Alternating Linear Bounded Automata), and the related (alternating) graphs with configurations as nodes, we first define *Alternating Graph Problem* (AGP), a variant of the well known alternating reachability problem [46], which is PTIME-complete. Then we show a logspace reduction which, given an instance of AGP, constructs an FS $\mathcal{T}$ with two distinguished states s, s' so that:

- $s \sim s'$ if the answer to the AGP instance is NO,
- $s \not\sqsubseteq_{tr} s'$ if the answer to the AGP instance is YES.

This implies that FS-REL$_{\mathcal{R}}$ is PTIME-hard for any relation $\mathcal{R}$ such that $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$ (Theorem 4.4). Note that we reduce AGP to the complement of FS-REL$_{\mathcal{R}}$ and then use Observation 2.2.

To define *Alternating Graph Problem (*AGP*)* formally, we need some auxiliary definitions.

An *alternating graph* is a finite directed graph where each node is labelled either as *conjunctive* (universal) or as *disjunctive* (existential). Formally it is a structure $G = (V, E, t)$ where V is a finite set of *nodes*, $E \subseteq V \times V$ is a set of *edges*, and $t : V \to \{\wedge, \vee\}$ is a *node-labelling* function partitioning V into the sets of conjunctive and disjunctive nodes.

See Figure 6 for an example of an alternating graph.

We use $\sigma(v)$ to denote the set of *successors* of a node v, i.e.,

$$\sigma(v) = \{v' \in V \mid (v, v') \in E\}.$$

Each conjunctive node v ($t(v) = \wedge$) with $\sigma(v) = \emptyset$ is called *accepting*, each disjunctive node v ($t(v) = \vee$) with $\sigma(v) = \emptyset$ is called *rejecting*.

For example, v_6 is accepting and v_1 is rejecting in Figure 6.

We now define the set Succ_G (or *Succ* when G is clear from context) of *successful nodes* (i.e. those from which accepting nodes are "alternation-reachable"). *Succ* is defined as the *least* subset of V satisfying (for each $v \in V$):

- if $t(v) = \wedge$ and $\sigma(v) \subseteq \mathit{Succ}$ then $v \in \mathit{Succ}$,
- if $t(v) = \vee$ and $\sigma(v) \cap \mathit{Succ} \neq \emptyset$ then $v \in \mathit{Succ}$.

We note that each accepting node is successful (belongs to *Succ*) and each rejecting node is unsuccessful (i.e., not successful).

For example, for the alternating graph in Figure 6 we have $\mathit{Succ} = \{v_2, v_4, v_5, v_6\}$.

Now we define *Alternating graph problem* (Agp):

Instance: An alternating graph $G = (V, E, t)$ and a node $v \in V$.

Question: Is v successful?

Remark. In the alternating reachability problem as defined in [46] it is required that there is exactly one accepting node in G and also no loops are allowed.

Problem Agp is PTIME-hard since it is a generalization of the alternating reachability problem which is known to be PTIME-hard (see for example [46] for a proof). It is clear from the given inductive definition that *Succ* can be computed in polynomial time (measured in the size of the given G), and Agp is thus in PTIME. Hence we have the following fact.

Fact 4.1 Agp *is* PTIME-*complete.*

Given an alternating graph $G = (V, E, t)$, with a rejecting node z, we aim at constructing an LTS $\mathcal{T}_G = (V, \mathit{Act}, \longrightarrow)$ (where the states correspond to the nodes of G) so that $z \sim v$ for each $v \in V - \mathit{Succ}_G$ and $z \not\sqsubseteq_{tr} v$ for each $v \in \mathit{Succ}_G$.

See [3] for a detailed description of the construction of $\mathcal{T}_G$. It is important that this construction can be done in a logarithmic space and that transitions going from a state corresponding to a gate v are fully determined only by $t(v)$ and $\sigma(v)$, which is important for the generalization of this construction in the proof of EXPTIME-hardness.

As an example, LTS $\mathcal{T}_G$ constructed for the graph G in Figure 6 is depicted in Figure 7.

The correctness of the construction is ensured by two following propositions (see [3] for their proofs) and the fact that $\mathcal{T}_G$ is constructed in such a way that $\mathit{Traces}(v) = \mathit{Act}^*$ for each unsuccessful v.

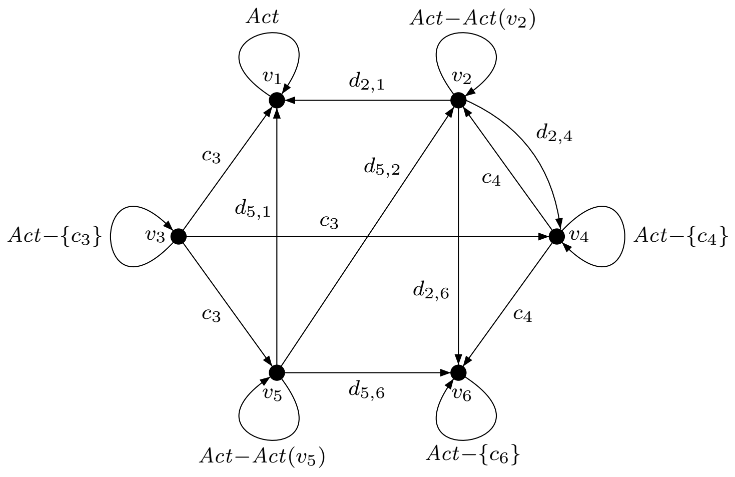

Figure 7: Example of $\mathcal{T}_G$ (with some transitions omitted)

Proposition 4.2 *If $v_1, v_2 \in V$ are unsuccessful in G then $v_1 \sim v_2$ in $\mathcal{T}_G$.*

Proposition 4.3 *For each G there is (a fixed) $w \in Act^*$ such that $w \notin Traces(v)$ in $\mathcal{T}_G$ for any $v \in Succ_G$.*

Theorem 4.4 Fs-Rel$_\mathcal{R}$ *is* PTIME*-hard for any $\mathcal{R}$ such that $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$.*

Proof. Given an alternating graph G and a distinguished node x (i.e., an instance of Agp), we can choose (or add) a rejecting node z and construct $\mathcal{T}_G$ (by the above logspace construction). Proposition 4.2 guarantees that if $x \notin Succ_G$ then $z \sim x$ and thus $(z, x) \in \mathcal{R}$. Proposition 4.3 guarantees that if $x \in Succ_G$ then $z \not\sqsubseteq_{tr} x$ and thus $(z, x) \notin \mathcal{R}$. □

The proof of EXPTIME-hardness of Pch-Rel$_\mathcal{R}$ for all relations $\mathcal{R}$, such that $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$, is based on a description of a logspace reduction from the problem Alba-accept, the problem whether a given alternating linear bounded automaton accepts a given word, which is known to be EXPTIME-complete [23]. In fact, we also consider so called reactive linear bounded automata (RLBAs) as a technically convenient intermediate step, and derive EXPTIME-completeness for Rlba-Rel$_\mathcal{R}$ as well.

Let (A, w) be an instance of Alba-accept where A is an ALBA and $w \in \Sigma^*$ its input (where $|w| = n$). ALBA A and $n \in \mathbb{N}$ determine the alternating graph

$$G_{(A,n)} = (V, E, t)$$

where V corresponds to the set of all configurations of A of size n, E represents possible transitions, and t labels the nodes according to the types of the corresponding configurations. Accepting (resp. rejecting) configurations of A are accepting (resp. rejecting) nodes in $G_{(A,n)}$.

If we consider $G_{(A,n)}$ (of exponential size) and then apply the previously described construction for explicitly given finite-state systems, we obtain an (explicit) FS, denoted $\mathcal{T}_A$. Both the set of nodes of $G_{(A,n)}$ and the set of states of $\mathcal{T}_A$ coincide with $\mathit{Conf}_{(A,n)}$, where $\mathit{Conf}_{(A,n)}$ denotes the set of all configurations of A of length n. If α_{rej} is a rejecting configuration in $\mathit{Conf}_{(A,n)}$ and $\alpha_{ini}(w)$ is the initial configuration of A with input w, then we have (in $\mathcal{T}_A$):

- $\alpha_{rej} \not\sqsubseteq_{tr} \alpha_{ini}(w)$ when A accepts w;
- $\alpha_{rej} \sim \alpha_{ini}(w)$ when A does not accept w.

The aim is, when given (A, w), to construct a PCH denoted $M_{(A,n)}$ which will represent ("realize") $\mathcal{T}_A$; moreover, we aim at a construction of $M_{(A,n)}$ that can be done in logarithmic space. (Note that the logspace construction will guarantee that the size of $M_{(A,n)}$ is polynomial in $|(A, w)|$.)

We know that a PCH can easily represent an LTS of exponential size; nevertheless a technical problem is that not only the number of states in $\mathcal{T}_A$ (i.e., the cardinality of $\mathit{Conf}_{(A,n)}$) is exponential but it is the case also for the alphabet $\mathit{Act}_{(A,n)}$ of $\mathcal{T}_A$.

To handle the alphabet-cardinality problem, we choose some straightforward (injective) encoding

$$\mathit{repr} : \mathit{Act}_{(A,n)} \to \{0,1\}^m \tag{3}$$

which encodes all elements of $\mathit{Act}_{(A,n)}$ by $0,1$-*strings of the same length* m $(\forall a \in \mathit{Act}_{(A,n)} : |\mathit{repr}(a)| = m)$ polynomial in $|(A, w)|$. The details of *repr* are not important. It is sufficient to require that decoding, including checking if $x \in \{0,1\}^m$ is in the range of *repr*, can be computed easily, which means that it is performable by the reactive LBA $B_{(A,n)}$ described later.

The LTS represented by $M_{(A,n)}$ will be finer than $\mathcal{T}_A$, having more states, but its action alphabet will be just $\{0, 1, \tau\}$. For each state α of $\mathcal{T}_A$ there will be a corresponding global state $\mathit{corresp}(\alpha)$ of $M_{(A,n)}$, and in the LTS represented by $M_{(A,n)}$ the following will be ensured:

- $\mathit{corresp}(\alpha) \sim \mathit{corresp}(\alpha')$ iff $\alpha \sim \alpha'$ in $\mathcal{T}_A$;
- $\mathit{corresp}(\alpha) \not\sqsubseteq_{tr} \mathit{corresp}(\alpha')$ iff $\alpha \not\sqsubseteq_{tr} \alpha'$ in $\mathcal{T}_A$.

Each transition $\alpha \xrightarrow{a} \alpha'$ in $\mathcal{T}_A$ will be "simulated" by a sequence of transitions $\mathit{corresp}(\alpha) \xrightarrow{u} \mathit{corresp}(\alpha')$ performed by $M_{(A,n)}$ where u arises from the sequence $\mathit{repr}(a) \in \{0,1\}^m$ by a suitable "padding" with (occurrences of) action τ; moreover, for all transitions $\alpha \xrightarrow{a} \alpha'$, the corresponding sequences u will have the same length, i.e. $|u| = \ell$ for a constant $\ell > m$.

It is technically convenient to specify the intended behaviour of $M_{(A,n)}$ by means of a special LBA, called a reactive LBA, and then show how reactive LBAs can be naturally implemented by PCH.

We can imagine that each step ($\alpha \vdash \alpha'$) in computations of an LBA also comprises "emitting" action τ; so each computation can be "observed" as a sequence of τ-actions by an "external observer". A *reactive linear bounded automaton*, an *RLBA*, is an LBA which is moreover equipped with a set Act of non-τ actions ($\tau \notin Act$); each step now emits either τ or $a \in Act$. For technical convenience, we require that the transitions emitting a non-τ action depend only on the current control state and do not change the tape nor the head position. RLBAs serve us for describing behaviours, not for accepting (rejecting) inputs; therefore we do not need an input alphabet nor accepting/rejecting states in the following formal definition.

An RLBA is a tuple $B = (Q_c, Q_r, \Gamma, Act, \delta, R)$, where Q_c and Q_r are finite sets of *computational control states* and *reactive control states*, respectively ($Q_c \cap Q_r = \emptyset$), Γ is a tape alphabet, Act is a finite set of actions, $\tau \notin Act$, $\delta \subseteq Q_c \times \Gamma \times Q \times \Gamma \times \{-1, 0, +1\}$, where $Q = Q_c \cup Q_r$, is the set of *computational transitions*, and $R \subseteq Q_r \times (Act \cup \{\tau\}) \times Q$ is the set of *reactive transitions.* Configurations and the successor relation $\alpha \vdash_B \alpha'$ are defined as in the case of LBAs (where $(q, a, q') \in R$ is understood as if $(q, x, q', x, 0) \in \delta$ for all $x \in \Gamma$).

Given $n \in \mathbb{N}$, RLBA B determines the LTS

$$\mathcal{T}(B, n) = (Conf_{(B,n)}, Act \cup \{\tau\}, \longrightarrow)$$

where $Conf_{(B,n)}$ is the set of all configurations of B of size n, and $\longrightarrow$ contains a transition $(q, w, i) \xrightarrow{a} (q', w', i')$ iff

- either $q \in Q_c$, $(q, w, i) \vdash_B (q', w', i')$ and $a = \tau$,
- or $q \in Q_r$, $(q, a, q') \in R$, $w = w'$ and $i = i'$.

See [3] for a detailed description of a construction of an RLBA $B_{(A,n)}$ for a given ALBA A with an input word w.

The construction can be done in a logarithmic space and shows that for the problem Rlba-Rel$_{\mathcal{R}}$

Instance: An RLBA B and two configurations α, α', $|\alpha| = |\alpha'|$.

Question: Is $(\alpha, \alpha') \in \mathcal{R}$ in $\mathcal{T}(B, |\alpha|)$?

we have the following theorem:

Theorem 4.5 Rlba-Rel$_{\mathcal{R}}$ *is* EXPTIME-*hard for any* $\mathcal{R}$ *such that* $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$.

An RLBA can be implemented by a PCH in a straightforward way, similarly as, e.g., Rabinovich [70] did for LBA (one component models a control unit together with position of a head and each tape cell is modelled by a separate component that stores the content of the cell), see [3] for the details.

Lemma 4.6 *Given an RLBA B and $n \in \mathbb{N}$, logarithmic workspace is sufficient to construct a centralized PCH $P_{(B,n)}$ which represents an LTS isomorphic to $\mathcal{T}(B,n)$.*

From Theorem 4.5 we thus get the following theorem.

Theorem 4.7 *The problem* PCH-REL$_{\mathcal{R}}$ *is* EXPTIME*-hard for any $\mathcal{R}$ such that $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$ even for centralized PCH.*

Remark. The construction can be also used for similar proofs (of EXPTIME-hardness) for other types of composed systems which use other means of synchronization, as, e.g., labelled 1-safe Petri nets.
The lower bound of EXPTIME-hardness can not be improved in general; this follows from the results surveyed in Subsection 4.5. ("Simulation-like" equivalences are in EXPTIME.)
We also note that hiding is crucial. We can not hope for general EXPTIME-hardness in the case of PC since the trace preorder and "trace-like" equivalences are in PSPACE for them (see Subsection 4.5).

4.4 Outline of the Proofs for Acyclic PC and PCH

4.4.1 DP-hardness for Acyclic PCH

Problem APCH-REL$_{\mathcal{R}}$ (for acyclic PCH) is stated in [70] as NP-hard and coNP-hard for each $\mathcal{R}$, $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$. There is a proof of coNP-hardness and it is mentioned that showing NP-hardness is similar (though it is, in fact, a bit more complicated); also a modification for coNP-hardness of APC-REL$_{\mathcal{R}}$ (without hiding) is suggested. In [3], a different (simpler) construction was shown, which allows us to derive all the mentioned cases as well as DP-hardness for APCH-REL$_{\mathcal{R}}$.

Theorem 4.8 *For any relation $\mathcal{R}$ such that $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$:*

- APCH-REL$_{\mathcal{R}}$ *is* DP*-hard for acyclic and centralized PCH.*
- APC-REL$_{\mathcal{R}}$ *is* coNP*-hard for acyclic and centralized PC.*

Proof idea. NP-hardness of APCH-REL$_{\mathcal{R}}$ can be shown by a reduction from an NP-complete problem called SHUFFLE, which is defined as follows. Given words $u, v \in \Sigma^*$, by *shuffle*(u,v) we denote the result of merging, or interleaving, i.e., the set of words of the form $u_1 v_1 u_2 v_2 \cdots u_n v_n$ where u_i, v_i are (possibly empty) words from Σ^* such that $u = u_1 u_2 \cdots u_n$ and $v = v_1 v_2 \ldots v_n$. The operation can be naturally generalized to languages, *shuffle*$(L_1, L_2) =$

$\bigcup_{u\in L_1, v\in L_2} \mathit{shuffle}(u,v)$, and the use of $\mathit{shuffle}(w_1, w_2, \dots, w_k)$ with more arguments has the obvious meaning. The following problem is known to be NP-complete [62, 81]:

Problem: SHUFFLE

INSTANCE: Words $w_1, w_2, \dots, w_k \in \Sigma^*$ and $w \in \Sigma^*$ such that $|w_1| + |w_2| + \dots + |w_k| = |w|$ (for some finite alphabet Σ).

QUESTION: Is it true that $w \in \mathit{shuffle}(w_1, w_2, \dots, w_k)$?

Remark. The constructions using SHUFFLE in this proof are quite simple compared to the constructions in [70], which use reductions from SAT.

Given an instance $w_1, w_2, \dots, w_k, w \in \Sigma^*$ of SHUFFLE, a pair of acyclic centralized PCH $\mathcal{T}$, $\mathcal{T}'$ with initial global states E_0 and E_0' can be constructed (in a logarithmic space) so that:

- $E_0 \sim E_0'$ when $w \in \mathit{shuffle}(w_1, w_2, \dots, w_k)$ (the SHUFFLE-answer is YES),
- $E_0 \not\sqsubseteq_{tr} E_0'$ otherwise (the SHUFFLE-answer is NO).

A proof of coNP-hardness of APC-REL$_{\mathcal{R}}$ (using no hiding) is similar, as well as the proof of DP-hardness of APCH-REL$_{\mathcal{R}}$, which uses reduction from the following DP-complete problem:

Problem: SHUFFLE-NONSHUFFLE

INSTANCE: Words $u_1, u_2, \dots, u_k, u \in \Sigma^*$ and $v_1, v_2, \dots, v_k, v \in \Delta^*$, where $\Sigma \cap \Delta = \emptyset$.

QUESTION: Is it true that $u \in \mathit{shuffle}(u_1, u_2, \dots, u_k)$ and $v \notin \mathit{shuffle}(v_1, v_2, \dots, v_k)$?

Again, see [3] for a detailed description of the constructions and proofs of their correctness. □

Due to the following proposition APC-REL$_{\mathcal{R}}$ is not NP-hard in general unless NP = coNP.

Proposition 4.9 APC-REL$_{\sqsubseteq_{tr}}$ *is in* coNP.

Proof. Given two acyclic PC P_1, P_2 (and their initial states), for showing that $P_1 \not\sqsubseteq_{tr} P_2$ it is sufficient to guess a trace w (of size $|P_1|$ at most) and verify that w is enabled in P_1 but disabled in P_2. To find out if a given $w = a_1 a_2 \dots a_n$ is enabled in a given PC (with no τ-actions and no hiding), by applying the usual subset construction for nondeterministic finite automata to the components we can successively represent all global states reachable by a_1, $a_1 a_2$, $a_1 a_2 a_3$, $\dots$ (as used, e.g., in [78]). □

It also seems unlikely that the lower bound of DP-hardness for APCH-REL$_{\mathcal{R}}$ can be much improved, as the next proposition shows.

Proposition 4.10 APCH-REL$_{\sqsubseteq_{tr}}$ *is in* Π_2^p *(in the polynomial hierarchy).*

Proof. Given two acyclic PCH P_1, P_2 (and their initial states), $P_1 \sqsubseteq_{tr} P_2$ means that for every trace w (of size $|P_1|$ at most), which is enabled in P_1, there is a sequence of global states in P_2 which shows that w is enabled in P_2. □

Nevertheless, at least for "simulation-like" relations we can derive PSPACE-hardness, even with no hiding, as the next subsubsection shows.

4.4.2 PSPACE-hardness of Simulation-like Relations on Acyclic PC

Theorem 4.11 *Problem* APC-REL$_{\mathcal{R}}$ *is* PSPACE*-hard for any* $\mathcal{R}$ *between bisimulation equivalence and simulation preorder (i.e.,* $\sim \subseteq \mathcal{R} \sqsubseteq_{sim}$*), even when restricted to (acyclic and) centralized PC.*

Proof idea. A reduction from the well-known PSPACE-complete problem QBF (truth of quantified boolean formulas) is used:

INSTANCE: $\varphi = \exists x_1 \forall x_2 \cdots \exists x_{n-1} \forall x_n F(x_1, x_2, \ldots, x_n)$ (where n is even).

QUESTION: Is φ true?

We assume that F is in CNF, i.e., in the form $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ where each *clause* $C_j = \ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3}$ contains exactly three *literals* (a literal being x_i or $\neg x_i$).

Roughly speaking, the reduction implements the following game:

1. Attacker and Defender alternately assign boolean values to variables x_1, $x_2, \ldots, x_n$ (in this order).
2. After the assignment, Defender chooses a clause C_j.
3. If some of literals $\ell_{j,1}, \ell_{j,2}, \ell_{j,3}$ is true under the assignment, Attacker wins; if not, Defender wins.

It is obvious that φ is true iff Attacker has a winning strategy. This is an example of an application of so called "Defender's Choice technique" which has been used for similar results (see [47] for a recent use). □

The PSPACE-hardness lower bound in Theorem 4.11 cannot be improved in general, as the following proposition shows.

Proposition 4.12 APCH-REL$_{\sim}$ *and* APCH-REL$_{\sqsubseteq_{sim}}$ *are in* PSPACE.

Proof. All plays of the (bi)simulation game on APCH P_1, P_2 have length (i.e., the number of rounds) at most $|P_1|$. They can be naturally organized in a tree which can be examined in polynomial space (by using the depth-first search), by which the player who has a winning strategy is determined. □

		$\sim$	$\sqsubseteq_{sim}$	$\sqsubseteq_{tr}$
	upper	PTIME (a)	PTIME (a)	PSPACE (b)
FS				PSPACE-hard (c)
	lower	PTIME-hard (d)		
	upper	PSPACE (e)	PSPACE (e)	coNP(f)
APC		PSPACE-hard (g)		
	lower	coNP-hard (h)		
	upper	PSPACE (e)	PSPACE (e)	Π_2^p (i)
APCH		PSPACE-hard (g)		
	lower	DP-hard (h)		
	upper	EXPTIME (j)	EXPTIME (j)	PSPACE (k)
PC		EXPTIME-hard (l)		
	lower	PSPACE-hard (m)		
	upper	EXPTIME (j)	EXPTIME (j)	EXPSPACE (n)
PCH		EXPTIME-hard (l)		EXPSPACE-hard (o)
	lower	EXPTIME-hard (p)		

Table 1: Overview of complexity results

4.5 Summary of Results

Table 1 provides a summary of the known results for the equivalence-checking problems considered in [3]. The "big" rows in the table contain results for different types of systems — FS, APC (acyclic PC), APCH (acyclic PCH), PC (parallel compositions), and PCH (parallel compositions with hiding). For each such type of systems, the known upper and lower complexity bounds are presented. The columns correspond to specific relations: bisimilarity $\sim$, simulation preorder $\sqsubseteq_{sim}$ and trace preorder $\sqsubseteq_{tr}$. The cells that span the columns for $\sim$ and $\sqsubseteq_{sim}$ contain hardness results holding for any $\mathcal{R}$ such that $\sim\subseteq \mathcal{R} \subseteq\sqsubseteq_{sim}$. The cells that span the columns from $\sim$ to $\sqsubseteq_{tr}$ contain hardness results holding for any $\mathcal{R}$ such that $\sim\subseteq \mathcal{R} \subseteq\sqsubseteq_{tr}$.
All hardness results in the table hold even for centralized systems.

Symbols (a)–(p) in the table refer to the following explanations.

a) Polynomiality easily follows by a greatest fixpoint construction; for more efficient algorithms see e.g. [54, 68].

b) It is a special case of language inclusion for nondeterministic finite automata (NFA), which is reducible to language equivalence — a well known PSPACE-complete problem (see, e.g., [65]).

c) This is easily derivable from the PSPACE-hardness of the above problem for NFA.

d) Proved in [5]; Theorem 4.4 (proved in [3]) provides an alternative proof.

e) Proposition 4.12.

f) Proposition 4.9.

g) Theorem 4.11.

h) Theorem 4.8.

i) Proposition 4.10.

j) The global transition system (of exponential size) can be constructed explicitly, for which a polynomial time algorithm from (a) can be used.

k) We can use the idea from [78], mentioned in Proposition 4.9. (It is sufficient to generate and verify a distinguishing trace of at most exponential length).

l) Proved in [58] (a reduction from ALBA-ACCEPT using a variant of the Defender Choice technique).

m) Proved in [70] (by a "master reduction").

n) The explicitly constructed global transition system is an NFA of exponential size, to which we can apply a polynomial space algorithm (see (b)).

o) Proved in [70] by a reduction from RE^2 (equivalence of regular expressions with squaring).

p) Theorem 4.7.

5 Bisimilarity between Normed BPA and Normed BPP

This section describes the article [2]:

- Petr Jančar, Martin Kot, Zdeněk Sawa: *Complexity of deciding bisimilarity between normed BPA and normed BPP*, *Information and Computation*, Special Issue: 19th International Conference on Concurrency Theory (CONCUR 2008), Volume 208, Issue 10, pp. 1193–1205, Elsevier, 2010.

A preliminary version of this article appeared (with no complexity analysis) at Concur 2008 [7].

5.1 Motivation and State of the Art

There are three natural "context-free" subclasses of process rewrite systems: *BPA* (Basic Process Algebra) that allows only sequential composition, *BPP* (Basic Parallel Processes) that allows only parallel composition, and *PA* that allows both.

When we consider the question of deciding bisimilarity of these types of systems, the known results are as follows. The best known algorithm for deciding bisimilarity on BPA was proposed by Burkart, Caucal and Steffen [19]. They claim their algorithm can be implemented to have double-exponential time complexity but this upper bound is not proved in the paper. The known lower bound is PSPACE-hardness of the problem [75]. For *normed* BPA, the polynomial-time algorithm was shown in [42] (with an upper bound $O(n^{13})$); the upper bound was later improved in [61], where an algorithm with running time in $O(n^8 \mathit{polylog}\ n)$ was described, and in [27], where the running time was improved to $O(n^5)$.

Deciding of bisimilarity on BPP is known to be PSPACE-complete. The PSPACE-hardness of the problem was shown by Srba [74], and the presence in PSPACE by Jančar [48]. For *normed* BPP, a polynomial time algorithm was presented in [41] (without a precise complexity analysis), based on so called *prime decompositions*; the upper bound $O(n^3)$ was shown in [49] by another algorithm, based on so called *dd-functions* (distance-to-disabling functions).

An algorithm for deciding bisimilarity on *normed* PA was shown by Hirshfeld and Jerrum [40]. The algorithm runs in doubly-exponential nondeterministic time. Decidability of bisimilarity on PA in the general (unnormed) case is a long-standing open problem.

The most difficult part of the above mentioned algorithm for normed PA [40] deals with the case when (a process expressed as) sequential composition is bisimilar to (a process expressed as) parallel composition. A basic subproblem is to analyze when a BPA process is bisimilar to a BPP process. Černá, Křetínský, Kučera [22] have shown that this subproblem is decidable in the *normed* case; their suggested algorithm is exponential. Decidability in the general (unnormed) case was shown in [50], without providing any complexity bound.

5.2 Main Results

The main result of [2] is a *polynomial* algorithm deciding whether a given normed BPA process α is bisimilar to a given normed BPP process M. The running time of the algorithm is $O(n^7)$.

An important ingredient of the algorithm is a new procedure, based on *dd*-functions, which transforms the normed BPP process M into "prime form" where bisimilarity coincides with equality; time complexity of this transformation is $O(n^3)$. Such transformation could be based on prime decompositions from [41] but with worse complexity (which was, in fact, not analyzed in [41]).

As a side result, our approach also shows a clear polynomial time algorithm, with running time $O(n^3)$, testing if there exists a bisimilar BPA process to a given BPP process; polynomiality was shown in [22], with no bound on the polynomial degree. Another side result is an algorithm for deciding bisimilarity between a given BPA process and a given finite-state process, with running time $O(n^4)$. Polynomiality of this problem was already shown by Kučera and Mayr [56]. In fact, they provided an $O(n^{12})$ algorithm for the more general case of *weak* bisimilarity; the complexity for the special case of (strong) bisimilarity was not analyzed in [56].

5.3 More Detailed Description of the Results

The central problem solved in [2] is problem nBPA-nBPP-BISIM, defined as follows:

Instance: A normed BPA-process (α_0, Σ), a normed BPP-process (M_0, Δ).

Question: Is $\alpha_0 \sim M_0$ (in the disjoint union of $\mathcal{S}_\Sigma$ and $\mathcal{S}_\Delta$) ?

(Here, $\mathcal{S}_\Sigma$ and $\mathcal{S}_\Delta$ denote the transition systems generated by Σ and Δ.)

As the size n of an instance of nBPA-nBPP-BISIM we understand the number of bits needed for its (natural) presentation; in particular we consider the numbers $F(t,p)$ in Δ and the numbers in M_0 to be written in binary.

In the rest of this section we assume a fixed nBPA Σ and a fixed nBPP Δ. By a *state* we generally mean a state in the disjoint union of $\mathcal{S}_\Sigma$ and $\mathcal{S}_\Delta$.

The algorithm for solving nBPA-nBPP-BISIM starts by a transformation of nBPP Δ into a *prime form*. We say that a *BPP net* Δ is *in prime form* if bisimilarity coincides with identity on the generated LTS, i.e., $M \sim M'$ iff

$M = M'$. (In this case, each place p is a "prime" since it is not equivalent to a composition of other places.)

It follows from the unique decomposition results in [41] that for each normed BPP system Δ there is an equivalent normed BPP system Δ' in prime form, and that Δ' can be constructed from Δ in polynomial time using the algorithm, described in [41], which computes certain prime decompositions of BPP-variables (i.e., BPP-net places); it is a polynomial time algorithm but its precise complexity has not been analyzed.

An alternative algorithm, based on the use of so called *dd*-functions, was presented in [2]. It transforms a given normed BPP system $\Delta = (P, \mathit{Tr}, \textsc{pre}, F, \mathit{Act}, l)$ into a normed BPP system $\Delta' = (P', \mathit{Tr}', \textsc{pre}', F', \mathit{Act}, l')$ in prime form, and any given state (marking) M of Δ into M' of Δ' such that $M \sim M'$. The running time of the algorithm is $O(n^3)$. Moreover, $|\mathit{Tr}'| \leq |\mathit{Tr}|, |P'| \leq |P|$, and Δ' is represented in space $O(n^3)$.

Given a BPP process (M_0, Δ) where Δ is a normed BPP system in the prime form, it can be easily checked if there exists a normed BPA process (α_0, Σ) such that $\alpha_0 \sim M_0$, by testing simple conditions described below. Before stating these conditions, we need some additional definitions.

Let α be a state (of $\mathcal{S}_\Sigma$ or $\mathcal{S}_\Delta$). The *norm* of α, denoted $\|\alpha\|$, is the length of the shortest $w \in \mathit{Act}^*$ such that $\alpha \xrightarrow{w} \varepsilon$. Note that this also defines norm $\|X\|$ for each variable (place) X.

A place $p \in P$ is *unbounded* in (M_0, Δ) iff for each $c \in \mathbb{N}$ there is a marking M' such that $M_0 \longrightarrow^* M'$ and $M'(p) > c$.

We define $\mathit{Car}(M) = \{p \in P \mid M(p) \geq 1\}$.

A place p is called a *single final place*, an SF-place, if all transitions that take a token from p are of the form $p \xrightarrow{a} p^k$, $k \geq 0$ (they can only put tokens back to p). It is easy to see that $\|p\| = 1$ for every SF-place p (since Δ is normed). We say that p is a non-SF-place if it is not an SF-place.

We say that an SF-place p is *growing* if there is a transition $p \xrightarrow{a} p^k$ for $k \geq 2$.

Lemma 5.1 *For (M_0, Δ), Δ being a normed BPP in prime form, there exists a normed BPA process (α_0, Σ) such that $\alpha_0 \sim M_0$, iff the following conditions hold:*

1. *each non-SF-place is bounded,*
2. *there is no M such that $M_0 \longrightarrow^* M$, $|\mathit{Car}(M)| \geq 2$ and $M(p) \geq 1$ for some growing SF-place p,*
3. *each non-growing SF-place p is bounded.*

We note that the conditions in Lemma 5.1 can be checked by straightforward standard algorithms, linear in the size of Δ in prime form (which means $O(n^3)$ if Δ is not in prime form). We thus have the following corollary.

Corollary 5.2 *The problem to decide if a given normed BPP process (not necessarily in prime form) is bisimilar to some (unspecified) normed BPA process can be solved in time $O(n^3)$.*

If (M_0, Δ) satisfies the conditions of Lemma 5.1, the corresponding BPA process (α_0, Σ) can be constructed but its size can be exponential with respect to the size of (M_0, Δ).

A basic idea of an algorithm for nBPA-nBPP-BISIM is to construct an nBPA process bisimilar to a given nBPP process (if it exists) and then to use some (polynomial time) algorithm for deciding if this constructed nBPA process is bisimilar to the nBPA process from the instance of nBPA-nBPP-BISIM. The complexity of such algorithm would be exponential in general, but the following theorem allows to obtain a polynomial time algorithm.

Theorem 5.3 *Assume a normed BPA system Σ, with the set V of variables, and a normed BPP system Δ in prime form, with the set P of places. The number of markings M of Δ such that $\alpha \sim M$ for some $\alpha \in V^+$ and M does not have all tokens in one* SF*-place is at most $4y^2$, where $y = \max\{|V|, |P|\}$.*

Theorem 5.3 is proved by partitioning the markings into four following classes and showing that there are at most y^2 markings in each class:

Class 1. Markings M with all tokens in one (non-SF) place ($|Car(M)| = 1$).

Class 2. Markings M with $|Car(M)| \geq 2$ where at least two different places with norm 1 are reachable; this necessarily means $M \longrightarrow^* M'$ for some M' satisfying $M'(p_1) \geq 1$, $M'(p_2) \geq 1$ for some $p_1 \neq p_2$ and $\|p_1\| = \|p_2\| = 1$.

Class 3. Markings M with $|Car(M)| \geq 2$ and with exactly one reachable ("sink") place p with norm 1, where p is a non-SF-place.

Class 4. Markings M with $|Car(M)| \geq 2$ and with exactly one reachable ("sink") place p with norm 1, where p is an SF-place.

Assume an instance of nBPA-nBPP-BISIM, i.e., nBPA process (α_0, Σ) and nBPP process (M_0, Δ). The polynomial algorithm for nBPA-nBPP-BISIM works as follows.

It first transforms (M_0, Δ) to bisimilar (M_0', Δ') where Δ' is in prime form. The algorithm then starts to build nBPA Σ' for (M_0', Δ'). The variables of Σ' represent either tokens in SF-places or markings of Δ' where not all tokens are in one SF-place. Whenever the number of variables of the latter type exceeds $4y^2$, where y is the maximum of $\{|V_\Sigma|, |P_{\Delta'}|\}$, then the algorithm stops immediately with the answer $\alpha_0 \not\sim M_0$; this is correct due to Theorem 5.3.

It is not necessary for the algorithm to test the conditions of Lemma 5.1 explicitly because it is ensured that in this case it will stop with the correct answer due to exceeding the number of variables.

If the number of variables does not exceed $4y^2$, the algorithm finishes the construction of Σ'. In fact, the algorithm does not construct Σ' explicitly but rather a succinct representation of it, since a straightforwardly constructed nBPA could be of exponential size. However, this succinct representation can be easily transformed into a "usual" nBPA of polynomial size by adding some additional variables and rules.

Since the size of the constructed nBPA process (α'_0, Σ') is polynomial with respect to the size of the original instance of the problem, we can use some known polynomial time algorithm for deciding bisimilarity on nBPA, such the algorithms from [42] or [27], to decide if $(\alpha_0, \Sigma) \sim (\alpha'_0, \Sigma')$, and by this obtain a polynomial time algorithm for nBPA-nBPP-BISIM.

The constructed nBPA process (α'_0, Σ') is in a very special form — it is a finite state system (FS) extended with "SF-tails". This allows us to avoid the use of general algorithms for deciding bisimilarity on nBPA and to develop a more efficient specialized algorithm.

The specialized algorithm has time complexity $O(n^7)$ and is inspired by several ideas used, e.g., in the proofs in [56, 57, 61]. In particular, it uses the idea from [61] of a reduction to the problem of finding a (unique) maximal solution of a certain set of boolean equations, which was used there in the algorithm for deciding bisimilarity on normed BPA. The idea considerably simplifies the complexity analysis and gives a better complexity bound than would be obtained by a straightforward analysis of the algorithm based on the computation of the fixpoint.

Putting everything together, we obtain the main result:

Theorem 5.4 *There is an algorithm solving* nBPA-nBPP-BISIM *in time* $O(n^7)$.

The described algorithm can be used for deciding bisimilarity between a given nBPA (of size m) and a finite state system (with k states and ℓ transitions) and the running time of the algorithm is $O(mk^3 + m\ell k + \ell^2) = O(n^4)$ in this case (where n is the size of the whole instance). In fact, the algorithm can be easily adapted for the case when the BPA and the FS in the instance are not required to be normed (as in [56, 57]) without affecting its complexity. The more general problem of deciding weak bisimilarity on a given BPA and FS process was considered in [56] and the algorithm presented there has running time $O(m^5(k+\ell)^7) = O(n^{12})$. (The special case of the strong bisimilarity was not analyzed there and we are not aware of any tighter results concerning its complexity.)

6 Non-Interleaving Equivalences on BPP

This section gives an overview of the article [1]:

- Sibylle Fröschle, Petr Jančar, Sławomir Lasota,Zdeněk Sawa: *Non-Interleaving Bisimulation Equivalences on Basic Parallel Processes*, *Information and Computation*, Volume 208, Issue 1, pp. 42–62, Elsevier, 2010.

6.1 Motivation and State of the Art

The classical bisimilarity models components running in parallel by nondeterministic interleaving of actions of these components, i.e., it does not distinguish between actions that can be run in parallel and actions that can be run only sequentially in an arbitrary order. There were proposed several variants of bisimilarity that take the distinction between these two cases into account and model concurrency in a more faithful way. Such equivalences are called *non-interleaving equivalences.*

Most non-interleaving bisimulation equivalences coincide on BPP, and they are equal to *history preserving bisimilarity* (*hp-b*) [80]. In [12] Aceto shows that distributed bisimilarity [20] and causal bisimilarity [28] coincide for a language that is essentially BPP without recursion. In an unpublished draft [55] Kiehn has extended these results by proving that location equivalence [21], causal bisimilarity, and distributed bisimilarity coincide over CPP, an extension of BPP that allows for synchronization in CCS style but disallows explicit τ actions. Causal bisimilarity is known to coincide with hp-b in general [11]. A direct proof of the coincidence between hp-b and distributed bisimilarity on BPP is provided in [30]. Finally, it has been shown in [60] that for BPP distributed bisimilarity coincides with performance equivalence [38]. To sum up, on BPP all relevant non-interleaving bisimulation equivalences coincide with history preserving bisimilarity, with one exception, which is the finer *hereditary history preserving bisimilarity* (*hhp-b*). Hhp-b takes a special position among non-interleaving equivalences: it is often considered to be *the* bisimulation equivalence for true-concurrency [52, 34]. Unlike all the other equivalences it is undecidable for finite-state systems, in particular, for 1-safe Petri nets [53]. Only a few positive results could be achieved for restricted classes [33].

In [30, 31], Fröschle has shown a tableau-based decision procedure for deciding hhp-b on BPP. Later, Fröschle and Lasota [35] have given a fixpoint characterization of hhp-b on BPP. The exact computational complexity of the procedure was not analyzed in these papers.

For hp-b, a polynomial-time algorithm follows immediately from the general scheme when we use the algorithm from [41] for deciding classical bisimilarity on normed BPP as a subroutine. A technically more complicated version of this approach was used for deciding distributed bisimilarity (and thus hp-b) on BPP by Lasota in [59]. (A generalized version of the algorithm from [41] was also used in [36] to show a polynomial-time algorithm deciding distributed bisimilarity on BPP_τ, an extension of BPP with synchronization on complementary actions in CCS style.) The degree of the polynomial has not been analyzed but it seems relatively large even when the (apparently more efficient) algorithm [49] is used.

6.2 Main Results

The main results of [1] are polynomial-time algorithms deciding hhp-b and hp-b on BPP. The running time of the former algorithm is $O(n^3 \log n))$ and of the latter $O(n^6)$.

The algorithms build on the ideas presented in [8] and [35] and partly in [32] but the presentation is substantially revised, unified, and given in a new self-contained framework. In particular, a common base of polynomial time algorithms for hhp-b and hp-b was clarified: speaking informally in game terminology, the hhp-b game as well as the hp-b game may be split into a number of "local" games played over BPP processes of causal depth 1. This insight forms a core ingredient of both algorithms, providing a fixpoint characterization of hhp-b and hp-b on *tree-like labelled event structures*.

Both polynomial time algorithms avoid a (time-consuming) transformation of BPP systems into the Execution Normal Form from [35].

Note also that the existence of polynomial time algorithms for deciding hp-b and hhp-b on BPP is in contrast with PSPACE-completeness of deciding classical bisimilarity on BPP [74, 48].

The paper [1] also gives a short proof of the following result from [32]: hhp-b and hp-b coincide for *Simple BPP* (*SBPP*) [29]. SBPP correspond to BPP in normal form, which represent the entire BPP class when interleaving equivalences are considered; when non-interleaving equivalences are considered, they form a strictly smaller class. Since hhp-b and hp-b do *not* coincide for BPP in general, the coincidence for SBPP underlines that SBPP and BPP do behave differently with respect to non-interleaving equivalences.

6.3 Definitions of Hp-Bisimilarity and Hhp-Bisimilarity on BPP

In this subsection, definitions of hp-bisimilarity and hhp-bisimilarity on BPP are described. The definitions are based on definitions of these relations on *event structures*. These definitions are carried to BPP processes where corresponding event structures are obtained from syntax-tree unfoldings of expressions representing BPP processes. (Another equivalent option would be to provide semantics of BPP processes in terms of net unfoldings as, e.g., in [31].)

6.3.1 Hp-Bisimilarity and Hhp-Bisimilarity on Labelled Event Structures

We recall the notions of *history preserving bisimilarity* (*hp-bisimilarity*) and *hereditary history preserving bisimilarity* (*hhp-bisimilarity*) on labelled event structures, presenting them by means of bisimulation games. It is a variation of definitions given in [16], [80], [53], and elsewhere.

An *event structure* is a tuple $(\mathcal{E}, \lhd, \#)$ where $\mathcal{E}$ is a set of *events*, $\lhd$ is a partial order on $\mathcal{E}$ called the *causal order*, and $\# \subseteq \mathcal{E} \times \mathcal{E}$ is an irreflexive and symmetric relation called the *conflict relation*. We require that $\{e' \mid e' \lhd e\}$ is finite (the number of causes is finite for each $e \in \mathcal{E}$), and that $e\#e'$ and $e' \lhd e''$ implies $e\#e''$. Events e, e' are *concurrent* iff none of $e \lhd e'$, $e' \lhd e$, $e\#e'$ holds. A *labelled event structure*, a *LES* in short, is a tuple $\mathcal{S} = (\mathcal{E}, \lhd, \#, Act, lab)$ where $(\mathcal{E}, \lhd, \#)$ is an event structure, Act is a set of *actions*, and $lab : \mathcal{E} \to Act$ is a *labelling* function.

By a *configuration* (i.e., a "computation state") of an LES $\mathcal{S} = (\mathcal{E}, \lhd, \#, Act, lab)$ we mean a *finite* set $C \subseteq \mathcal{E}$ which is conflict-free, i.e., $\forall e, e' \in C : \neg(e\#e')$, and downwards closed wrt causality, i.e., $\forall e, e' : (e \in C \wedge e' \lhd e) \Rightarrow e' \in C$. We implicitly view a configuration as a labelled partial order, i.e., a structure $(C, \lhd, lab)$ where $\lhd$ and lab are inherited from $\mathcal{S}$. We refer to these structures when saying that two configurations C_1, C_2 of possibly different LESs with the same action set Act are isomorphic. (An isomorphism $f : C_1 \to C_2$ is thus a bijection which respects the causal order and the labelling.)
There is a natural transition relation between configurations: an event e is *enabled* at C if $e \notin C$ and $C' = C \cup \{e\}$ is a configuration; we then write $C \xrightarrow{e} C'$.

We now define the *hp-game* and the *hhp-game* simultaneously.
The *(h)hp-game* between Attacker and Defender on two LESs $\mathcal{S}_1$, $\mathcal{S}_2$ with the same action set Act is played as follows. Positions are triples (C_1, f, C_2) where C_1 is a configuration of $\mathcal{S}_1$, C_2 is a configuration of $\mathcal{S}_2$, and f is an isomorphism between C_1 and C_2. The *initial position* is $(\emptyset, \emptyset, \emptyset)$. From the current position (C_1, f, C_2), a play proceeds by the following rules.

1. Attacker chooses $i \in \{1, 2\}$ and an event e_i enabled at C_i. Defender has to respond by choosing an event e_{3-i} which is enabled at C_{3-i} and for which $f' = f \cup \{(e_1, e_2)\}$ is an isomorphism between $C'_1 = C_1 \cup \{e_1\}$ and $C'_2 = C_2 \cup \{e_2\}$ (which also entails $lab(e_1) = lab(e_2)$). The play continues from the new position (C'_1, f', C'_2).

2. In the *hhp-game* (but not in the hp-game), Attacker may alternatively perform a *backtracking move*: he chooses $e \in C_1$ such that e is maximal in C_1 (wrt the respective causal order $\lhd$), and removes e and $f(e)$ (which is necessarily maximal in C_2) from C_1 and C_2, respectively. The new position is thus $(C_1 - \{e\}, f - \{(e, f(e))\}, C_2 - \{f(e)\})$.

3. The play continues like this either forever, in which case Defender wins, or until either Attacker or Defender is unable to move, in which case the other player wins.

Two LESs $\mathcal{S}_1$ and $\mathcal{S}_2$ are *hp-bisimilar* (*hhp-bisimilar*) iff Defender has a winning strategy in the hp- (hhp-) game on $\mathcal{S}_1$, $\mathcal{S}_2$; we write $\mathcal{S}_1 \sim_{hp} \mathcal{S}_2$ ($\mathcal{S}_1 \sim_{hhp} \mathcal{S}_2$). It is straightforward to show that if $\mathcal{S}_1 \not\sim_{hp} \mathcal{S}_2$ ($\mathcal{S}_1 \not\sim_{hhp} \mathcal{S}_2$) then Attacker has a winning strategy.

Remark. It is more standard to define relations $\sim_{hp}$ and $\sim_{hhp}$ as the union of hp-bisimulations and hhp-bisimulations, respectively. However, the definitions by games are more appropriate for proofs in [1].

We can note that $\sim_{hhp}$ is finer than $\sim_{hp}$, i.e., $\mathcal{S}_1 \sim_{hhp} \mathcal{S}_2$ implies $\mathcal{S}_1 \sim_{hp} \mathcal{S}_2$, and, in fact, it can be shown to be *strictly* finer.

Convention. Many later notions and results are analogous for $\sim_{hp}$ and $\sim_{hhp}$. We thus let h range over $\{hp, hhp\}$, and we write $\sim_h$ and the h-game when meaning that any of 'hp', 'hhp' can be substituted for 'h' in a given context.

6.3.2 Hp-Bisimilarity and Hhp-Bisimilarity on BPP

Each BPP expression E can be presented by its *syntax tree*, denoted by *stree*(E): it is a rooted tree whose nodes are labelled with elements of $\{\mathbf{0}, +, \|\} \cup Act \cup Var$. Each node labelled by $+$ or $\|$ has two children; each node labelled by an action has one child; and each node labelled by $\mathbf{0}$ or by a variable is a leaf.

Example 6.1 *Figure 8 shows stree*(E) *with nodes* $u_0, u_1, \ldots, u_7$ *for expression* $E = ((\mathtt{a}.\mathtt{X}_1) \parallel (\mathtt{b}.(\mathtt{X}_1 + (\mathtt{a}.\mathtt{X}_2))))$.

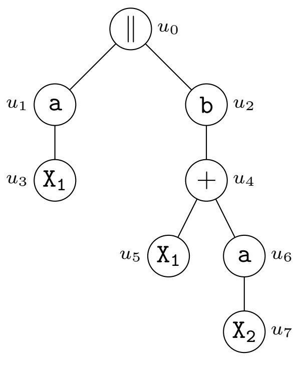

Figure 8: Syntax tree for expression $((\mathtt{a}.\mathtt{X}_1) \parallel (\mathtt{b}.(\mathtt{X}_1 + (\mathtt{a}.\mathtt{X}_2))))$

Given a BPP system $\Delta = \{X_i \stackrel{\text{def}}{=} E_i \mid 1 \leq i \leq k\}$, each BPP process (E, Δ) naturally corresponds to its *unfolded syntax tree*, denoted by $\mathit{unf}(E)$, which is defined as the limit of the following process:

1. Start by taking a copy of the syntax tree $\mathit{stree}(E)$ as the current tree CT.
2. Whenever there is a leaf u in CT labelled with variable X_i, replace the singleton subtree u with a copy of $\mathit{stree}(E_i)$. Take the result to be the new CT.

The trees $\mathit{unf}(E)$ naturally give rise to labelled event structures of special kind, from which they inherit (hereditary) history-preserving bisimilarity and other concepts. For convenience we treat a broader class of trees and the corresponding "tree-like event structures".

A *process tree* T is a (possibly infinite) rooted tree equipped with a labelling $\mathit{lab} : V \to \{\mathbf{0}, +, \|\} \cup \mathit{Act}$ where V is the set of nodes of T; we stipulate the following conditions hold:

- each node of T labeled with $\mathbf{0}$ is a leaf (it has no children);
- each node labeled with an action (element of Act) has at most one child.

A node u is called an *action node* iff $\mathit{lab}(u) \in \mathit{Act}$; we refer to the set of action nodes of T by $\mathit{actnodes}(T)$; a node v with $\mathit{lab}(v) = +$ is called a *choice node.*

Notation for trees: Given a rooted tree T, $\mathit{root}(T)$ denotes its root. We write $u \in T$ to say that u is a node of T. By $\mathit{tree}(u)$, where $u \in T$, we denote the (full) subtree of T rooted in the node u.

By $\lhd$ we denote the *tree-order* on the nodes: $v \lhd v'$ iff v lies on the path from $\mathit{root}(T)$ to v'; we assume $v \lhd v$. If $v \lhd v'$, $v \neq v'$, then v is a *predecessor* of v' and v' is a *successor* of v. We note that for any two nodes u_1, u_2 such that $u_1 \not\lhd u_2$, $u_2 \not\lhd u_1$ there is a unique node v such that $u_1 \in \mathit{tree}(v_1)$, $u_2 \in \mathit{tree}(v_2)$ for two different children v_1, v_2 of v; such v is called the *closest common predecessor* of u_1, u_2.

For a process tree T, labelled by actions from Act, the *labelled event structure associated to* T is the tuple

$$\mathtt{LES}(T) = (\mathit{actnodes}(T), \lhd, \#, \mathit{Act}, \mathit{lab})$$

where the events are the action nodes of T, the causal order $\lhd$ and the labelling lab are induced by the tree-order and the labelling in T, respectively, and the conflict relation $\#$ on $\mathit{actnodes}(T)$ is defined as follows:

> $u_1 \# u_2$ iff $u_1 \not\lhd u_2$, $u_2 \not\lhd u_1$, and the closest common predecessor of u_1, u_2 is a choice node (with label $+$).

The LESs associated with process trees are called the *tree-like labelled event structures.*

Remark. The axioms of event structures are easily seen to be satisfied. We also note that if two action nodes u_1, u_2 are concurrent (they are causally unrelated and non-conflicting) then their closest common predecessor is labelled with $\|$.

(Hereditary) history-preserving bisimilarity is naturally carried over to process trees and BPP processes:

$$\begin{array}{lll} T_1 \sim_h T_2 & \text{iff} & \mathtt{LES}(T_1) \sim_h \mathtt{LES}(T_2), \\ E_1 \sim_h E_2 & \text{iff} & \mathtt{LES}(\mathit{unf}(E_1)) \sim_h \mathtt{LES}(\mathit{unf}(E_2)). \end{array}$$

There is an example in [16] (see also [1]) of two variable-free BPP processes E, F, where each occurrence of action is followed by '$.\mathbf{0}$', such that $E \sim_{hp} F$ but $E \not\sim_{hhp} F$, which demonstrates that hhp-bisimilarity is *strictly* finer than hp-bisimilarity even on a very restricted class of BPP processes.

6.4 Formulation of Problems

The main aim is to obtain efficient polynomial-time algorithms for the problems of deciding hp- and hhp-bisimilarity on BPP processes, i.e., for the problems specified as follows (where $\sim_h$ stands for $\sim_{hhp}$ or $\sim_{hp}$):

INSTANCE: BPP processes (E, Δ_1) and (F, Δ_2).

QUESTION: Is $(E, \Delta_1) \sim_h (F, \Delta_2)$?

It is useful to note the following trivial reduction: instead of BPP processes (E, Δ_1) and (F, Δ_2) we can take a BPP system Δ given by the disjoint union of Δ_1 and Δ_2, extended with two fresh variables X, Y and with definitions $X \stackrel{\text{def}}{=} a.E$ and $Y \stackrel{\text{def}}{=} a.F$ for some action a, and then ask if $X \sim_h Y$. In fact, the algorithms provide finer answers — they partition all subexpressions in the BPP definition Δ wrt $\sim_h$.

So let $\Delta = \{X_i \stackrel{\text{def}}{=} E_i \mid 1 \leq i \leq k\}$ be a BPP definition; we use Act for $\mathit{Act}(\Delta)$.

We assume that the defining expressions $E_1, E_2, \ldots, E_k$ are available as a forest, denoted by $\mathit{forest}(\Delta)$, of k disjoint syntax trees $\mathit{stree}(E_1), \mathit{stree}(E_2), \ldots, \mathit{stree}(E_k)$ The nodes of (the trees in) $\mathit{forest}(\Delta)$ which are labelled by non-variable symbols are called the *nodes of BPP definition* Δ:

$$\mathtt{Nodes}(\Delta) = \{\alpha \mid \alpha \text{ is a node of } \mathit{forest}(\Delta) \text{ with } \mathit{lab}(\alpha) \in \mathit{Act} \cup \{+, \|, \mathbf{0}\}\}\,.$$

Each $\alpha \in \mathtt{Nodes}(\Delta)$ naturally represents a subexpression of some defining expression E_i in Δ (which is not a single variable); we denote this subexpression by E_α. Every E_α can be viewed as a BPP process, and we can thus carry over the notions for BPP processes to $\texttt{Nodes}(\Delta)$. In particular, we write $\alpha \sim_h \beta$ whenever $E_\alpha \sim_h E_\beta$.

The problems BPP-HHP-BISIM (where $h = hhp$) and BPP-HP-BISIM (where $h = hp$) are defined as follows:

INPUT: A BPP system Δ.

OUTPUT: The partition of $\texttt{Nodes}(\Delta)$ into equivalence classes of $\sim_h$, denoted by $\mathcal{P}_h(\Delta)$.

Note that $X_i \sim_h \alpha$ where $\alpha = \mathit{root}(\mathit{stree}(E_i))$. Thus the original problems are indeed subsumed by BPP-HP-BISIM and BPP-HHP-BISIM though we have not included variable occurrences in $\texttt{Nodes}(\Delta)$.

6.5 Efficient Algorithms

The main results of [1] are polynomial time algorithms for problems BPP-HHP-BISIM and BPP-HP-BISIM:

Theorem 6.2 *There is an algorithm solving* BPP-HHP-BISIM *(i.e., computing $\mathcal{P}_{hhp}(\Delta)$ for a given BPP system Δ) in time $O(n^3 \log n)$.*

Theorem 6.3 *There is an algorithm solving* BPP-HP-BISIM *(i.e., computing $\mathcal{P}_{hp}(\Delta)$ for a given BPP system Δ) in time $O(n^6)$.*

A size of instance n in these theorems is defined as follows. For a BPP expression E we let $\mathit{size}(E)$ be the number of occurrences of symbols (including parentheses). The size of a definition $X \stackrel{\text{def}}{=} E$ is taken to be $\mathit{size}(E) + 2$, and the size of a BPP system Δ, denoted by $\mathit{size}(\Delta)$, is the sum of the sizes of the definitions in Δ.

Remark. It might be more accurate to view the size of Δ as the number of bits needed for a natural description of Δ but in the complexity analysis in [1] we use the unit cost complexity model [13], i.e., it assumes that operations like adding two numbers with $O(\log n)$ bits (where $n = \mathit{size}(\Delta)$) take constant time, so the difference does not matter.

The resulting partition $\mathcal{P}_h(\Delta)$ can be described as a fixed-point of a certain function that partitions elements of $\texttt{Nodes}(\Delta)$ according to *depth-1 h-games*, i.e., games where Attacker is restricted to use only *depth-1 event*, i.e., events that were enabled already at the start of the game.

The algorithms then can find the required fixed-point by repeatedly solving $\sim_h$ on *depth-1 trees*, which are defined as follows. The *depth-1 action nodes* of a process tree T are the nodes corresponding to depth-1 events in $\texttt{LES}(T)$. (Thus all predecessors of a depth-1 action node are labelled by $+$ or $\|$.) A process tree T is a *depth-1 tree* iff all action nodes of T are leaves (which also means that all action nodes of T are depth-1 action nodes). Note that there is no causal dependency between (different) events in LESs associated to depth-1 trees.

In principle, the only difference between algorithms for BPP-HHP-BISIM and BPP-HP-BISIM is the subroutine for computing $\sim_h$ on depth-1 trees.

In the case of BPP-HHP-BISIM, this subproblem is solved by transforming depth-1 trees to a special form, called the *trivial choice free form* (*TCF form*), and checking if the resulting trees are isomorphic. It is proved that for each depth-1 trees T, T' in the TCF form holds that $T \sim_{hhp} T'$ iff T and T' are isomorphic.

To obtain the running time $O(n^3 \log n)$, some other optimizations are used to avoid recomputations.

In the case of BPP-HP-BISIM, the subproblem can be solved by reducing it to deciding bisimilarity on normed BPP processes [41, 49], for which polynomial time algorithms are known.

The algorithm for BPP-HP-BISIM described in [1] uses (a variant of) the algorithm from [49] to decide $\sim_{hp}$ on depth-1 trees. The algorithm is based on the use of "distance-to-disabling" functions, which were introduced in [48]. In [1], the algorithm is specialized to the special simpler case where given BPP nets are *acyclic*.

Again, some optimizations of the algorithm are used to avoid recomputations and to obtain the running time $O(n^6)$.

The next subsection shows that for so called *simple BPP* the relations $\sim_{hhp}$ and $\sim_{hp}$ coincide, and so the more efficient algorithm for BPP-HHP-BISIM (with running time $O(n^3 \log n)$) can be used to solve BPP-HP-BISIM on these systems.

6.6 Simple BPP

Simple BPP processes (*SBPP*) [29] are BPP processes in a "Greibach" normal form, which is usually used when (interleaving) bisimilarity is considered. (They have been also introduced in [24], under the name BPP$_g$.) Following [29], we define *SBPP expressions* by the grammar:

$$P \ ::= \ X \ | \ S \ | \ P_1 \parallel P_2$$

where S stands for an *initially sequential expression* given by the following grammar:

$$S \ ::= \ \mathbf{0} \ | \ a.P \ | \ S_1 + S_2 \, .$$

Thus SBPP restricts the mixture of choice and parallel composition: general summation is replaced by *guarded summation.* In particular, this excludes processes such as $(P_1 \parallel P_2) + P_3$.

An *SBPP system* Δ is a BPP system $\{X_i \overset{\text{def}}{=} P_i \mid 1 \leq i \leq k\}$ where all P_i are SBPP expressions (over $Act(\Delta)$ and $Var(\Delta)$). An *SBPP process* is a pair

(P, Δ) where Δ is an SBPP system and P is an SBPP expression over $Act(\Delta)$ and $Var(\Delta)$.

It is proved in [1] that hp-bisimilarity coincides with hhp-bisimilarity on SBPP:

Theorem 6.4 *Two SBPP processes are hp-bisimilar iff they are hhp-bisimilar.*

This implies that when non-interleaving equivalences are considered, SBPP processes form a strictly smaller class than BPP processes, since on BPP, hhp-bisimilarity is a strictly finer relation than hp-bisimilarity.

7 Alternating finite automata

This section presents the article [4]:

- Petr Jančar, Zdeněk Sawa: *A note on emptiness for alternating finite automata with a one-letter alphabet*, *Information Processing Letters*, Volume 104, Issue 5, pp. 164–167, Elsevier, 2007.

7.1 Motivation and the State of the Art

One of natural problems of automata theory is checking *emptiness*, i.e., checking whether the language accepted by a given automaton is empty. It is well known that the emptiness problem is PSPACE-complete for alternating finite automata (AFA). The PSPACE-hardness of the problem follows from the PSPACE-completeness of the universality problem for nondeterministic finite automata, i.e., the problem of checking whether a given nondeterministic automaton accepts *all* words.

Holzer has shown in [44] that the emptiness problem for AFA remains PSPACE-hard even if we consider only alternating finite automata with a one-letter alphabet. Let us call this problem 1L-Afa-Emptiness.

The PSPACE-hardness of 1L-Afa-Emptiness can be used in proofs of PSPACE-hardness of some other problems. Sometimes reductions from 1L-Afa-Emptiness can be much simpler than reductions from some other well known PSPACE-complete problems such as quantified boolean formula or acceptance of a word by a linear bounded automaton.

In particular, 1L-Afa-Emptiness can be useful for proving some complexity lower bounds for model checking and equivalence checking problems. Examples of such problems are some problems dealing with one-counter automata. The complexity lower bounds of [72] can be strengthened by using a simple reduction from 1L-Afa-Emptiness as Markus Lohrey pointed out during the conference presentation of [72]. Also most of results in [6], which deal with one-counter automata and one-counter nets can be improved from DP-hardness to PSPACE-hardness by using 1L-Afa-Emptiness as the starting point of the reductions. Other example of use of PSPACE-hardness of 1L-Afa-Emptiness is [76], where Jiří Srba used it to prove PSPACE-hardness of some problems concerning visibly pushdown automata.

If one is interested in the actual proof of PSPACE-hardness of 1L-Afa-Emptiness, it is a bit unpleasant to find that Holzer uses the emptiness problem for so called EP0L systems [71], which was shown to be PSPACE-complete in [67], where the proof of PSPACE-hardness (solving a long-term open question) uses a series of reductions among several problems, one of these reductions being handled by a reference to [26].

7.2 Main Result

The main result of [4] is a simple direct proof of PSPACE-hardness of the problem 1L-AFA-EMPTINESS, i.e., the question whether a given alternating finite automaton with a one-letter alphabet accepts an empty language.

The proof proceeds by a "master reduction," i.e., it shows how to reduce *each* problem in PSPACE directly to 1L-AFA-EMPTINESS. In fact, the basic idea of the reduction was implicitly present already in the seminal paper on alternation [23]. A little adjustment of the construction could also serve to show the PSPACE-hardness of all problems in the above mentioned series in [67].

7.3 More Detailed Description of the Result

Let us start by recalling some basic definitions concerning alternating finite automata.

For a set X we use $Bool^+(X)$ to denote the set of (positive) boolean formulas that only use $\wedge$ and $\vee$ as boolean connectives and elements of X as variables. By $[\phi]_\nu$ we denote the truth value (0 or 1) of formula $\phi \in Bool^+(X)$ under the boolean assignment $\nu : X \to \{0,1\}$.

An *alternating finite automaton* (AFA) is a structure $A = (Q, \Sigma, \delta, q_0, F)$ where Q is the finite set of *states*, Σ is the finite *alphabet*, $\delta : Q \times \Sigma \to Bool^+(Q)$ is the *transition function*, q_0 is the *initial state*, and $F \subseteq Q$ is the set of *accepting states*.

We define the predicate $Acc \subseteq Q \times \Sigma^*$ by induction on the length of the second component; $Acc(q, w)$ is to be read as "A starting in q accepts w."

- $Acc(q, \varepsilon)$ iff $q \in F$.
- $Acc(q, aw)$ iff $[\delta(q, a)]_\nu = 1$ for the boolean assignment ν satisfying $(\,\nu(q') = 1 \Leftrightarrow Acc(q', w)\,)$ for all $q' \in Q$.

AFA A accepts the language $L(A) = \{w \in \Sigma^* \mid Acc(q_0, w)\}$.

When $|\Sigma| = 1$, we say that A is a 1L-AFA (1L being read "one letter").

We are interested in the problem 1L-AFA-EMPTINESS:

INSTANCE: 1L-AFA A.

QUESTION: Is $L(A) = \emptyset$?

The main result of [4] is:

Theorem 7.1 1L-AFA-EMPTINESS *is* PSPACE-*complete.*

The membership of the emptiness problem in PSPACE is straightforward, even in the case of general AFA; it was shown in [51].

The PSPACE-hardness of 1L-Afa-Emptiness is proved as follows. Assume an arbitrary problem P in PSPACE. There must exist a deterministic Turing machine M and a polynomial $p(n)$ such that M decides P and its space complexity is bounded by $p(n)$ (i.e., given an instance w of P as an input, M uses at most $p(n)$ cells on its tape during computation on w). To prove PSPACE-hardness of 1L-Afa-Emptiness, it is sufficient to describe an algorithm that for a given Turing machine M, polynomial $p(n)$, and a word w constructs 1L-AFA A_w such that

$$L(A_w) \neq \emptyset \text{ iff } M \text{ accepts } w \text{ using at most } p(|w|) \text{ cells on the tape.}$$

To be a logspace reduction from P to the (complement of) 1L-Afa-Emptiness, it is ensured that the space complexity of the algorithm is logarithmic wrt $|n|$. The algorithm can assume a fixed M and $p(n)$.

Remark. It is not important that the algorithm reduces P to the complement of 1L-Afa-Emptiness, because the class PSPACE is closed wrt complement and because the same algorithm reduces the complement of P to 1L-Afa-Emptiness (recall Observation 2.2).

References

Referenced journal papers of Zdeněk Sawa

[1] Sibylle Fröschle, Petr Jančar, Sławomir Lasota, and Zdeněk Sawa. Non-interleaving bisimulation equivalences on Basic Parallel Processes. *Information and Computation*, 208(1):42–62, 2010.

[2] Petr Jančar, Martin Kot, and Zdeněk Sawa. Complexity of deciding bisimilarity between normed BPA and normed BPP. *Information and Computation, Special Issue: 19th International Conference on Concurrency Theory (CONCUR 2008)*, 208(10):1193–1205, 2010.

[3] Zdeněk Sawa and Petr Jančar. Hardness of equivalence checking for composed finite-state systems. *Acta Informatica*, 46(3):169–191, 2009.

[4] Petr Jančar and Zdeněk Sawa. A note on emptiness for alternating finite automata with a one-letter alphabet. *Information Processing Letters*, 104(5):164–167, 2007.

[5] Zdeněk Sawa and Petr Jančar. Behavioural equivalences on finite-state systems are PTIME-hard. *Computing and Informatics*, 24(5):513–528, 2005.

[6] Petr Jančar, Antonín Kučera, Faron Moller, and Zdeněk Sawa. DP lower bounds for equivalence-checking and model-checking of one-counter automata. *Information and Computation*, 188(1):1–19, 2004.

Referenced conference papers of Zdeněk Sawa

[7] Petr Jančar, Martin Kot, and Zdeněk Sawa. Normed BPA vs. normed BPP revisited. In *Proceedings of the 19th International Conference on Concurency Theory (CONCUR 2008)*, volume 5201 of *Lecture Notes in Computer Science*, pages 434–446. Springer, 2008.

[8] Petr Jančar and Zdeněk Sawa. On distributed bisimilarity over Basic Parallel Processes. In *Proc. AVIS'05*, 2005.

[9] Zdeněk Sawa. Equivalence checking of non-flat systems is EXPTIME-hard. In *Proceedings of CONCUR 2003*, volume 2761 of *Lecture Notes in Computer Science*, pages 237–250. Springer-Verlag, 2003.

[10] Zdeněk Sawa and Petr Jančar. P-hardness of equivalence testing on finite-state processes. In *Proceedings of SOFSEM 2001*, volume 2234 of *Lecture Notes in Computer Science*, pages 326–335. Springer, 2001.

Other references

[11] Luca Aceto. History preserving, causal and mixed-ordering equivalence over stable event structures. *Fundamenta Informaticae*, 17(4):319–331, 1992.

[12] Luca Aceto. Relating distributed, temporal and causal observations of simple processes. *Fundamenta Informaticae*, 17(4):369–397, 1992.

[13] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[14] Jos C. M. Baeten and W. P. Weijland. Process algebra. *Cambridge Tracts in Theoretical Computer Science*, 18, 1990.

[15] José Balcázar, Joaquim Gabarró, and Miklós Sántha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4(6A):638–648, 1992.

[16] Marek A. Bednarczyk. Hereditary history preserving bisimulation or what is the power of the future perfect in program logics. Technical report, Polish Academy of Sciences, Gdańsk, 1991.

[17] Jan A. Bergstra and Jan Willem Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[18] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

[19] Olaf Burkart, Didier Caucal, and Bernhard Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, volume 969 of *Lecture Notes in Computer Science*, pages 423–433. Springer-Verlag, 1995.

[20] Ilaria Castellani. *Bisimulations for Concurrency*. PhD thesis, University of Edinburgh, 1988.

[21] Ilaria Castellani. Process algebras with localities. In Bergstra et al. [18], chapter 15, pages 945–1046.

[22] Ivana Černá, Mojmír Křetínský, and Antonín Kučera. Comparing expressibility of normed BPA and normed BPP processes. *Acta Informatica*, 36:233–256, 1999.

[23] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.

[24] Søren Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, University of Edinburgh, 1993.

[25] Rance Cleaveland and Oleg Sokolsky. Equivalence and preorder checking for finite-state systems. In Bergstra et al. [18], chapter 6, pages 391–424.

[26] Karel Culik, Josef Gruska, and Arto Salomaa. On a family of L languages resulting from systolic tree automata. *Theoretical Computer Science*, 23(3):231–242, 1983.

[27] Wojciech Czerwinski and Sławomir Lasota. Fast equivalence-checking for normed context-free processes. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 260–271. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

[28] Philippe Darondeau and Pierpaolo Degano. Causal trees. In *Proc. ICALP'89*, volume 372 of *Lecture Notes in Computer Science*, pages 234–248, 1989.

[29] Javier Esparza and Astrid Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *CAV'95*, volume 939 of *Lecture Notes in Computer Science*, pages 353–366. Springer-Verlag, 1995.

[30] Sibylle Fröschle. Decidability of plain and hereditary history-preserving bisimulation for BPP. In *Proc. EXPRESS'99*, volume 27 of *Electronic Notes in Theoretical Computer Science*, 1999.

[31] Sibylle Fröschle. *Decidability and Coincidence of Equivalences for Concurrency.* PhD thesis, University of Edinburgh, 2004.

[32] Sibylle Fröschle. Composition and decomposition in true-concurrency. In Vladimiro Sassone, editor, *Proc. FOSSACS'05*, volume 3441 of *Lecture Notes in Computer Science*, pages 333–347. Springer, 2005.

[33] Sibylle Fröschle. The decidability border of hereditary history preserving bisimilarity. *Information Processing Letters*, 93(6):289–293, 2005.

[34] Sibylle Fröschle and Thomas Hildebrandt. On plain and hereditary history-preserving bisimulation. In *MFCS'99*, volume 1672 of *Lecture Notes in Computer Science*, pages 354–365. Springer-Verlag, 1999.

[35] Sibylle Fröschle and Sławomir Lasota. Decomposition and complexity of hereditary history preserving bisimulation on BPP. In *Proc. CONCUR'05*, volume 3653 of *Lecture Notes in Computer Science*, pages 263–277. Springer-Verlag, 2005.

[36] Sibylle Fröschle and Sławomir Lasota. Normed processes, unique decomposition, and complexity of bisimulation equivalences. In *Proc. Infinity'06*, volume 239 of *Electronic Notes in Theoretical Computer Science*, pages 17–42. Elsevier, 2009.

[37] Alan Gibbons and Wojciech Rytter. *Efficient Parallel Algorithms.* Cambridge University Press, 1988.

[38] Roberto Gorrieri, Marco Roccetti, and Enrico Stancampiano. A theory of processes with durational actions. *Theoretical Computer Science*, 140(1):73–94, 1995.

[39] Jan Friso Groote and Faron Moller. Verification of parallel systems via decomposition. In *Proc. of CONCUR'92*, volume 630 of *Lecture Notes in Computer Science*, pages 62–76. Springer Verlag, 1992.

[40] Yoram Hirshfeld and Mark Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proceedings of 26th International Colloquium on Automata, Languages and Programming (ICALP'99)*, volume 1644 of *Lecture Notes in Computer Science*, pages 412–421. Springer-Verlag, 1999.

[41] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. *Mathematical Structures in Computer Science*, 6:251–259, 1996.

[42] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158:143–159, 1996.

[43] C. A. R. Hoare. *Communcating Sequential Processes.* Prentice-Hall, 1985.

[44] Markus Holzer. On emptiness and counting for alternating finite automata. In *Proceedings of Developments in Language Theory II*, pages 88–97. World Scientific, 1996.

[45] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

[46] Niel Immerman. *Descriptive Complexity*, pages 53–54. Springer-Verlag, 1998.

[47] Petr Jančar and Jiří Srba. Undecidability of bisimilarity by Defender's forcing. *Journal of the Association for Computing Machinery*, 55(1), February 2008.

[48] Petr Jančar. Strong bisimilarity on Basic Parallel Processes is PSPACE-complete. In *Proceedings of 18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 218–227. IEEE Computer Society, 2003.

[49] Petr Jančar and Martin Kot. Bisimilarity on normed Basic Parallel Processes can be decided in time $O(n^3)$. In *Proceedings of the Third International Workshop on Automated Verification of Infinite-State Systems (AVIS 2004)*, 2004.

[50] Petr Jančar, Antonín Kučera, and Faron Moller. Deciding bisimilarity between BPA and BPP processes. In *Proceedings of the 14th International Conference on Concurency Theory (CONCUR 2003)*, volume 2761 of *Lecture Notes in Computer Science*, pages 159–173. Springer-Verlag, 2003.

[51] Tao Jiang and Bala Ravikumar. A note on the space complexity of some decision problems for finite automata. *Information Processing Letters*, 40(1):25–31, October 1991.

[52] André Joyal, Morgens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

[53] Marcin Jurdziński, Mogens Nielsen, and Jiří Srba. Undecidability of domino games and hhp-bisimilarity. *Information and Computation*, 184:343–368, 2003.

[54] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, May 1990.

[55] Astrid Kiehn. A note on distributed bisimulations. Unpublished draft, 1999.

[56] Antonín Kučera and Richard Mayr. Weak bisimilarity between finite-state systems and BPA or normed BPP is decidable in polynomial time. *Theoretical Computer Science*, 270:667–700, 2002.

[57] Antonín Kučera and Richard Mayr. A generic framework for checking semantic equivalences between pushdown automata and finite-state automata. In *IFIP TCS*, pages 395–408. Kluwer, 2004.

[58] François Laroussinie and Philippe Schnoebelen. The state explosion problem from trace to bisimulation equivalence. In *Proc. 3rd Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'2000), Berlin, Germany, Mar.-Apr. 2000*, volume 1784 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2000.

[59] Sławomir Lasota. A polynomial-time algorithm for deciding true concurrency equivalences of Basic Parallel Processes. In *Proc. MFCS'03*, volume 2747 of *Lecture Notes in Computer Science*, pages 521–530. Springer-Verlag, 2003.

[60] Sławomir Lasota. Decidability of performance equivalence for basic parallel processes. *Theoretical Computer Science*, 360:172–192, 2006.

[61] Sławomir Lasota and Wojciech Rytter. Faster algorithm for bisimulation equivalence of normed context-free processes. In *Proceedings of 31st International Symposium on Mathematical Foundations of Computer Science (MFCS 2006)*, volume 4162 of *Lecture Notes in Computer Science*, pages 646–657. Springer-Verlag, 2006.

[62] Anthony Mansfield. On the computational complexity of a merge recognition problem. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 5(1):119–122, 1983.

[63] Richard Mayr. Combining Petrin nets and PA-processes. In *Proceedings of TACS'97*, volume 1281 of *Lecture Notes in Computer Science*. Springer, 1997.

[64] Richard Mayr. Process rewrite systems. *Information and Computation*, 156(1-2):264–286, 2000.

[65] Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE, 25–27 October 1972.

[66] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[67] Angelo Monti and Alessandro Roncato. Completeness results concerning systolic tree automata and E0L languages. *Information Processing Letters*, 53(1):11–16, 1995.

[68] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, December 1987.

[69] David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science: 5th GI-Conference, Karlsruhe*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, March 1981.

[70] Alexander Rabinovich. Complexity of equivalence problems for concurrent systems of finite agents. *Information and Computation*, 139(2):111–129, 15 December 1997.

[71] Grzegorz Rozenberg and Arto Salomaa. *The Mathematical Theory of L Systems*, volume 90 of *Pure and Applied Mathematics*. Academic Press, 1980.

[72] Olivier Serre. Parity games played on transition graphs of one-counter processes. In Luca Aceto and Anna Ingólfsdóttir, editors, *Proceedings of FOSSACS 2006*, volume 3921 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2006.

[73] Sandeep K. Shukla, Harry B. Hunt, Daniel J. Rosenkrantz, and Richard E. Stearns. On the complexity of relational problems for finite state processes. In *Proceedings of ICALP'96*, volume 1099 of *Lecture Notes in Computer Science*, pages 466–477. Springer-Verlag, 1996.

[74] Jiří Srba. Strong bisimilarity and regularity of Basic Parallel Processes is PSPACE-hard. In *Proceedings of 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2002)*, volume 2285 of *Lecture Notes in Computer Science*, pages 535–546. Springer, 2002.

[75] Jiří Srba. Strong bisimilarity and regularity of Basic Process Algebra is PSPACE-hard. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *Lecture Notes in Computer Science*, pages 716–727. Springer, 2002.

[76] Jiří Srba. Visibly pushdown automata: From language equivalence to simulation and bisimulation. In Zoltán Ésik, editor, *Proceedings of CSL 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 89–103. Springer-Verlag, 2006.

[77] Colin Stirling. Bisimulation, model checking and other games. Notes for Mathfit Workshop on Finite Model Theory, University of Wales, Swansea, July 1996.

[78] Antti Valmari and Antti Kervinen. Alphabet-based synchronisation is exponentially cheaper. In *Proceedings of CONCUR 2002*, volume 2421 of *Lecture Notes in Computer Science*, pages 161–176, 2002.

[79] Rob J. van Glabbeek. The linear time—branching time spectrum I. The semantics of concrete, sequential processes. In Bergstra et al. [18], chapter 1, pages 3–99.

[80] Rob J. van Glabbeek and Ursula Goltz. Equivalence notions for concurrent systems and refinement of actions. In *Proc. MFCS'89*, volume 379 of *Lecture Notes in Computer Science*, pages 237–248, 1989.

[81] Manfred K. Warmuth and David Haussler. On the complexity of iterated shuffle. *Journal of Computer and System Sciences*, 28(3):345–358, June 1984.

Appendix 1

Behavioural equivalences on finite-state systems are PTIME-hard

Authors: Zdeněk Sawa, Petr Jančar

Journal: *Computing and Informatics*, Volume 24, Issue 5, pages 513–528, Slovak Academy of Sciences, Institute of Informatics, 2005.

Grants: the Grant Agency of the Czech Republic, Grant No. 201/03/1161, and in final stages also the support by the Czech Ministry of Education, Project 1M6840770004

Appendix 2

Hardness of equivalence checking for composed finite-state systems

Authors: Zdeněk Sawa, Petr Jančar

Journal: *Acta Informatica*, Volume 46, Issue 3, pages 169–191, Springer, 2009.

Grant: the Czech Ministry of Education, Grant No. 1M0567

Appendix 3

Complexity of deciding bisimilarity between normed BPA and normed BPP

Authors: Petr Jančar, Martin Kot, Zdeněk Sawa

Journal: *Information and Computation*, Special Issue: 19th International Conference on Concurrency Theory (CONCUR 2008), Volume 208, Issue 10, pages 1193–1205, Elsevier, 2010.

Grant: the Czech Ministry of Education, Grant No. 1M0567

Appendix 4

Non-interleaving bisimulation equivalences on Basic Parallel Processes

Authors: Sibylle Fröschle, Petr Jančar, Sławomir Lasota, Zdeněk Sawa

Journal: *Information and Computation*, Volume 208, Issue 1, pages 42–62, Elsevier, 2010.

Grant: the Czech Ministry of Education, Grant No. 1M0567

Appendix 5

A note on emptiness for alternating finite automata with a one-letter alphabet

Authors: Petr Jančar, Zdeněk Sawa

Journal: *Information Processing Letters*, Volume 104, Issue 5, pages 164–167, Elsevier, 2007.

Grant: the Czech Ministry of Education, Grant No. 1M0567

List of Publications of Zdeněk Sawa

Remark. Listed items contain information about databases in which they are recorded (Web of Science, Scopus, DBLP). Items that do not contain this information are not recorded in any of these databases.

Journal articles also contain information about the Impact Factor (IF) of the given journal in the year when the article was published (or the newest impact factor if the impact factor for the given year was not determined yet), and about the rank of the journal in its category (or more categories if the journal is assigned to more than one category). The information about the impact factors and ranks comes from Journal Citation Reports (JCR) published by Thomson Reuters for the given year.

Journal Publications:

- Zdeněk Sawa: *Efficient Construction of Semilinear Representations of Languages Accepted by Unary Nondeterministic Finite Automata*, *Fundamenta Informaticae*, IOS Press, not published yet, accepted for publication in May 2011.
  IF 0.522 (year 2010), category: Computer Science, Software Engineering, rank 86 of 99, category: Mathematics, Applied, rank 172 of 236

- Petr Jančar, Martin Kot, Zdeněk Sawa: *Complexity of deciding bisimilarity between normed BPA and normed BPP*, *Information and Computation*, Special Issue: 19th International Conference on Concurrency Theory (CONCUR 2008), Volume 208, Issue 10, pp. 1193–1205, Elsevier, 2010.
  DOI: `10.1016/j.ic.2009.10.012`
  In: Web of Science, Scopus, DBLP
  IF 0.825 (year 2010), category: Computer Science, Theory & Methods, rank 61 of 97, category: Mathematics, Applied, rank 106 of 236

- Sibylle Fröschle, Petr Jančar, Sławomir Lasota,Zdeněk Sawa: *Non-Interleaving Bisimulation Equivalences on Basic Parallel Processes*, *Information and Computation*, Volume 208, Issue 1, pp. 42–62, Elsevier, 2010.
  DOI: `10.1016/j.ic.2009.06.001`
  In: Web of Science, Scopus, DBLP
  IF 0.825 (year 2010), category: Computer Science, Theory & Methods, rank 61 of 97, category: Mathematics, Applied, rank 106 of 236

- Zdeněk Sawa, Petr Jančar: *Hardness of equivalence checking for composed finite-state systems*, *Acta Informatica*, Volume 46, Issue 3, pp. 169–191, Springer, 2009.
  DOI: `10.1007/s00236-008-0088-x`

In: Web of Science, Scopus, DBLP
IF 0.923 (year 2009), category: Computer Science, Information Systems, rank 77 of 116

- Petr Jančar, Zdeněk Sawa: *A note on emptiness for alternating finite automata with a one-letter alphabet*, *Information Processing Letters*, Volume 104, Issue 5, pp. 164–167, Elsevier, 2007.
  DOI: `10.1016/j.ipl.2007.06.006`
  In: Web of Science, Scopus, DBLP
  IF 0.660 (year 2007), category: Computer Science, Information Systems, rank 59 of 92

- Zdeněk Sawa, Petr Jančar: *Behavioural Equivalences on Finite-State Systems are PTIME-hard*, *Computing and Informatics*, Volume 24, Issue 5, pp. 513–528, Slovak Academy of Sciences, Institute of Informatics, 2005.
  In: Web of Science, Scopus, DBLP
  IF 0.091 (year 2005), category: Computer Science, Artificial Intelligence, rank 78 of 79

- Petr Jančar, Antonín Kučera, Faron Moller, Zdeněk Sawa: *DP lower bounds for equivalence-checking and model-checking of one-counter automata*, *Information and Computation*, Volume 188, Issue 1, pp. 1–19, Elsevier, 2004.
  DOI: `10.1016/S0890-5401(03)00171-8`
  In: Web of Science, Scopus, DBLP
  IF 0.920 (year 2004), category: Computer Science, Theory & Methods, rank 28 of 70, category: Mathematics, Applied, rank 35 of 162

Conference Papers:

- Zdeněk Sawa: *Efficient Construction of Semilinear Representations of Languages Accepted by Unary NFA*, 4th International Workshop on Reachability Problems (RP 2010), LNCS 6227, pp. 176–182, Springer-Verlag, 2010.
  DOI: `10.1007/978-3-642-15349-5_12`
  In: Web of Science, Scopus, DBLP

- Petr Jančar, Martin Kot, Zdeněk Sawa: *Normed BPA vs. Normed BPP Revisited*, Proceedings of CONCUR 2008, LNCS 5201, pp. 434–446, Springer-Verlag, 2008. (best paper award)
  DOI: `10.1007/978-3-540-85361-9_34`
  In: Web of Science, Scopus, DBLP

- Zdeněk Sawa, Petr Jančar: *History Preserving Bisimilarity on Basic Parallel Processes*, Proceedings of TAAPSD'06, Kiev, pp. 9–14, 2006.

- Petr Jančar, Zdeněk Sawa: *On Distributed Bisimilarity over Basic Parallel Processes*, Proceedings of workshop AVIS 2005 (A Satellite Workshop of ETAPS 2005), 2005.

- Martin Kot, Zdeněk Sawa: *Bisimulation Equivalence of a BPP and a Finite State System can be Decided in Polynomial Time*, Proceedings of the 6th International Workshop on Verification of Infinite-State Systems (INFINITY 2004), *Electronic Notes in Theoretical Computer Science*, Volume 138, Issue 3, pp. 49–60, Elsevier, 2005.
  DOI: `10.1016/j.entcs.2005.02.065`
  In: Scopus, DBLP

- Zdeněk Sawa: *Equivalence Checking of Non-flat Systems Is EXPTIME-hard*, Proceedings of CONCUR 2003, LNCS 2761, pp. 237–250, Springer-Verlag, 2003.
  DOI: `10.1007/978-3-540-45187-7_16`
  In: Web of Science, Scopus, DBLP

- Petr Jančar, Antonín Kučera, Faron Moller, Zdeněk Sawa: *Equivalence-Checking with One-Counter Automata: A Generic Method for Proving Lower Bounds*, Proceedings of FOSSACS 2002, LNCS 2303, pp. 172–186, Springer-Verlag, 2002.
  DOI: `10.1007/3-540-45931-6_13`
  In: Web of Science, DBLP

- Zdeněk Sawa, Petr Jančar: *P-hardness of Equivalence Testing on Finite-State Processes*, Proceedings of SOFSEM 2001, LNCS 2234, pp. 326–335, Springer, 2001.
  DOI: `10.1007/3-540-45627-9_29`
  In: DBLP

- Petr Jančar, Faron Moller, Zdeněk Sawa: *Simulation Problems for One-Counter Machines*, Proceedings of SOFSEM'99, LNCS 1725, pp. 404–413, Springer-Verlag, 1999.
  DOI: `10.1007/3-540-47849-3_28`
  In: Web of Science, DBLP

Ph.D. thesis:

- Zdeněk Sawa: *Complexity and Decidability of Some Equivalence-Checking Problems*, Ph.D. Thesis, 2005.

List of Citations

- Total number of citations:
  44 citations, including 15 self-citations (according to Web of Science),
  40 citations, including 8 self-citations (according to Scopus),
  81 citations, including 21 self-citations (according to Google Scholar)

Journal Publications

- Zdeněk Sawa: *Efficient Construction of Semilinear Representations of Languages Accepted by Unary Nondeterministic Finite Automata*, *Fundamenta Informaticae*, IOS Press, not published yet, accepted for publication in May 2011.

  – no citations

- Petr Jančar, Martin Kot, Zdeněk Sawa: *Complexity of deciding bisimilarity between normed BPA and normed BPP*, *Information and Computation*, Special Issue: 19th International Conference on Concurrency Theory (CONCUR 2008), Volume 208, Issue 10, pp. 1193–1205, Elsevier, 2010.
  DOI: `10.1016/j.ic.2009.10.012`
  In: Web of Science, Scopus, DBLP

  – no citations

- Sibylle Fröschle, Petr Jančar, Sławomir Lasota, Zdeněk Sawa: *Non-Interleaving Bisimulation Equivalences on Basic Parallel Processes*, *Information and Computation*, Volume 208, Issue 1, pp. 42–62, Elsevier, 2010.
  DOI: `10.1016/j.ic.2009.06.001`
  In: Web of Science, Scopus, DBLP

  – no citations

- Zdeněk Sawa, Petr Jančar: *Hardness of equivalence checking for composed finite-state systems*, *Acta Informatica*, Volume 46, Issue 3, pp. 169–191, Springer, 2009.
  DOI: `10.1007/s00236-008-0088-x`
  In: Web of Science, Scopus, DBLP

  Citations:

  – Laura Bozzelli, Axel Legay, Sophie Pinchinat: *Hardness of preorder checking for basic formalisms*, *Theoretical Computer Science*, Volume 412, Issue 49, pp. 6795–6808, Elsevier, 2011.
  DOI: `10.1016/j.tcs.2011.08.037`
  citation recorded in: Web of Science, Scopus

  - Laura Bozzelli, Axel Legay, Sophie Pinchinat: *Hardness of Preorder Checking for Basic Formalisms*, 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16), LNAI 6355, pp. 119–135, Springer, 2010.
    DOI: `10.1007/978-3-642-17511-4_8`
    citation recorded in: Scopus

- Petr Jančar, Zdeněk Sawa: *A note on emptiness for alternating finite automata with a one-letter alphabet*, *Information Processing Letters*, Volume 104, Issue 5, pp. 164–167, Elsevier, 2007.
  DOI: `10.1016/j.ipl.2007.06.006`
  In: Web of Science, Scopus, DBLP

  Citations:

  - Ashutosh Trivedi, Dominik Wojtczak: *Recursive Timed Automata*, Automated Technology for Verification and Analysis (ATVA 2010), LNCS 6252, pp. 306–324, Springer-Verlag, 2010.
    DOI: `10.1007/978-3-642-15643-4_23`
    citation recorded in: Web of Science, Scopus
  - Tomáš Brázdil, Václav Brožek, Kousha Etessami, Antonín Kučera, Dominik Wojtczak: *One-Counter Markov Decision Processes*, Processings of 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10), pp. 863–874, SIAM, 2010.
    citation recorded in: Scopus
  - Stefan Göller and Markus Lohrey: *Branching-time Model Checking of One-counter Processes*, 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010), pp. 405–416, 2010.
    DOI: `10.4230/LIPIcs.STACS.2010.2472`
  - Anthony Widjaja To: *Model Checking FO(R) over One-Counter Processes and beyond*, Proceedings of Computer Science Logic (CSL 2009), LNCS 5771, pp. 485–499, Springer-Verlag, 2009.
    DOI: `10.1007/978-3-642-04027-6_35`
    citation recorded in: Web of Science, Scopus
  - Jiří Srba: *Beyond Language Equivalence on Visibly Pushdown Automata*, *Logical Methods in Computer Science*, Volume 5, Issue 1, Paper 2, pp. 1–22, Tech Univ Braunschweig, 2009.
    DOI: `10.2168/LMCS-5(1:2)2009`
    citation recorded in: Web of Science, Scopus
  - Stefan Göller, Richard Mayr, Anthony Widjaja To: *On the Computational Complexity of Verifying One-Counter Processes*, Proceedings of 24th Annual IEEE Symposium on Logic in Computer Science (LICS 2009), pp. 235–244, IEEE Computer Society, 2009.
    DOI: `10.1109/LICS.2009.37`
    citation recorded in: Web of Science, Scopus

- Stefan Göller: *Reachability on prefix-recognizable graphs*, *Information Processing Letters*, Volume 108, Issue 2, pp. 71–74, Elsevier, 2008.
  DOI: `10.1016/j.ipl.2008.04.008`
  citation recorded in: Web of Science, Scopus

Self-Citations:

- Tomáš Brázdil, Petr Jančar, Antonín Kučera: *Reachability Games on Extended Vector Addition Systems with States*, 37th International Colloquium on Automata, Languages and Programming (ICALP 2010), Part II, LNCS 6199, pp. 478–489, Springer-Verlag, 2010.
  DOI: `10.1007/978-3-642-14162-1_40`
  citation recorded in: Web of Science, Scopus

• Zdeněk Sawa, Petr Jančar: *Behavioural Equivalences on Finite-State Systems are PTIME-hard*, *Computing and Informatics*, Volume 24, Issue 5, pp. 513–528, Slovak Academy of Sciences, Institute of Informatics, 2005.
In: Web of Science, Scopus, DBLP

Citations:

- Laura Bozzelli, Axel Legay, Sophie Pinchinat: *Hardness of preorder checking for basic formalisms*, *Theoretical Computer Science*, Volume 412, Issue 49, pp. 6795–6808, Elsevier, 2011.
  DOI: `10.1016/j.tcs.2011.08.037`
  citation recorded in: Scopus
- Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Møller, Jiří Srba: *Parametric Modal Transition Systems*, 9th International Symposium on Automated Technology for Verification and Analysis (ATVA 2011), LNCS 6996, pp. 275–289, Springer-Verlag, 2011.
  DOI: `10.1007/978-3-642-24372-1_20`
  citation recorded in: Scopus
- Laura Bozzelli, Axel Legay, Sophie Pinchinat: *Hardness of Preorder Checking for Basic Formalisms*, 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16), LNAI 6355, pp. 119–135, Springer, 2010.
  DOI: `10.1007/978-3-642-17511-4_8`
  citation recorded in: Scopus
- Pierre-Cyrille Héam, Olga Kouchnarenko, Jérôme Voinot: *Component simulation-based substitutivity managing QoS and composition issues*, *Science of Computer Programming*, Volume 75, Issue 10, pp. 898–917, Elsevier, 2010.
  DOI: `10.1016/j.scico.2010.02.004`
  citation recorded in: Scopus

- Pierre-Cyrille Héam, Olga Kouchnarenko, Jérôme Voinot: *Component Simulation-based Substitutivity Managing QoS Aspects*, Proceedings of the 5th International Workshop on Formal Aspects of Component Software (FACS 2008), *Electronic Notes in Theoretical Computer Science*, Volume 260, pp. 109–123, Elsevier, 2010.
  DOI: `10.1016/j.entcs.2009.12.034`
  citation recorded in: Scopus
- Nikola Beneš, Jan Křetínský, Kim G. Larsen, Jiří Srba: *On determinism in modal transition systems*, *Theoretical Computer Science*, Volume 410, Issue 41, pp. 4026–4043, Elsevier, 2009.
  DOI: `10.1016/j.tcs.2009.06.009`
  citation recorded in: Web of Science, Scopus
- Nikola Beneš, Jan Křetínský, Kim G. Larsen, Jiří Srba: *Checking Thorough Refinement on Modal Transition Systems Is EXPTIME-Complete*, Theoretical Aspects of Computing (ICTAC 2009), LNCS 5684, pp. 112–126, Springer-Verlag, 2009.
  DOI: `10.1007/978-3-642-03466-4_7`
  citation recorded in: Web of Science, Scopus
- Jiří Srba: *Beyond Language Equivalence on Visibly Pushdown Automata*, *Logical Methods in Computer Science*, Volume 5, Issue 1, Paper 2, pp. 1–22, Tech Univ Braunschweig, 2009.
  DOI: `10.2168/LMCS-5(1:2)2009`
  citation recorded in: Web of Science, Scopus

Self-Citations:

- Zdeněk Sawa, Petr Jančar: *Hardness of equivalence checking for composed finite-state systems*, *Acta Informatica*, Volume 46, Issue 3, pp. 169–191 Springer, 2009.
  DOI: `10.1007/s00236-008-0088-x`
  citation recorded in: Web of Science, Scopus

• Petr Jančar, Antonín Kučera, Faron Moller, Zdeněk Sawa: *DP lower bounds for equivalence-checking and model-checking of one-counter automata*, *Information and Computation*, Volume 188, Issue 1, pp. 1–19, Elsevier, 2004.
DOI: `10.1016/S0890-5401(03)00171-8`
In: Web of Science, Scopus, DBLP

Citations:

- Stanislav Böhm, Stefan Göller: *Language Equivalence of Deterministic Real-Time One-Counter Automata is NL-Complete*, Mathematical Foundations of Computer Science 2011 (MFCS 2011), LNCS 6907, pp. 194–205, Springer-Verlag, 2011.
  DOI: `10.1007/978-3-642-22993-0_20`
  citation recorded in: Scopus

– Stéphane Demri, Ranko Lazic, Arnaud Sangnier: *Model checking memoryful linear-time logics over one-counter automata*, *Theoretical Computer Science*, Volume 411, Issues 22–24, pp. 2298–2316, Elsevier, 2010.
DOI: `10.1016/j.tcs.2010.02.021`
citation recorded in: Web of Science, Scopus

– Stefan Göller, Christoph Haase, Joël Ouaknine, James Worrell: *Model Checking Succinct and Parametric One-Counter Automata*, 37th International Colloquium on Automata, Languages and Programming (ICALP 2010), LNCS 6199, pp. 575–586, Springer-Verlag, 2010.
DOI: `10.1007/978-3-642-14162-1_48`
citation recorded in: Web of Science, Scopus

– Stéphane Demri, Régis Gascon: *The Effects of Bounding Syntactic Resources on Presburger LTL*, *Journal of Logic and Computation*, Volume 19, Issue 6, pp. 1541–1575, Oxford University Press, 2009.
DOI: `10.1093/logcom/exp037`
citation recorded in: Web of Science, Scopus

– Anthony Widjaja To: *Model Checking FO(R) over One-Counter Processes and beyond*, Proceedings of Computer Science Logic (CSL 2009), LNCS 5771, pp. 485–499, Springer-Verlag, 2009.
DOI: `10.1007/978-3-642-04027-6_35`
citation recorded in: Web of Science

– Christoph Haase, Stephan Kreutzer, Joël Ouaknine, James Worrell: *Reachability in Succinct and Parametric One-Counter Automata*, Proceedings of CONCUR 2009, LNCS 5710, pp. 369–383, Springer, 2009.
DOI: `10.1007/978-3-642-04081-8_25`
citation recorded in: Web of Science, Scopus

– Jiří Srba: *Beyond Language Equivalence on Visibly Pushdown Automata*, *Logical Methods in Computer Science*, Volume 5, Issue 1, Paper 2, pp. 1–22, Tech Univ Braunschweig, 2009.
DOI: `10.2168/LMCS-5(1:2)2009`
citation recorded in: Web of Science, Scopus

– Stefan Göller, Richard Mayr, Anthony Widjaja To: *On the Computational Complexity of Verifying One-Counter Processes*, Proceedings of 24th Annual IEEE Symposium on Logic in Computer Science (LICS 2009), pp. 235–244, IEEE Computer Society, 2009.
DOI: `10.1109/LICS.2009.37`
citation recorded in: Web of Science, Scopus

– Stéphane Demri, Ranko Lazic, Arnaud Sangnier: *Model Checking Freeze LTL over One-Counter Automata*, Proceedings of 11th International Conferenceon the Foundations of Software Science and

Computational Structures (FOSSACS 2008), LNCS 4962, pp. 490–504, Springer-Verlag, 2008.
DOI: `10.1007/978-3-540-78499-9_34`
citation recorded in: Web of Science, Scopus

- Stéphane Demri, Régis Gascon: *Verification of qualitative $\mathbb{Z}$ constraints*, *Theoretical Computer Science*, Volume 409, Issue 1, pp. 24–40, Elsevier, 2008.
DOI: `10.1016/j.tcs.2008.07.023`
citation recorded in: Scopus
- Jiří Srba: *Visibly Pushdown Automata: From Language Equivalence to Simulation and Bisimulation*, Proceedings of Computer Science Logic (CSL 2006), LNCS 4207, pp. 89–103, Springer-Verlag, 2006.
DOI: `10.1007/11874683_6`
citation recorded in: Web of Science, Scopus
- Olivier Serre: *Parity games played on transition graphs of one-counter processes*, Proceedings of 9th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2006), LNCS 3921, pp. 337–351, Springer-Verlag, 2006.
DOI: `10.1007/11690634_23`
citation recorded in: Web of Science, Scopus
- Stéphane Demri, Régis Gascon: *Verification of Qualitative $\mathbb{Z}$ Constraints*, 16th International Conference on Concurrency Theory (CONCUR 2005), LNCS 3653, pp. 518–532, Springer-Verlag, 2005.
DOI: `10.1007/11539452_39`
citation recorded in: Scopus

Self-Citations:

- Tomáš Brázdil, Petr Jančar, Antonín Kučera: *Reachability Games on Extended Vector Addition Systems with States*, 37th International Colloquium on Automata, Languages and Programming (ICALP 2010), Part II, LNCS 6199, pp. 478–489, Springer-Verlag, 2010.
DOI: `10.1007/978-3-642-14162-1_40`
citation recorded in: Web of Science, Scopus
- Tomáš Brázdil, Václav Brožek, Kousha Etessami, Antonín Kučera, Dominik Wojtczak: *One-Counter Markov Decision Processes*, Processings of 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10), pp. 863–874, SIAM, 2010.
citation recorded in: Scopus
- Petr Jančar, Zdeněk Sawa: *A note on emptiness for alternating finite automata with a one-letter alphabet*, *Information Processing Letters*, Volume 104, Issue 5, pp. 164–167, Elsevier, 2007.
DOI: `10.1016/j.ipl.2007.06.006`
citation recorded in: Web of Science, Scopus

  - Antonín Kučera, Petr Jančar: *Equivalence-checking on infinite-state systems: Techniques and results*, *Theory and Practice of Logic Programming*, Volume 6, Issue 3, pp. 227–264, Cambridge University Press, 2006.
    DOI: `10.1017/S1471068406002651`
    citation recorded in: Web of Science, Scopus

Conference Papers

- Zdeněk Sawa: *Efficient Construction of Semilinear Representations of Languages Accepted by Unary NFA*, 4th International Workshop on Reachability Problems (RP 2010), LNCS 6227, pp. 176–182, Springer-Verlag, 2010.
  DOI: `10.1007/978-3-642-15349-5_12`
  In: Web of Science, Scopus, DBLP

  Citations:

  - Pawel Gawrychowski: *Chrobak Normal Form Revisited, with Applications*, 16th International Conference on Implementation and Application of Automata (CIAA 2011), LNCS 6807, pp. 142–153, Springer, 2011.
    DOI: `10.1007/978-3-642-22256-6_14`
    citation recorded in: Scopus

- Petr Jančar, Martin Kot, Zdeněk Sawa: *Normed BPA vs. Normed BPP Revisited*, Proceedings of CONCUR 2008, LNCS 5201, pp. 434–446, Springer-Verlag, 2008.
  DOI: `10.1007/978-3-540-85361-9_34`
  In: Web of Science, Scopus, DBLP

  Citations:

  - Wojciech Czerwinski, Sibylle Froschle, Sławomir Lasota: *Partially-commutative context-free processes: Expressibility and tractability*, *Information and Computation*, Volume 209, Issue 5, pp. 782–798, Elsevier, 2011.
    DOI: `10.1016/j.ic.2010.12.003`
    citation recorded in: Web of Science
  - Wojciech Czerwinski, Sibylle Fröschle, Sławomir Lasota: *Partially-Commutative Context-Free Processes*, Proceedings of CONCUR 2009 – Concurrency Theory, LNCS 5710, pp. 259–273, Springer, 2009.
    DOI: `10.1007/978-3-642-04081-8_18`
    citation recorded in: Web of Science, Scopus

  Self-Citations:

  - Petr Jančar, Martin Kot, Zdeněk Sawa: *Complexity of deciding bisimilarity between normed BPA and normed BPP*, Information and Computation, Special Issue: 19th International Conference on Concurrency Theory (CONCUR 2008), Volume 208, Issue 10, pp. 1193–1205, Elsevier, 2010.
    DOI: `10.1016/j.ic.2009.10.012`
    citation recorded in: Web of Science, Scopus
  - Petr Jančar: *Selected Ideas Used for Decidability and Undecidability of Bisimilarity*, Proceedings of Developments in Language Theory (DLT 2008), LNCS 5257, pp. 56–71, Springer-Verlag, 2008.
    DOI: `10.1007/978-3-540-85780-8_4`

- Zdeněk Sawa, Petr Jančar: *History Preserving Bisimilarity on Basic Parallel Processes*, Proceedings of TAAPSD'06, Kiev, pp. 9–14, 2006.

  – no citations

- Petr Jančar, Zdeněk Sawa: *On Distributed Bisimilarity over Basic Parallel Processes*, Proceedings of workshop AVIS 2005 (A Satellite Workshop of ETAPS 2005), 2005.

  – no citations

- Martin Kot, Zdeněk Sawa: *Bisimulation Equivalence of a BPP and a Finite State System can be Decided in Polynomial Time*, Proceedings of the 6th International Workshop on Verification of Infinite-State Systems (INFINITY 2004), *Electronic Notes in Theoretical Computer Science*, Volume 138, Issue 3, pp. 49–60, Elsevier, 2005.
  DOI: `10.1016/j.entcs.2005.02.065`
  In: Scopus, DBLP

  Citations:

  - Sławomir Lasota: *EXPSPACE lower bounds for the simulation preorder between a communication-free Petri net and a finite-state system*, *Information Processing Letters*, Volume 109, Issue 15, pp. 850–855, Elsevier, 2009.
    DOI: `10.1016/j.ipl.2009.04.003`
    citation recorded in: Scopus
  - Antonín Kučera, Petr Jančar: *Equivalence-checking on infinite-state systems: Techniques and results*, *Theory and Practice of Logic Programming*, Volume 6, Issue 3, pp. 227–264, Cambridge University Press, 2006.
    DOI: `10.1017/S1471068406002651`

- Zdeněk Sawa: *Equivalence Checking of Non-flat Systems Is EXPTIME-hard*, Proceedings of CONCUR 2003, LNCS 2761, pp. 237–250, Springer-Verlag, 2003.
  DOI: `10.1007/978-3-540-45187-7_16`
  In: Web of Science, Scopus, DBLP

Citations:

- Philippe Balbiani, Fahima Cheikh Alili, Pierre-Cyrille Héam, Olga Kouchnarenko: *Composition of Services with Constraints*, Proceedings of the 6th International Workshop on Formal Aspects of Component Software (FACS 2009), *Electronic Notes in Theoretical Computer Science* Volume 263, pp. 31–46, Elsevier, 2010.
  DOI: `10.1016/j.entcs.2010.05.003`
- Sławomir Lasota: *EXPSPACE lower bounds for the simulation preorder between a communication-free Petri net and a finite-state system*, *Information Processing Letters*, Volume 109, Issue 15, pp. 850–855, Elsevier, 2009.
  DOI: `10.1016/j.ipl.2009.04.003`
  citation recorded in: Web of Science, Scopus
- Stéphane Demri, François Laroussinie, Philippe Schnoebelen: *A parametric analysis of the state-explosion problem in model checking*, *Journal of Computer and System Sciences*, Volume 72, Issue 4, pp. 547–575, Elsevier, 2006.
  DOI: `10.1016/j.jcss.2005.11.003`
- François Laroussinie: *Model checking temporisé – algorithmes efficaces et complexité*, Mémoire d'habilitation, Université Paris 7, Paris, France, 2005.
  URL: `www.liafa.jussieu.fr/ francoisl/PUBLIS/hdr.pdf`

Self-Citations:

- Zdeněk Sawa, Petr Jančar: *Hardness of equivalence checking for composed finite-state systems*, *Acta Informatica*, Volume 46, Issue 3, pp. 169–191, Springer, 2009.
  DOI: `10.1007/s00236-008-0088-x`
  citation recorded in: Web of Science, Scopus
- Zdeněk Sawa, Petr Jančar: *Behavioural equivalences on finite-state systems are PTIME-hard*, *Computing and Informatics*, Volume 24, Issue 5, pp. 513–528, Slovak Academy Sciences Inst Informatics, 2005.
  citation recorded in: Web of Science

• Petr Jančar, Antonín Kučera, Faron Moller, Zdeněk Sawa: *Equivalence-Checking with One-Counter Automata: A Generic Method for Proving Lower Bounds*, Proceedings of FOSSACS 2002, LNCS 2303, pp. 172–186, Springer-Verlag, 2002.
DOI: `10.1007/3-540-45931-6_13`
In: Web of Science, DBLP

Citations:

- Stéphane Demri: *LTL over integer periodicity constraints*, *Theoretical Computer Science*, Volume 360, Issues 1–3, pp. 96–123, Else-

vier, 2006.
DOI: `10.1016/j.tcs.2006.02.019`

Self-Citations:

– Petr Jančar, Antonín Kučera, Faron Moller, Zdeněk Sawa: *DP lower bounds for equivalence-checking and model-checking of one-counter automata*, *Information and Computation*, Volume 188, Issue 1, pp. 1–19, Elsevier, 2004.
DOI: `10.1016/S0890-5401(03)00171-8`
citation recorded in: Web of Science

– Antonín Kučera: *The complexity of bisimilarity-checking for one-counter processes*, *Theoretical Computer Science*, Volume 304, Issues 1–3, pp. 157–183, Elsevier, 2003.
DOI: `10.1016/S0304-3975(03)00081-1`
citation recorded in: Web of Science

– Antonín Kučera, Petr Jančar: *Equivalence-Checking with Infinite-State Systems: Techniques and Results*, SOFSEM 2002: Theory and Practice of Informatics, LNCS 2540, pp. 41–73, Springer-Verlag, 2002.
DOI: `10.1007/3-540-36137-5_3`
citation recorded in: Web of Science

• Zdeněk Sawa, Petr Jančar: *P-hardness of Equivalence Testing on Finite-State Processes*, Proceedings of SOFSEM 2001, LNCS 2234, pp. 326–335, Springer, 2001.
DOI: `10.1007/3-540-45627-9_29`
In: DBLP

Citations:

– Rohit Chadha, Axel Legay, Pavithra Prabhakar and Mahesh Viswanathan: *Complexity Bounds for the Verification of Real-Time Software*, Proceedings of the 11th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2010), LNCS 5944, pp. 95–111, Springer-Verlag, 2010.
DOI: `10.1007/978-3-642-11319-2_10`

– Pierre-Cyrille Héam, Olga Kouchnarenko, Jérôme Voinot: *Component simulation-based substitutivity managing QoS and composition issues*, *Science of Computer Programming*, Volume 75, Issue 10, pp. 898–917, Elsevier, 2010.
DOI: `10.1016/j.scico.2010.02.004`

– Pierre-Cyrille Héam, Olga Kouchnarenko, Jérôme Voinot: *Component Simulation-based Substitutivity Managing QoS Aspects*, Proceedings of the 5th International Workshop on Formal Aspects of Component Software (FACS 2008), *Electronic Notes in Theoretical Computer Science*, Volume 260, pp. 109–123, Elsevier, 2010.
DOI: `10.1016/j.entcs.2009.12.034`

- Rastislav Lenhardt: *Probabilistic Automata with Parameters*, M.Sc. thesis, supervisors Dr James Worrell, Dr Joël Ouaknine, University of Oxford, 2009.
  URL: `http://www.cs.ox.ac.uk/files/2736/PAwP.pdf`
- Jiří Srba: *Visibly Pushdown Automata: From Language Equivalence to Simulation and Bisimulation*, Proceedings of Computer Science Logic (CSL 2006), LNCS 4207, pp. 89–103, Springer-Verlag, 2006.
  DOI: `10.1007/11874683_6`
- Antonín Kučera, Richard Mayr: *Why Is Simulation Harder than Bisimulation?*, Proceedings of CONCUR 2002, LNCS 2421, pp. 594–609, Springer-Verlag, 2002.
  DOI: `10.1007/3-540-45694-5_39`
- Stéphane Demri, François Laroussinie, Philippe Schnoebelen: *A Parametric Analysis of the State Explosion Problem in Model Checking*, Proceedings of 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2002), LNCS 2285, pp. 620–631, Springer-Verlag, 2002.
  DOI: `10.1007/3-540-45841-7_51`

Self-Citations:

- Zdeněk Sawa, Petr Jančar: *Behavioural equivalences on finite-state systems are PTIME-hard*, *Computing and Informatics*, Volume 24, Issue 5, pp. 513–528, Slovak Academy Sciences Inst Informatics, 2005.
- Zdeněk Sawa: *Equivalence Checking of Non-flat Systems Is EXPTIME-hard*, Proceedings of CONCUR 2003, LNCS 2761, pp. 237–250, Springer-Verlag, 2003.
  DOI: `10.1007/978-3-540-45187-7_16`

• Petr Jančar, Faron Moller, Zdeněk Sawa: *Simulation Problems for One-Counter Machines*, Proceedings of SOFSEM'99, LNCS 1725, pp. 404–413, Springer-Verlag, 1999.
DOI: `10.1007/3-540-47849-3_28`
In: Web of Science, DBLP

Citations:

- Stanislav Böhm, Stefan Göller: *Language Equivalence of Deterministic Real-Time One-Counter Automata is NL-Complete*, Mathematical Foundations of Computer Science 2011 (MFCS 2011), LNCS 6907, pp. 194–205, Springer-Verlag, 2011.
  DOI: `10.1007/978-3-642-22993-0_20`
- Benedek Nagy, Friedrich Otto: *An Automata-Theoretical Characterization of Context-Free Trace Languages*, SOFSEM 2011: Theory and Practice of Computer Science, LNCS 6543, pp. 406–417,

Springer-Verlag, 2011.
DOI: `10.1007/978-3-642-18381-2_34`
citation recorded in: Web of Science

- Jiří Srba: *Beyond Language Equivalence on Visibly Pushdown Automata*, *Logical Methods in Computer Science*, Volume 5, Issue 1, Paper 2, pp. 1–22, Tech Univ Braunschweig, 2009.
  DOI: `10.2168/LMCS-5(1:2)2009`
  citation recorded in: Web of Science
- Alban Ponse, Mark B. van der Zwaag: *Risk Assessment for One-Counter Threads*, *Theory of Computing Systems*, Volume 43, Numbers 3–4, pp. 563–582, Springer, 2008.
  DOI: `10.1007/s00224-007-9034-5`
- Jan A. Bergstra, Inge Bethke, Alban Ponse: *Decision problems for pushdown threads*, *Acta Informatica*, Volume 44, Number 2, pp. 75–90, Springer, 2007.
  DOI: `10.1007/s00236-007-0040-5`
- Jiří Srba: *Visibly Pushdown Automata: From Language Equivalence to Simulation and Bisimulation*, Proceedings of Computer Science Logic (CSL 2006), LNCS 4207, pp. 89–103, Springer-Verlag, 2006.
  DOI: `10.1007/11874683_6`
  citation recorded in: Web of Science
- Alban Ponse, Mark B. van der Zwaag: *An Introduction to Program and Thread Algebra*, Proceedings of Logical Approaches to Computational Barriers, Second Conference on Computability in Europe (CiE 2006), LNCS 3988, pp. 445–458, Springer-Verlag, 2006.
  DOI: `10.1007/11780342_46`
- Jan A. Bergstra, Inge Bethke, Alban Ponse: *Thread algebra and risk assessment services*, Proceedings of Logic Colloquium 2005, pp. 1–17, Cambridge University Press, 2005.
- Antonín Kučera: *The complexity of bisimilarity-checking for one-counter processes*, *Theoretical Computer Science*, Volume 304, Issues 1–3, pp. 157–183, Elsevier, 2003.
  DOI: `10.1016/S0304-3975(03)00081-1`
  citation recorded in: Web of Science
- Antonín Kučera, Richard Mayr: *Simulation Preorder over Simple Process Algebras*, *Information and Computation*, Volume 173, Issue 2, pp. 184–198, Elsevier, 2002.
  DOI: `10.1006/inco.2001.3122`
  citation recorded in: Web of Science
- Olivier Finkel: *An Effective Extension of the Wagner Hierarchy to Blind Counter Automata*, Proceedings of Computer Science Logic (CSL 2001), LNCS 2142, pp. 369–383, Springer-Verlag, 2001.
  DOI: `10.1007/3-540-44802-0_26`

– Antonín Kučera: *On Simulation-Checking with Sequential Systems*, Proceedings of Advances in Computing Science (ASIAN 2000), LNCS 1961, pp. 133–148, Springer-Verlag, 2000.
DOI: `10.1007/3-540-44464-5_11`
citation recorded in: Web of Science

– Antonín Kučera: *Efficient Verification Algorithms for One-Counter Processes*, Proceedings of Twenty-Seventh International Colloquium on Automata, Languages and Programming (ICALP 2000), LNCS 1853, pp. 317–328, Springer, 2000.
DOI: `10.1007/3-540-45022-X_28`
citation recorded in: Web of Science

Self-Citations:

– Antonín Kučera, Petr Jančar: *Equivalence-checking on infinite-state systems: Techniques and results*, *Theory and Practice of Logic Programming*, Volume 6, pp. 227–264, Cambridge Univ Press, 2006.
DOI: `10.1017/S1471068406002651`
citation recorded in: Web of Science

– Petr Jančar, Antonín Kučera, Faron Moller, Zdeněk Sawa: *DP lower bounds for equivalence-checking and model-checking of one-counter automata*, *Information and Computation*, Volume 188, Issue 1, pp. 1–19, Elsevier, 2004.
DOI: `10.1016/S0890-5401(03)00171-8`
citation recorded in: Web of Science

– Antonín Kučera, Petr Jančar: *Equivalence-Checking with Infinite-State Systems: Techniques and Results*, SOFSEM 2002: Theory and Practice of Informatics, LNCS 2540, pp. 41–73, Springer-Verlag, 2002.
DOI: `10.1007/3-540-36137-5_3`
citation recorded in: Web of Science

– Petr Jančar, Antonín Kučera, Faron Moller, Zdeněk Sawa: *Equivalence-Checking with One-Counter Automata: A Generic Method for Proving Lower Bounds*, Proceedings of FOSSACS 2002 (Foundations of Software Science and Computation Structures), LNCS 2303, pp. 172–186, Springer-Verlag, 2002.
DOI: `10.1007/3-540-45931-6_13`
citation recorded in: Web of Science

– Olaf Burkart, Didier Caucal, Faron Moller, Bernhard Steffen: *Verification on Infinite Structures*, Chapter 9 in Handbook of Process Algebra (edited by Jan A. Bergstra, Alban Ponse and Scott A. Smolka), Elsevier, 2001.

– Petr Jančar, Antonín Kučera, Faron Moller: *Simulation and Bisimulation over One-Counter Processes*, Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science

(STACS'2000), LNCS 1770, pp. 334–345, Springer-Verlag, 2000.
DOI: 10.1007/3-540-46541-3_28