Equivalence problems for finite-state and one-counter processes

Zdeněk Sawa

1 Introduction

In the study of processes two main activities are *modelling* and *verification*. Modelling means representing of processes, usually by means of some mathematical formalism, in a way that abstracts from unimportant details of the system. Verification of processes means proving statements about processes, such as that a given process has a given property or that the behaviors of two systems are equivalent.

In the study of concurrent systems, different formalisms were proposed for modelling of systems. Examples of such formalisms are Petri nets, different process calculi (such as CCS [29], CSP [17] and ASP [4]) or rewrite transition systems [10]. In any case, not matter what formalism is used for the description of the system, the behavior of a system is interpreted like an edge-labelled graph, whose nodes correspond to different states of the system, and whose edges correspond to transitions from one state to another. A label on an edge represents an action connected with the transition (typically a communication with an environment of the process).

Definition 1.1 *A* labelled transition system *is a tuple* $\langle S, Act, \longrightarrow\rangle$ *where:*

- S *is a set of* states *(finite or infinite).*
- Act *is a finite set of* actions.
- $\longrightarrow \subseteq S \times Act \times S$ *is the* transition relation. *We write* $\alpha \xrightarrow{a} \beta$ *instead of* $\langle \alpha, a, \beta\rangle \in \longrightarrow$, *and we extend this notation to the reflexive and transitive closure of* $\longrightarrow$, *i.e. we write* $\alpha \xrightarrow{w} \beta$, *where* $w \in Act^*$.

We identify a *process* with one state of a transition system. The transition system then describes all possible behavior of the process.

The concurrent behavior of the system is modelled using interleaving semantics. That means that concurrent execution of actions a and b, is modelled like a sequence of actions ab or ba.

In the field of the concurrency theory, two main problems are intensively studied:

- Given two systems, are the behaviors of these systems equal with respect to a certain equivalence notion? This problem is called *equivalence problem* or *equivalence checking.*
- Given a system and a property expressed in a certain temporal or modal logic (as a formula ϕ), does the system satisfy the property (is it the model of the formula ϕ)? This problem is called *model checking.*

The problem of model checking is not considered in this paper.

Now there is a question, when we should consider two systems to be equivalent. It turns out that the notion of language equivalence,[1] as traditionally used in the theory of formal languages, is not suitable for concurrent systems.

In the case of language equivalence, we usually have a system (like a finite automaton, pushdown automaton, Turing machine etc.) that has a word written on its input tape at the start, performs some computation on the input and accepts or rejects it. Two systems are considered to

[1] If we identify actions with symbols and sequences of actions with words.

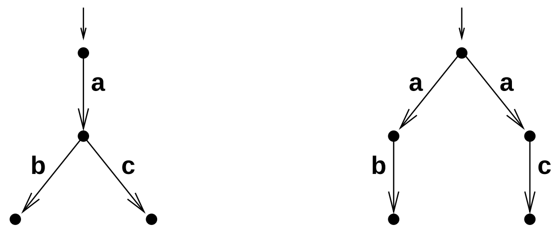


Figure 1: An example of two processes with quite different capabilities

be equivalent if they accept and reject the same sets of words. The situation is quite different in the case of concurrent systems. Their input is not known at the start of the computation, but is received during the computation, they receive actions and react to them.

This difference can be illustrated in the following example depicted in the figure 1 (start states of both processes are denoted with small arrows). Both processes are able to perform the same sequences of actions (ab and ac). The difference between them is that the first one can perform the action a and then choose between actions b and c, but the second one must choose between two a actions at the start and then can perform only one of b or c, depending on its previous chose.

It is clear from this example that the language equivalence is too coarse for concurrent systems, because two systems with quite different capabilities are considered to be equivalent under this notion of equivalence.

Moreover, the concurrent systems are often systems, which are not supposed to terminate at all, such as network protocols, control systems etc. The behavior of such systems can not be modelled in terms of accepting or rejecting their input.

Remark: In the equivalence problem, we can consider instead of two separate systems only one system (formed as a disjoint union of these systems). The equivalence problem is then formulated as a problem of equivalence of a pair of states of the given transition system.

There has been many different notions of equivalence proposed in the literature. Van Glabbeek classified these equivalences in [31] and organized them in a hierarchy called *linear time/branching time spectrum.* A diagram containing some of the most important equivalences in the hierarchy is in the figure 2. (An arrow from equivalence R to equivalence S means that any states related by equivalence R are also related by S, but the converse is not true for some systems. That is, S is "coarser" equivalence than R and R is "finer" than S.) All equivalences in this hierarchy lie between bisimulation equivalence (which is thus the finest of all these equivalences) and trace equivalence (which is the coarsest).

We call the transition system *deterministic*, if for each state α and any action a there is at most one α' s.t. $\alpha \xrightarrow{a} \alpha'$, and we call it *nondeterministic* otherwise. All equivalences in the hierarchy differ only if we consider nondeterministic transition systems, because for deterministic systems are all these equivalences identical and the hierarchy collapses.

In the literature are also defined such equivalences that take into account unobservable actions (called τ-actions), i.e. transitions performed by the process without an interaction with the environment (weak bisimulation equivalence is an example of such equivalence), but these equivalences are not discussed in this paper.

2 Basic definitions

2.1 Equivalences

There is no "right" equivalence. Different equivalences can be useful for different purposes and it depends on the processes and properties we want to study, which one is appropriate. But it

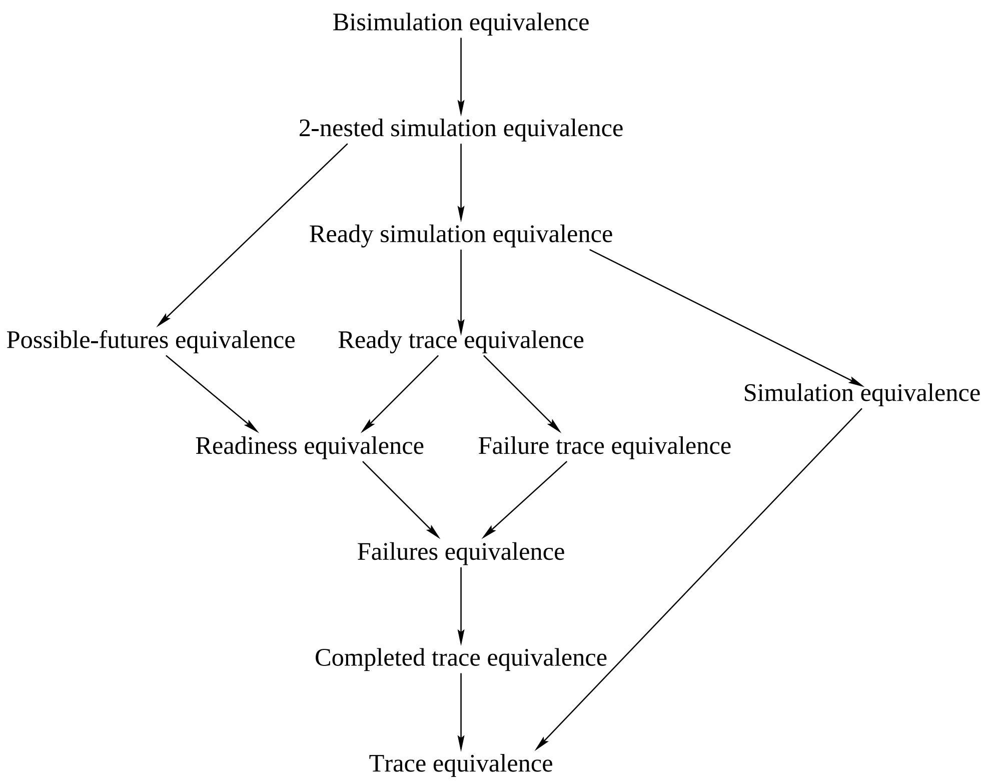

Bisimulation equivalence
2-nested simulation equivalence
Ready simulation equivalence
Possible-futures equivalence
Ready trace equivalence
Simulation equivalence
Readiness equivalence
Failure trace equivalence
Failures equivalence
Completed trace equivalence
Trace equivalence

Figure 2: The linear time/branching time spectrum

turns out that bisimulation equivalence, simulation equivalence and trace equivalence are very important. Their definitions follow:

Definition 2.1 (Bisimulation equivalence) *A binary relation $\mathcal{R}$ on states of a transition system is called a* bisimulation *iff for any $\langle \alpha, \beta \rangle \in \mathcal{R}$ two following conditions hold:*

- *If $\alpha \xrightarrow{a} \alpha'$ for some $a \in Act$ then $\beta \xrightarrow{a} \beta'$ for some β' such that $\langle \alpha', \beta' \rangle \in \mathcal{R}$.*
- *If $\beta \xrightarrow{a} \beta'$ for some $a \in Act$ then $\alpha \xrightarrow{a} \alpha'$ for some α' such that $\langle \alpha', \beta' \rangle \in \mathcal{R}$.*

States α and β are bisimulation equivalent *or* bisimilar*, written $\alpha \sim \beta$, iff $(\alpha, \beta) \in \mathcal{R}$ for some bisimulation $\mathcal{R}$.*

Definition 2.2 (Simulation equivalence) *A binary relation $\mathcal{R}$ on states of a transition system is called a* simulation *iff for any $\langle \alpha, \beta \rangle \in \mathcal{R}$ the following condition holds:*

$$\text{If } \alpha \xrightarrow{a} \alpha' \text{ for some } a \in Act \text{ then } \beta \xrightarrow{a} \beta' \text{ for some } \beta' \text{ such that } \langle \alpha', \beta' \rangle \in \mathcal{R}.$$

State α is simulated *by state β, written $\alpha \preceq_s \beta$, iff $(\alpha, \beta) \in \mathcal{R}$ for some simulation $\mathcal{R}$. States α and β are* simulation equivalent*, written $\alpha \equiv_s \beta$, iff $\alpha \preceq_s \beta$ and $\beta \preceq_s \alpha$.*

Definition 2.3 (Trace equivalence) *If α is a state of a transition system, we define the set of* traces*, denoted $tr(\alpha)$, as a set of all possible sequences of actions from the given state, formally:*

$$tr(\alpha) = \{ w \in Act^* \mid \exists \alpha' \text{ such that } \alpha \xrightarrow{w} \alpha' \}$$

The states α and β are trace equivalent*, written $\alpha \equiv_{tr} \beta$, iff $tr(\alpha) = tr(\beta)$.*

All these equivalences could be also equivalently characterized using *characteristic games*. Such games are played by two players, denoted Player I and Player II in the following. The goal of Player I is to show that the given two states are not equivalent and the goal of Player II is to show the contrary. The states are equivalent if there is a *winning strategy* for Player II.

For example the characteristic game for bisimulation equivalence (bisimulation game) is defined such that in every move Player I can choose a process he wants to play with, he performs one transition of this process and this transition must be matched in the other process by Player II by a transition with the same label. The game continues until one of the players can not perform any transition — then the player that is stuck loses. If the game never ends, Player II wins.

The characteristic game for the simulation equivalence (simulation game) is defined similarly, but Player I can choose the process he wants to play with only at the beginning, and then he must always play with the chosen process.

2.2 Classes of processes

Now, if we want to solve the equivalence problem for some transition systems, there is a question to what extend we can automate this testing. It should be clear that it is not possible to test algorithmically equivalence of systems that have full Turing power, because we can easily reduce the halting problem to such an equivalence problem.

On the other hand, if we consider systems with finitely many states, all the equivalences in the hierarchy are decidable (some of them in polynomial time), at least by some kind of "naive" algorithm based on exhaustive search.

Now there is a natural question, where exactly lies the boundary between systems for which equivalence problem is decidable and systems for which it is undecidable, and what is the complexity of the problem (in the case that it is decidable)?

Some of the most important classes of processes that are studied in the field are (see e.g. [7] for definitions):

- FSA — finite state automata
- BPA — basic process algebras
- BPP — basic parallel processes
- PDA — pushdown automata
- PN — Petri nets
- OCA — one-counter automata
- OCN — one-counter nets

In the proofs in this paper only classes FSA, OCA and OCN are considered.

FSA is a class of processes with finitely many states. It is basically the well known class of nondeterministic finite automata.

OCA (also called one-counter machines) form a proper subclass of PDA. PDA is a class of processes that have a finite control unit (as FSA) equipped with an infinite stack, i.e. the states of PDA are elements of $Q\Gamma^*$, where Q is a finite set of control states and Γ is a finite set of stack symbols. There is defined a finite set of transition rules of the form $pA \xrightarrow{a} q\alpha$, where $p, q \in Q$, $A \in \Gamma$ and $\alpha \in \Gamma^*$. The possible transitions are of the form $pA\beta \xrightarrow{a} q\alpha\beta$, where $\beta \in \Gamma^*$ and $pA \xrightarrow{a} q\alpha$ is a transition rule.

OCA is a subclass of PDA with the restrictions that the set of stack symbols is $\Gamma = \{Z, I\}$ and the allowed transition rules are of the form $pI \xrightarrow{a} qI^*$ or $pZ \xrightarrow{a} qI^*Z$ where $p, q \in Q$. All reachable states are of the form $pII\ldots IZ$, so the stack behaves in fact like a counter, because only the number of Is is important. We use more convenient notation $p(i)$ for states, where $p \in Q$ and $i \in \mathcal{N}$ represents the number of Is on the stack.

It can be shown that every OCA process can be represented (up to the labelling of states) by a process that has at most 2 Is in the right hand sides of transition rules, i.e. the value of the counter can increase or decrease at most by 1 in one step.

OCA are "strong" in the sense that they are able to test for zero, they can perform different actions if the value of the counter is zero and if it is non-zero.

OCN is a family of "weak" OCA that are not able to test for zero. If there is some transition possible when the value of the counter is zero, is is possible also if it is non-zero. Formally, if there is a transition rule $pZ \xrightarrow{a} q\alpha Z$ where $p, q \in Q$, $Z, I \in \Gamma$ and $\alpha \in I^*$, then there is also a transition rule $pI \xrightarrow{a} q\alpha I$. The class OCN corresponds to the class of Petri nets with one unbounded place.

3 State of the art

In this section we present a survey of known decidability and complexity results for equivalence problems. It turns out that all equivalences in linear time/branching time spectrum are decidable for finite-state processes, and the main concern is therefore that on the computational complexity. On the other hand, for many classes of infinite-state systems, no equivalence in the spectrum is decidable, however, for certain interesting classes and equivalences (especially for bisimulation equivalence) is the equivalence problem decidable.

3.1 Finite-state processes

The equivalences in the spectrum can be generally divided to *simulation-like* and *trace-like* equivalences.

The simulation-like equivalences are the equivalences between bisimulation and simulation equivalence (such as bisimulation equivalence, simulation equivalence, ready-simulation equivalence, n-nested simulation etc.) It was shown that all these equivalences are decidable in polynomial time (see [19] for a more detailed survey of the results).

The trace-like equivalences are defined in terms of unbounded sequences of actions (traces). In general all equivalences in the spectrum between ready trace equivalence and trace equivalence, and between possible-futures equivalence and trace equivalence belong to this category. It was proved in [25] that the equivalence problem for finite-state processes is PSPACE-complete for all these equivalences.

It was proved in [2] that the problem of equivalence of finite-state processes is P-complete for bisimulation equivalence.

3.2 Infinite-state processes

In [5] (and in [6]) Baeten, Bergstra and Klop proved that bisimulation equivalence is decidable for *normed* BPA processes. (A process is *normed*, if from any reachable state is reachable a state, where no transitions are possible.) Normed BPA processes correspond to context-free grammars without redundant nonterminals. Simpler proof of this result was presented by Caucal [9] and yet another proof using tableau technique was presented in [20]. Later, Hirshfeld, Jerrum and Moller showed that this problem is in PTIME [15].

The decidability of bisimulation equivalence for BPA processes in general case was established in [13]. The elementary algorithm for the problem (in 2-EXPTIME) was presented in [8].

It was shown in [14] that the equivalence problem for BPA processes is undecidable for all equivalences in the spectrum except bisimilarity.

Bisimulation equivalence of BPP processes is decidable, as was shown in [11] and [12]. It was shown by Mayr [27] that this problem is coNP-hard. For normed BPP is this problem decidable in polynomial time [16].

As was shown in [18], the equivalence problem for BPP is undecidable for all equivalences except bisimilarity.

Sénizergues has shown that bisimilarity of PDA processes is decidable [30]. Mayr has shown in [28] that this problem is PSPACE-hard.

Jančar has shown in [21] that the equivalence problem for PN is undecidable for any equivalence between bisimulation equivalence and trace equivalence. (The undecidability holds even for MSA—multiset automata, that form a proper subclass of PN.)

Bisimulation equivalence was shown to be decidable for OCA processes by Jančar in [22].

The proof of decidability of simulation equivalence for OCN was first presented in [1]. A simpler proof was presented in [23] and also in [24], where was also shown that this problem is undecidable for OCA.

4 Own results

In the following own results are presented. The main own results in the field achieved so far are:

- Simulation equivalence is undecidable for OCA processes.
- Deciding simulation equivalence is DP-hard for OCN processes.
- Deciding equivalence of finite-state processes is P-hard for all equivalences between bisimulation equivalence and trace equivalence.

The first result has been published in [24]. The undecidability was shown using reduction from the halting problem for Minsky machine with two counters. The other two results were not published yet, so the proofs of these results are presented in the following two subsections.

4.1 DP-hardness of simulation equivalence for one-counter nets

In the following is proved that testing of simulation equivalence is DP-hard problem for OCN processes.

DP is the class of decision problems defined as follows:

$$\mathsf{DP} = \{A \mid A = B \cap C \text{ for some } B \text{ in } \mathsf{NP} \text{ and } C \text{ in } \mathsf{coNP}\}$$

A problem P is DP-hard if any problem in DP can be reduced to P in polynomial time. P is DP-complete, if P is DP-hard and $P \in \mathsf{DP}$. (DP-hardness of the problem means that the problem is NP-hard and also coNP-hard.)

DP-hardness of the problem is shown using a polynomial time reduction from the SAT-UNSAT problem, which is known to be DP-complete. The SAT-UNSAT problem is defined as follows:

INSTANCE: A pair (ϕ_1, ϕ_2) of Boolean formulae in conjunctive normal form (CNF), such that any clause in formulae contains exactly 3 literals.

QUESTION: Is ϕ_1 satisfiable and is ϕ_2 not satisfiable?

The reduction is based on two following lemmas, that stay that the well known 3-SAT problem can be reduced in polynomial time to the problem if one OCN process is simulated by another, and also to the complement of this problem.

Lemma 4.1 3-SAT *can be reduced in polynomial time to the problem if* $p(0) \not\preceq_s p'(0)$, *where* $p(0)$ *and* $p'(0)$ *are the start states of OCN processes.*

Lemma 4.2 3-SAT *can be reduced in polynomial time to the problem if* $p(0) \preceq_s p'(0)$, *where* $p(0)$ *and* $p'(0)$ *are the start states of OCN processes.*

The constructions in the proofs of these two lemmas are quite similar and use some ideas that were used previously in [26]. The most important idea is the way how different valuations of boolean variables in the formula are encoded in states of OCN processes.

They are encoded as values of the counter and this encoding uses prime numbers. Let us denote π_i the i^{th} prime number. Let $Var = \{x_1, x_2, \dots, x_n\}$ be a set of boolean variables in the formula and let $k \in \mathcal{N}$ be a value of the counter. Using this value we can define a valuation $\nu_k : \mathit{Var} \to \{\texttt{true}, \texttt{false}\}$ s.t. $x_i = \texttt{true}$ iff $k \bmod \pi_i = 0$. It is clear that for any valuation ν there is some $k \in \mathcal{N}$ s.t. $\nu \equiv \nu_k$. We can take for example k to be product of $f(i)$ for $1 \leq i \leq n$ where

$$f(i) = \begin{cases} \pi_i & \text{if } \nu(x_i) = \texttt{true} \\ 1 & \text{if } \nu(x_i) = \texttt{false} \end{cases}$$

The important fact is that the value of the sum $\sum_{i=1}^{n} \pi_i$ is in $O(n^3)$ (see, e.g., [3]). This ensures that the following constructions can be achieved in polynomial time.

Let us first describe a construction of (parts of) OCN processes that are used for testing of boolean values in the valuation ν_k (under assumption that the values of counters of both processes are k).

We create the transition system Δ. We add the control states s and s_F to Δ. Let $\mathit{Var} = \{x_1, x_2, \ldots, x_n\}$ be a set of boolean variables. For every variable x_i (where $1 \leq i \leq n$) we add two sets of control states to Δ: $\{\langle x_i, j\rangle \mid 0 \leq j < \pi_i\}$ and $\{\langle \bar{x}_i, j\rangle \mid 0 \leq j < \pi_i\}$.

We add the following transitions (for each $1 \leq i \leq n$):

- $sI \xrightarrow{a} s\varepsilon$, $sI \xrightarrow{b} s_F I$ and $sZ \xrightarrow{b} s_F Z$,
- $\langle x_i, j\rangle I \xrightarrow{a} \langle x_i, (j+1) \bmod \pi_i\rangle\varepsilon$ for each $0 \leq j < \pi_i$,
- $\langle x_i, 0\rangle I \xrightarrow{b} s_F\varepsilon$,
- $\langle x_i, j\rangle I \xrightarrow{b} s_F I$ and $\langle x_i, j\rangle Z \xrightarrow{b} s_F Z$ for each $0 < j < \pi_i$,
- $\langle \bar{x}_i, j\rangle I \xrightarrow{a} \langle \bar{x}_i, (j+1) \bmod \pi_i\rangle\varepsilon$ for each $0 \leq j < \pi_i$,
- $\langle \bar{x}_i, 0\rangle I \xrightarrow{b} s_F I$ and $\langle \bar{x}_i, 0\rangle Z \xrightarrow{b} s_F Z$,
- $\langle \bar{x}_i, j\rangle I \xrightarrow{b} s_F\varepsilon$ for each $0 < j < \pi_i$.

It holds for any $k \in \mathcal{N}$ that $s(k) \not\preceq_s \langle x_i, 0\rangle(k)$ iff $k \bmod \pi_i = 0$, i.e. iff $\nu_k(x_i) = \texttt{true}$. Similarly, $s(k) \not\preceq_s \langle \bar{x}_i, 0\rangle(k)$ iff $k \bmod \pi_i \neq 0$, i.e. iff $\nu_k(x_i) = \texttt{false}$. The proofs of these facts are straightforward.

Notice that the number of control states of Δ is in $O(n^3)$, and the construction of Δ can be done in polynomial time.

Proof of lemma 4.1: Let ϕ be an instance of 3-Sat. Let us suppose that ϕ contains a set of boolean variables $\{x_1, x_2, \ldots, x_n\}$ and is of the form $C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each C_j is a clause of the form $L_{j,1} \vee L_{j,2} \vee L_{j,3}$, where each $L_{j,k}$ is either x_i or $\neg x_i$.

We construct processes with start states $p(0), p'(0)$, such that $p(0) \preceq_s p'(0)$ iff ϕ is not satisfiable. We start with the transition system Δ constructed above. We add control states p and r and the transitions:

- $pZ \xrightarrow{a} pIZ$ and $pI \xrightarrow{a} pII$,
- $pZ \xrightarrow{b} rZ$ and $pI \xrightarrow{b} rI$,
- $rZ \xrightarrow{x} sZ$ and $rI \xrightarrow{x} sI$ where $x \in \{a, b, c\}$.

Then we add the control state p' and for every clause C_j $(1 \leq j \leq m)$ we add the control state r_j.

Finally we add the transitions:

- $p'Z \xrightarrow{a} p'IZ$ and $p'I \xrightarrow{a} p'II$,
- $p'Z \xrightarrow{b} r_j Z$ and $p'I \xrightarrow{b} r_j I$ for each $1 \leq j \leq m$,
- for each clause C_j (of the form $L_{j,1} \vee L_{j,2} \vee L_{j,3}$):

$$r_j X \xrightarrow{a} Q_{j,1} X \qquad r_j X \xrightarrow{b} Q_{j,2} X \qquad r_j X \xrightarrow{c} Q_{j,3} X \qquad\qquad X \in \{I, Z\}$$

  where $Q_{j,k}$ is $\langle x_i, 0\rangle$ if $L_{j,k}$ is x_i, and $\langle \bar{x}_i, 0\rangle$ if $L_{j,k}$ is $\neg x_i$.

If we describe the problem, if $p(0) \preceq_s p'(0)$, in terms of the simulation game, the game has the following phases:

1. Player I chooses some valuation of boolean variables in the formula (encoded as a value of the counter of OCN) — he can increase the value of the counter of the first process using a actions. Player II must increase the value of the counter of his process to the same value. Notice, that Player I can not increase the value infinitely long and he must eventually stop increasing it and perform an action b otherwise Player II would win.
2. Player II chooses some clause C_j of the formula. Player I has only one possible transition, but Player II can choose between states $r_1, \dots, r_m$ that correspond to the clauses of the formula. The values of the counters are not changed in this phase.
3. Player I chooses one literal in the clause C_j (if Player II has chosen the control state r_j in the previous move) by choosing one of the actions a, b, c. The values of the counters are not changed in this phase.
4. The value of the chosen literal is tested. Player I wins if the value of the literal is `true`, and Player II wins otherwise.

Now Player I has the winning strategy, if ϕ is satisfiable — he can choose a valuation that makes ϕ true. Then there is in every clause at least one literal with the value `true` in the given valuation. On the other hand if ϕ is not satisfiable, then in every valuation is some clause with all literals with the value `false`, so Player II has the winning strategy now, because in phase 2 he can choose a state corresponding to this clause. □

The proof of lemma 4.2 is rather similar to the proof of lemma 4.1 and is only sketched.

Proof of lemma 4.2 (sketch): The roles of the players are interchanged now.

The way how Player II chooses the valuation is a little bit more complicated than in the previous case, because Player II can not choose it by increasing the value of the counter (because it that case he would always win by increasing the value infinitely long), so another technique is used. First, Player I must set the value of the counter to "big" enough value — he must choose a value that is multiple of the product of all π_i for $1 \leq i \leq n$, where $\{x_1, x_2, \dots, x_n\}$ is a set of boolean variables in the formula. Player II then chooses the valuation by decreasing this value. The game then continues as in the previous case.

It can be divided into the following phases:

1. Player I must set the value of the counter to some value.
2. It is tested, if Player I set the value to be the multiple of the product of π_i for $1 \leq i \leq n$. If not, Player II wins.
3. Player II chooses the valuation (by decreasing the value of the counter).
4. Player I chooses some clause.
5. Player II chooses some literal in that clause.
6. The value of the chosen literal is tested, Player II wins iff the value of the literal in the given valuation is `true`.

It can be easily checked that Player II has winning strategy iff the formula ϕ is satisfiable — he can choose valuation that makes ϕ true. Then there is in every clause at least one literal with the value `true`. Player I has winning strategy if ϕ is not satisfiable, because in every valuation he can choose a clause where all literals have the value `false`.

The details of the construction of processes are omitted. □

Theorem 4.3 *The problem if $p(0) \preceq_s p'(0)$, where $p(0), p'(0)$ are states of OCN, is* DP*-hard.*

Proof: Let (ϕ_1, ϕ_2) be an instance of Sat-Unsat problem. Using lemma 4.2, we can construct (in polynomial time) OCN processes with start states $q(0), q'(0)$, such that $q(0) \preceq_s q'(0)$ iff ϕ_1 is satisfiable, and using lemma 4.1 we can construct OCN processes with start states $r(0), r'(0)$, such that $r(0) \preceq_s r'(0)$ iff ϕ_2 is not satisfiable.

Now we can construct a disjoint union of these processes with new control states p, p', where we add the transitions:

$$pX \xrightarrow{a} qX \qquad pX \xrightarrow{b} rX \qquad p'X \xrightarrow{a} q'X \qquad p'X \xrightarrow{b} r'X$$

where $X \in \{I, Z\}$, $a, b \in Act$ and $a \neq b$. As can be easily checked, it holds that $p(0) \preceq_s p'(0)$ iff $q(0) \preceq_s q'(0)$ and $r(0) \preceq_s r'(0)$, i.e. iff ϕ_1 is satisfiable and ϕ_2 is not satisfiable. □

Corollary 4.4 *The problem if $q(0) \equiv_s q'(0)$, where $q(0), q'(0)$ are states of OCN, is* DP*-hard.*

Proof: We reduce the problem, if $p(0) \preceq_s p'(0)$, that is DP-hard (theorem 4.3), to the problem, if $q(0) \equiv_s q'(0)$, using the following general construction.

Given states α, β of some transition system Δ, we add the states γ, γ' and the transitions $\gamma \xrightarrow{a} \alpha$, $\gamma \xrightarrow{a} \beta$ and $\gamma' \xrightarrow{a} \beta$ to Δ. As can be easily checked, $\gamma \equiv_s \gamma'$ iff $\alpha \preceq_s \beta$, because $\gamma' \preceq_s \gamma$ is always true, and $\gamma \preceq_s \gamma'$ holds only if $\alpha \preceq_s \beta$. □

4.2 P-hardness of all equivalences for finite-state processes

In the following is proved that deciding equivalence of finite-state processes is P-hard problem for any equivalence between (and including) bisimulation equivalence and trace equivalence.

A problem P is P-hard if any problem in PTIME can be reduced to P by LOGSPACE reduction. A problem P is P-complete if P is P-hard and $P \in$ PTIME.

P-complete problems are considered to be inherently sequential in the sense, that there is not known an efficient parallel algorithm for any P-complete problem. A parallel algorithm is considered to be efficient, if its time complexity is polylogarithmic, i.e. if its time complexity is in $O(\log^k n)$ for some constant k (where n is the size of an instance) when only a polynomial number of processors is used, i.e. when the number of processors is in $O(n^k)$ for some constant k. A class of problems, for which such algorithms exist, is called NC (Nick's class).

It is known that NC $\subseteq$ PTIME, but if NC $\neq$ PTIME (i.e. if NC $\subsetneq$ PTIME) is an open question. It is widely believed that the inclusion is proper, and that there are problems that are in PTIME, but are not in NC. P-complete problems are candidates to be such problems, because it can be shown that if NC $\subsetneq$ PTIME, then no P-complete would be in NC, and if at least one of P-complete problems would be in NC, then NC = PTIME.

The P-completeness of deciding bisimilarity of finite state processes was shown in [2] and equivalences between ready-trace equivalence and trace equivalence and between possible-futures equivalence and trace equivalence are known to be PSPACE-complete [25], so the result obtained here is new only for the remaining equivalences in linear time/branching time spectrum.

To prove P-hardness of deciding equivalence, we describe a reduction from Cvp (circuit value problem — it is described in detail below) to the problem of equivalence of finite-state processes. The Cvp problem is known to be P-complete. Given an instance w of Cvp, we construct two finite-state processes with the start states p_0 and q_0, s.t. p_0 and q_0 are not trace equivalent if w is accepted by an algorithm deciding Cvp, and p_0 and q_0 are bisimilar if w is rejected. It follows that Cvp can be reduced in this way to deciding of any equivalence X s.t. $\sim \subseteq X \subseteq \equiv_{tr}$. (In fact, Cvp is reduced to the problem of non-equivalence, but the difference does not matter.) A similar technique of constructing two processes, that are bisimilar in one case and are not even trace equivalent in the other, was used e.g. in [21].

We start with a few definitions:

Boolean circuit is a directed, acyclic, labelled graph, in which the nodes of indegree zero are the inputs, and all other nodes are of indegree 2 and are labelled by one of $\{\wedge, \vee\}$. One node (with outdegree zero) is the output node. The nodes are called also *gates*. We say that a gate x depends on a gate y, if there is an edge from y to x. A gate labelled with $\wedge$ or $\vee$ computes its value from values on gates it depends on, using boolean function indicated in its label. The values on gates are boolean values, denoted as 0 and 1 in the following.

The CVP problem is defined as follows:

INSTANCE: A boolean circuit and an assignment of boolean values to its input gates.
QUESTION: Is on the output gate the value 1?

We may further assume that the gates $\{x_1, x_2, \ldots, x_n\}$ are ordered in such a way, that x_1 is the output gate and if x_i depends on x_j, then $i < j$. We may assume w.l.o.g. that the output gate in not also an input gate. The problem CVP still remains P-complete under these assumptions.

Let us define the function $t : I \to \{\wedge, \vee, 0, 1\}$, where $I = \{1, 2, \ldots, n\}$, that is used to denote the types of the gates:

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

A pair of (partial) functions $c_1, c_2 : I \to I$ is used to express dependence between gates. If a gate x_i depends on gates x_j and x_k, where $i < j < k$ then $c_1(i) = j$ and $c_2(i) = k$. If x_i is an input gate, then values of $c_1(i), c_2(i)$ are undefined.

The function $v : I \to \{0, 1\}$ is used to denote the actual values on gates, i.e. $v(i)$ is the value on the gate x_i.

We will construct a transition system denoted Δ, and p_0, q_0 will be two different states of Δ. The process with the start state p_0 (resp. q_0) will be called the process p_0 (resp. the process q_0). The set of actions of Δ will be $\{0, 1, h\}$.

The sequences of actions, that could be performed by both processes, will be of the form $a_1 a_2 \ldots a_n$ where each a_i corresponds to the value on the gate x_i. If x_i is an input gate, i.e. if $t(i) = 0$ or $t(i) = 1$, then a_i must be the value on the input gate, i.e. $a_i = t(i)$ if $t(i) = 0$ or $t(i) = 1$. If x_i is the output gate, then a_i must be 1, i.e. $a_1 = 1$. In all other cases a_i may be 0 or 1.

The difference between the processes p_0 and q_0 is that the process p_0 can always perform the action h after that sequence, but the process q_0 can perform the action h after the sequence only if the sequence is "wrong" in the sense that it does not correspond to the actual values on the gates.

The sequence is considered to be *wrong* if it contains some wrong action a_i (where $1 \le i \le n$). The action a_i is *wrong* if $a_i = 1$ and either $t(i) = \wedge$ and at least one of a_j, a_k is 0, where $j = c_1(i)$ and $k = c_2(i)$, or $t(i) = \vee$ and both a_j, a_k are 0.

We call the action a_i *cheating* if $a_i = 1$, but $v(i) = 0$. As can be easily checked, if a_i is cheating, then either a_i is wrong, or at least one of a_j, a_k is cheating. So if a sequence contains a cheating action, then it must be wrong, because $i < j, k$ and the actions that correspond to the input gates can not be cheating.

Notice, that if $v(1) = 1$ (i.e. if there is the value 1 on the output gate), then at least one sequence that is not wrong exists (the one where $a_i = v(i)$ for each $1 \le i \le n$), and if $v(1) = 0$, then every sequence is wrong.

The set of states of the transition system Δ will be the union of the following sets:

- $\{p_i \mid 0 \le i \le n\}$,
- $\{q_i \mid 0 \le i \le n\}$,
- $\{r_{i,j} \mid i \le j < c_2(i)\}$ for each $1 \le i \le n$, such that $t(i) = \vee$,
- $\{r_{i,j} \mid i \le j < c_1(i)\}$ and $\{r'_{i,j} \mid i \le j < c_2(i)\}$ for each $1 \le i \le n$, such that $t(i) = \wedge$.

		$t(j)$			
		$\wedge$	$\vee$	0	1
p_{j-1}		$\xrightarrow{0,1} p_j$	$\xrightarrow{0,1} p_j$	$\xrightarrow{0} p_j$	$\xrightarrow{1} p_j$
q_{j-1}		$\xrightarrow{0,1} q_j$	$\xrightarrow{0,1} q_j$	$\xrightarrow{0} q_j$	$\xrightarrow{1} q_j$
If $t(i) = \wedge$:					
$r_{i,j-1}$	if $i < j < c_1(i)$	$\xrightarrow{0,1} r_{i,j}$	$\xrightarrow{0,1} r_{i,j}$	$\xrightarrow{0} r_{i,j}$	$\xrightarrow{1} r_{i,j}$
	if $j = c_1(i)$	$\xrightarrow{0} p_j$	$\xrightarrow{0} p_j$	$\xrightarrow{0} p_j$	
$r'_{i,j-1}$	if $i < j < c_2(i)$	$\xrightarrow{0,1} r'_{i,j}$	$\xrightarrow{0,1} r'_{i,j}$	$\xrightarrow{0} r'_{i,j}$	$\xrightarrow{1} r'_{i,j}$
	if $j = c_2(i)$	$\xrightarrow{0} p_j$	$\xrightarrow{0} p_j$	$\xrightarrow{0} p_j$	
If $t(i) = \vee$:					
$r_{i,j-1}$	if $i < j < c_1(i)$ or $c_1(i) < j < c_2(i)$	$\xrightarrow{0,1} r_{i,j}$	$\xrightarrow{0,1} r_{i,j}$	$\xrightarrow{0} r_{i,j}$	$\xrightarrow{1} r_{i,j}$
	if $j = c_1(i)$	$\xrightarrow{0} r_{i,j}$	$\xrightarrow{0} r_{i,j}$	$\xrightarrow{0} r_{i,j}$	
	if $j = c_2(i)$	$\xrightarrow{0} p_j$	$\xrightarrow{0} p_j$	$\xrightarrow{0} p_j$	
Common:					
any state from the set $\{p_{j-1}, q_{j-1}, r_{i,j-1}, r'_{i,j-1}\}$		$\xrightarrow{1} q_j$ $\xrightarrow{1} r_{j,j}$ $\xrightarrow{1} r'_{j,j}$	$\xrightarrow{1} q_j$ $\xrightarrow{1} r_{j,j}$		$\xrightarrow{1} q_j$

Table 1: The transitions of the transition system Δ

We say that the state α is on the p-line, if $\alpha = p_i$ for some i, that it is on the q-line, if $\alpha = q_i$ for some i, and that it is on the r-line, if $\alpha = r_{i,j}$ for some i, j. We say that the state α is on the level j if α is p_j, q_j, $r_{i,j}$ or $r'_{i,j}$ for some i.

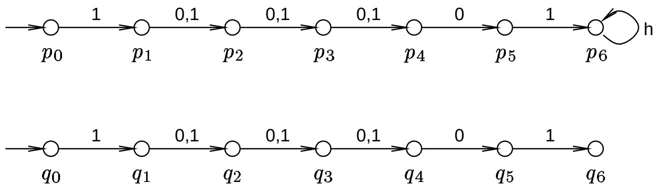
The transitions of Δ are described in the table 1. All transitions are of the form $\alpha_{j-1} \xrightarrow{x} \beta_j$, where $1 \leq j \leq n$, α_{j-1} is a state on the level $j-1$, β_j is a state on the level j and $x \in \{0, 1\}$. The rows in the table correspond to the different possible values of α_{j-1}, the columns correspond to the different values of $t(j)$ and the fields in the table are of the form $\xrightarrow{x} \beta_j$. Some fields may contain more than one transition. The notation $\xrightarrow{0,1} \beta_j$ stands for two transitions ($\xrightarrow{0} \beta_j$ and $\xrightarrow{1} \beta_j$). The last row in the table contains transitions that are possible in any α_{j-1}.

There is one exception to the transitions described in the table: No transitions labelled with 0 are possible in the states p_0 and q_0.

There is one transition not mentioned in the table — the transition $p_n \xrightarrow{h} p_n$.

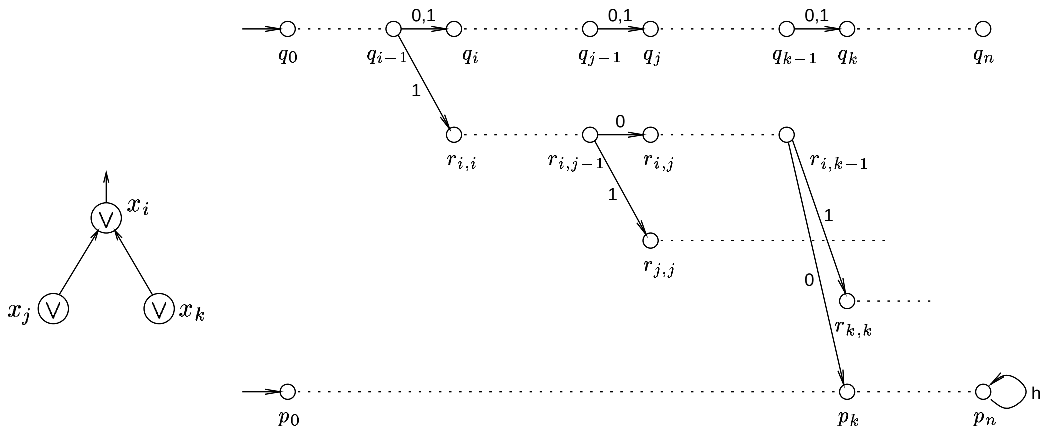
The construction can be described informally as follows:

- We start with the p-line and q-line. An example of this construction is in the figure 3 for a circuit where $n = 6$, x_5 and x_6 are input gates and values on x_5 and x_6 are 0 and 1.
- For every gate x_i, such that $t(i) = \vee$ we add one r-line in a way depicted in the figure 4. This r-line can be used for testing, whether the action a_i is wrong. If it is so, this r-line allows the process q_0 to reach the same state as process p_0.
- For every gate x_i, such that $t(i) = \wedge$ we add two r-lines in a way depicted in the figure 5. These r-lines can be used for testing, whether the action a_i is wrong. If it is so, this r-lines allow the process q_0 to reach the same state as process p_0.
- To states on the p-line and to states on r-lines we add transitions labelled with 1 to all states, to which such transitions from the states on the q-line exist on the same level.



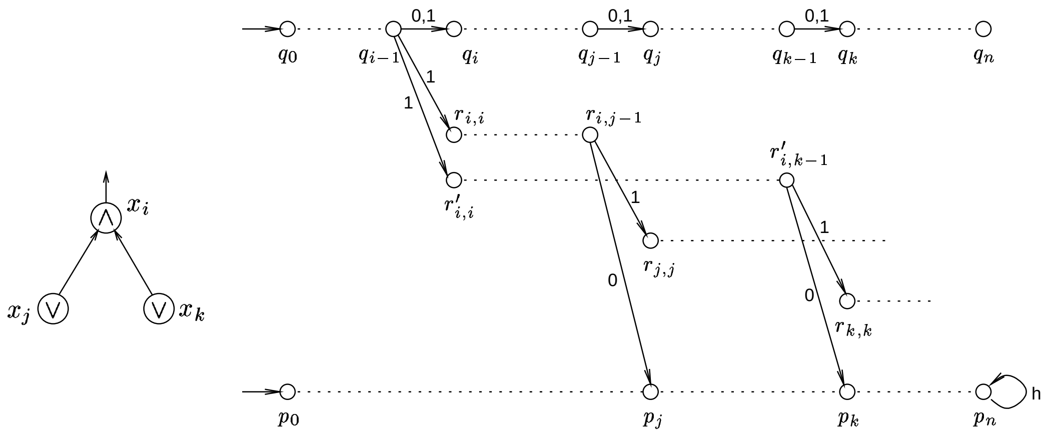
1
0,1
0,1
0,1
0
1
h
p_0 p_1 p_2 p_3 p_4 p_5 p_6
1
0,1
0,1
0,1
0
1
q_0 q_1 q_2 q_3 q_4 q_5 q_6

Figure 3: An example of the start of the construction of the transition system Δ



x_i
x_j
x_k
q_0 q_{i-1} q_i q_{j-1} q_j q_{k-1} q_k q_n
0,1
0,1
0,1
1
0
1
1
0
$r_{i,i}$ $r_{i,j-1}$ $r_{i,j}$ $r_{i,k-1}$ $r_{j,j}$ $r_{k,k}$
h
p_0 p_k p_n

Figure 4: The construction for a gate labelled with $\vee$



x_i
x_j
x_k
q_0 q_{i-1} q_i q_{j-1} q_j q_{k-1} q_k q_n
0,1
0,1
0,1
1
1
$r_{i,i}$ $r_{i,j-1}$ $r'_{i,i}$ $r'_{i,k-1}$ $r_{j,j}$ $r_{k,k}$
1
0
1
0
h
p_0 p_j p_k p_n

Figure 5: The construction for a gate labelled with $\wedge$

Lemma 4.5 *If $v(1) = 1$, then there is a trace from the state p_0, that can not be matched by any trace from the state q_0.*

PROOF: The trace $a_1a_2\dots a_nh$, where $a_i = v(i)$ for each $1 \leq i \leq n$, can be performed by the process p_0. It is clear that the sequence $a_1a_2\dots a_n$ is not wrong.

Let us suppose that this trace can be also performed by the process q_0. Because the h action can be performed only in the state p_n, and because a state on the p-line can be reached from a state on the q-line only through some r-line that ensures that the sequence $a_1a_2\dots a_n$ is wrong, the sequence must be wrong, but this is a contradiction. $\square$

Lemma 4.6 *If $v(1) = 0$, then the states p_0 and q_0 are bisimilar.*

PROOF: We will describe the proof in terms of the bisimulation game. Notice, that in this game are both processes in every move in the states on the same level. They start in states on the level 0, then go to states on the level 1, then on the level 2 and so on. Notice also, that the the set of actions, that can be performed, is the same for all states on the same level j (where $0 \leq j < n$), and this set depends only on $t(j + 1)$.

Let us first suppose that Player I uses only transitions on the p-line, i.e. transitions of the form $p_{j-1} \xrightarrow{a_j} p_j$. The sequence $a_1a_2\dots a_n$ must be wrong, because $a_1 = 1$ is cheating and Player I can not play $a_1 = 0$ (there is no such transition).

The winning strategy for Player II can be described as follows: Whenever Player I uses cheating action a_i ($a_i = 1$ in this case), use transition to the corresponding r-line and then show that either a_i was wrong (and then you can reach the same state as the process p_0) or at least one of a_j, a_k (where $j = c_1(i)$ and $k = c_2(i)$) must be cheating and you can use the same strategy to show it. (If $t(i) = \wedge$ choose between two r-lines the one that shows that a_i is wrong or that must contain a cheating action.) Otherwise, keep on the line where you are.

As can be easily checked, this strategy always allows Player II to show, that there is some wrong action a_i, so he can reach some state on the p-line, and then he simply mimics all moves performed by Player I.

Now we consider the general case, where Player I can perform any transition, and we show that Player I is in fact forced to use transitions on the p-line.

Let us suppose that Player I deviates from playing on the p-line using some transition labelled with a_j. If $a_j = 0$, then due the fact that in every state is possible at most one transition labelled with 0, the players only interchange their roles in this move and the result is exactly the same as if Player I has not deviated from playing on the p-line.

So suppose $a_j = 1$. If Player I uses a transition to some of q_j, $r_{i,j}$ or $r'_{i,j}$ in one of the processes, then the matching transition to the same state exists in the other process, so after the move both processes are in the same state and Player II wins.

The only remaining case is the situation, when Player I uses the transition $r_{i,j-1} \xrightarrow{1} r_{i,j}$ (resp. $r'_{i,j-1} \xrightarrow{1} r'_{i,j}$) in the process q_0. Then Player II plays $p_{j-1} \xrightarrow{1} p_j$ and the situation after the move is again exactly the same, as if Player I has not deviated from playing on the p-line. $\square$

Theorem 4.7 *Deciding equivalence of finite-state processes is* P*-hard problem for any equivalence between bisimulation equivalence and trace equivalence.*

PROOF: The correctness of the described construction follows from lemmas 4.5 and 4.6, so it remains to show that this construction can be achieved in LOGSPACE.

The algorithm performing the construction requires only fixed number of variables to store some indexes, pointers and values. For example to construct all transitions from some state α on the level i, the only information required is i, $j = c_1(i + 1)$, $k = c_2(i + 1)$, $t(i + 1)$, $t(j)$ and $t(k)$ (and also i', $j' = c_1(i')$, $k' = c_2(i')$, $t(i')$, $t(j')$ and $t(k')$, if $\alpha = r_{i',i}$ or $\alpha = r'_{i',i}$).

Let n be the number of gates in the circuit. Then the transition system Δ contains $O(n^2)$ states, and all the variables required during the construction can be stored using only $O(\log n)$ space. $\square$

References

[1] P.A. Abdulla and K. Čerāns. Simulation is decidable for one-counter nets. In *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 253–268. Springer, 1998.

[2] C. Alvarez, J.L. Balcazar, J. Gabarro, and M. Santha. Parallel complexity in the design and analysis of concurrentsystems. In *PARLE91*, volume 505 of *LNCS*. Springer-Verlag, 1991.

[3] E. Bach and J. Shallit. *Algorithmic Number Theory. Vol. 1, Efficient Algoritms.* The MIT Press, 1996.

[4] J. C. M. Baeten and W. P. Weijand. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, Cambridge, England, 1990.

[5] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *Proceedings of the Conference on Parallel Architectures and Languages Europe (PARLE). Volume II: Parallel Languages*, volume 259 of *LNCS*, pages 93–114. Springer-Verlag, 1987.

[6] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM*, 40(3):653–682, July 1993.

[7] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. To appear in *Handbook of Process Algebras.*

[8] O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In *Proceedings of MFCS'95*, volume 969 of *LNCS*, pages 423–433. Springer-Verlag, 1995.

[9] D. Caucal. Graphes canoniques de graphes algébriques. Technical Report RR-0872, Inria, Institut National de Recherche en Informatique et en Automatique, 1988.

[10] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, November 1992.

[11] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for all basic parallel processes. In *CONCUR'93*, volume 715 of *LNCS*, pages 143–157. Springer, 1993.

[12] S. Christensen, Y. Hirshfeld, and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *LICS'93*, pages 386–396. IEEE Computer Society Press, 1993.

[13] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free process. In *CONCUR'92*, volume 630 of *LNCS*, pages 138–147. Springer, 1992.

[14] J.F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):354–371, 1994.

[15] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. Technical Report ECS-LFCS-940286, Department of Computer Science, 1994.

[16] Y. Hishfeld, M. Jerrum, and F. Moller. A polynimial algorithm for deciding bisimulation equivalence of normed basic parallel processes. To appear in *Mathematical Structures in Computer Science*, 1996.

[17] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[18] H. Hüttel. Undecidable equivalences for basic parallel processes. *LNCS*, 789:454, 1994.

[19] H. Hüttel and S. Shukla. On the complexity of deciding behavioural equivalences and preorders. Technical Report SUNYA-CS-96-03, State University of New York at Albany, December 28, 1996.

[20] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings of LICS 91*, pages 376–386. IEEE Computer Society Press, 1991.

[21] P. Jančar. Undecidability of bisimilarity for petri nets and some related problems. *Theoretical Computer Science*, 148:281–301, 1995.

[22] P. Jančar. Bisimulation equivalence is decidable for one-counter processes. In *ICALP'97*, volume 1256 of *LNCS*, pages 549–559, 1997.

[23] P. Jančar and F. Moller. Simulation of one-counter nets via colouring. Technical Report 159, Computing Science Department, Uppsala University, 1999.

[24] P. Jančar, F. Moller, and Z. Sawa. Simulation problems for one-counter machines. In *Proceedings of SOFSEM'99*, LNCS. Springer, 1999.

[25] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[26] A. Kučera. Efficient verification algorithms for one-counter processes. In *Proceedings of ICALP 2000*, volume 1853 of *LNCS*, pages 192–207. Springer, 2000.

[27] R. Mayr. On the complexity of bisimulation problems for basic parallel processes. In *ICALP'2000*, volume 1853 of *LNCS*, Geneva, Switzerland, 2000. Springer Verlag.

[28] R. Mayr. On the complexity of bisimulation problems for pushdown automata. In *IFIP TCS'2000*, volume 1872 of *LNCS*, Sendai, Japan, 2000. Springer Verlag.

[29] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[30] G. Sénizergues. Decidability of bisimulation equivalence of equational graphs of finite out-degree. In *Proc. of FOCS'98.* IEEE, 1998.

[31] R.J. van Glabbeek. The linear time – branching time spectrum. In *Proceedings CONCUR'90*, volume 458 of *LNCS*, pages 278–297, Amsterdam, 1990. Springer-Verlag.

Contents