Equivalence-Checking with One-Counter Automata: A Generic Method for Proving Lower Bounds*

Petr Jančar[1], Antonín Kučera[2], Faron Moller[3], and Zdeněk Sawa[1]

[1] Dept. of Computer Science, FEI, Technical University of Ostrava, 17. listopadu 15, CZ-708 33 Ostrava, Czech Republic {Petr.Jancar,Zdenek.Sawa}@vsb.cz
[2] Faculty of Informatics, Masaryk University, Botanická 68a, CZ-602 00 Brno, Czech Republic tony@fi.muni.cz
[3] Dept. of Computer Science, University of Wales Swansea, Singleton Park, Swansea SA2 8PP, Wales F.G.Moller@swansea.ac.uk

Abstract. We present a general method for proving **DP**-hardness of equivalence-checking problems on one-counter automata. For this we show a reduction of the SAT-UNSAT problem to the truth problem for a fragment of (Presburger) arithmetic. The fragment contains only special formulas with one free variable, and is particularly apt for transforming to simulation-like equivalences on one-counter automata. In this way we show that the membership problem for any relation subsuming bisimilarity and subsumed by simulation preorder is **DP**-hard (even) for one-counter *nets* (where the counter cannot be tested for zero). We also show **DP**-hardness for deciding simulation between one-counter automata and finite-state systems (in both directions).

1 Introduction

In concurrency theory, a *process* is typically defined to be a state in a *transition system*, which is a triple $\mathcal{T} = (S, \Sigma, \rightarrow)$ where S is a set of *states*, Σ is a set of *actions* and $\rightarrow \subseteq S \times \Sigma \times S$ is a *transition relation*. We write $s \xrightarrow{a} t$ instead of $(s, a, t) \in \rightarrow$, and we extend this notation in the natural way to elements of Σ^*. A state t is *reachable* from a state s, written $s \rightarrow^* t$, iff $s \xrightarrow{w} t$ for some $w \in \Sigma^*$.

In this paper, we consider such processes generated by *one-counter automata*, nondeterministic finite-state automata operating on a single counter variable which takes values from the set $\mathbb{N} = \{0, 1, 2, \ldots\}$. Formally this is a tuple $\mathcal{A} = (Q, \Sigma, \delta^=, \delta^>, q_0)$ where Q is a finite set of *control states*, Σ is a finite set of *actions*,

$$\delta^= : Q \times \Sigma \rightarrow \mathcal{P}(Q \times \{0, 1\}) \quad \text{and}$$
$$\delta^> : Q \times \Sigma \rightarrow \mathcal{P}(Q \times \{-1, 0, 1\})$$

are *transition functions* (where $\mathcal{P}(M)$ denotes the power-set of M), and $q_0 \in Q$ is a distinguished *initial* control state. $\delta^=$ represents the transitions which are enabled when the counter value is zero, and $\delta^>$ represents the transitions which are enabled when the counter value is positive. $\mathcal{A}$ is a *one-counter net* if and only if for all pairs $(q, a) \in Q \times \Sigma$ we have that $\delta^=(q, a) \subseteq \delta^>(q, a)$.

* This work was supported by the Grant Agency of the Czech Republic, Grant No. 201/00/0400.

To the one-counter automaton $\mathcal{A}$ we associate the transition system $\mathcal{T}_{\mathcal{A}} = (S, \Sigma, \rightarrow)$, where $S = \{p(n) : p \in Q, n \in \mathbb{N}\}$ and $\rightarrow$ is defined as follows:

$$p(n) \xrightarrow{a} q(n+i) \quad \text{iff} \quad \begin{cases} n = 0, \text{ and } (q,i) \in \delta^{=}(p,a); \;\; \textit{or} \\ n > 0, \text{ and } (q,i) \in \delta^{>}(p,a). \end{cases}$$

Note that any transition increments, decrements, or leaves unchanged the counter value; and a decrementing transition is only possible if the counter value is positive. Also observe that when $n > 0$ the transitions of $p(n)$ do not depend on the actual value of n. Finally note that a one-counter *net* can in a sense test if its counter is nonzero (that is, it can perform some transitions only on the proviso that its counter is nonzero), but it cannot test in any sense if its counter is zero. For ease of presentation, we understand *finite-state* systems (corresponding to transition systems with finitely many states) to be one-counter nets where $\delta^{=} = \delta^{>}$ and the counter is never changed. Thus, the parts of $\mathcal{T}_{\mathcal{A}}$ reachable from $p(i)$ and $p(j)$ are isomorphic and finite for all $p \in Q$ and $i, j \in \mathbb{N}$.

Remark. Let us mention that the class of transition systems generated by one-counter nets is the same (up to isomorphism) as that generated by the class of labelled Petri nets with (at most) one unbounded place. The class of transition systems generated by one-counter automata is the same (up to isomorphism) as that generated by the class of realtime pushdown automata (i.e. pushdown automata without ε-transitions) with a single stack symbol (apart from a special bottom-of-stack marker).

The *equivalence-checking* approach to the formal verification of concurrent systems is based on the following scheme: the specification S (i.e., the intended behaviour) and the actual implementation I of a system are defined as states in transition systems, and then it is shown that S and I are *equivalent*. There are many ways to capture the notion of process equivalence (see, e.g., [15]); however, *simulation* and *bisimulation* equivalence [12, 14] are of special importance, as their accompanying theory has found its way into many practical applications.

Given a transition system $\mathcal{T} = (S, \Sigma, \rightarrow)$, a *simulation* is a binary relation $\mathcal{R} \subseteq S \times S$ satisfying the following property: whenever $(s,t) \in \mathcal{R}$,

$$\text{if } s \xrightarrow{a} s' \text{ then } t \xrightarrow{a} t' \text{ for some } t' \text{ with } (s', t') \in \mathcal{R}.$$

s *is simulated* by t, written $s \sqsubseteq t$, iff $(s,t) \in \mathcal{R}$ for some simulation $\mathcal{R}$; and s and t are *simulation equivalent*, written $s \simeq t$, iff $s \sqsubseteq t$ and $t \sqsubseteq s$. The union of a family of simulation relations is clearly itself a simulation relation; hence, the relation $\sqsubseteq$, being the union of all simulation relations, is in fact the maximal simulation relation, and is referred to as the *simulation preorder*. A characteristic property is that $s \sqsubseteq t$ iff the following holds: if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' with $s' \sqsubseteq t'$.

A *bisimulation* is a symmetric simulation relation, and s and t are *bisimulation equivalent*, or *bisimilar*, written $s \sim t$, if they are related by a bisimulation.

Simulations and bisimulations can also be used to relate states of *different* transition systems; formally, we can consider two transition systems to be a single one by taking the disjoint union of their state sets.

Let A and B be classes of processes. The problem of deciding whether a given process s of A is simulated by a given process t of B is denoted by $A \sqsubseteq B$; similarly, the problem of deciding if s and t are simulation equivalent (or bisimilar) is denoted by $A \simeq B$ (or $A \sim B$, respectively). The classes of all one-counter automata, one-counter nets, and finite-state systems are denoted OCA, OCN, and FS, respectively.

The state of the art: The OCN $\sqsubseteq$ OCN problem was first considered in [1], where it was shown that if two OCN processes are related by *some* simulation, then they are also related by a semilinear simulation (i.e. a simulation definable in Presburger arithmetic), which suffices for semidecidability (and thus decidability) of the positive subcase. (The negative subcase is semidecidable by standard arguments.) A simpler proof was given later in [7] by employing certain "geometric" techniques, which allow you to conclude that the simulation preorder (over a given one-counter net) is itself semilinear. Moreover, it was shown there that the OCA $\sqsubseteq$ OCA problem is undecidable. The decidability of the OCA $\sim$ OCA problem was demonstrated in [4] by showing that the greatest bisimulation relation over states of a given one-counter automaton is also semilinear. The relationship between simulation and bisimulation problems for processes of one-counter automata has been studied in [6] where it was shown that one can effectively reduce certain simulation problems to their bisimulation counterparts by applying a technique proposed in [10]. The complexity of bisimilarity-checking with one-counter automata was studied in [8], where the problem OCN $\sim$ OCN (as well as the problem of *weak* bisimilarity [12] between OCN and FS processes) was shown to be **DP**-hard; however, the problem OCA $\sim$ FS was shown to be solvable in polynomial time. Complexity bounds for simulation-checking were given in [9], where it was shown that the problems OCN $\sqsubseteq$ FS and FS $\sqsubseteq$ OCN (and thus also OCN $\simeq$ FS) are in **P**, while OCA $\sqsubseteq$ FS and OCA $\simeq$ FS are **coNP**-hard.

Our contribution: In this paper we generalize the techniques used in [8, 9] for establishing lower complexity bounds for certain equivalence-checking problems, and present a general method for showing **DP**-hardness of problems for one-counter automata. (The class **DP** [13] consists of those languages which are expressible as a difference of two languages from **NP**, and is generally believed to be larger than the union of **NP** and **coNP**. Section 2.2 contains further comments on **DP**.) The "generic part" of the method is presented in Section 2 , where we define a simple fragment of Presburger arithmetic, denoted OCP, which is

- sufficiently powerful so that satisfiability and unsatisfiability of boolean formulas are both polynomially reducible to the problem of deciding the truth of formulas of OCP, which implies that this latter problem is **DP**-hard (Theorem 2); yet
- sufficiently simple so that the problem of deciding the truth of OCP formulas is polynomially reducible to various equivalence-checking problems (thus providing the "application part" of the proposed method). The reduction is typically constructed inductively on the structure of OCP formulas, thus making the proofs readable and easily verified.

In Section 3.1 we apply the method to the OCN $\leftrightarrow$ OCN problem where $\leftrightarrow$ is any relation which subsumes bisimilarity and is subsumed by simulation preorder (thus, besides bisimilarity and simulation equivalence also, e.g., ready simulation equivalence or 2-nested simulation equivalence), showing **DP**-hardness of these problems (Theorem 5). In Section 3.2 we concentrate on simulation problems between one-counter and finite-state automata, and prove that OCA $\sqsubseteq$ FS, FS $\sqsubseteq$ OCA, and OCA $\simeq$ FS are all **DP**-hard (Theorem 7), thus improving on the bounds presented in [8, 9]. Finally, in Section 4 we draw some conclusions and present a detailed summary of known results.

2 The OCP Fragment of Arithmetic

In this section, we introduce a fragment of (Presburger) arithmetic, denoted OCP (which can be read as "One-Counter Properties"). We then show how to encode the problems of satisfiability and unsatisfiability of boolean formulas in OCP, and thus deduce **DP**-hardness of the truth problem for (closed formulas of) OCP. (The name of the language is motivated by a relationship to one-counter automata which will be explored in the next section.)

2.1 Definition of OCP

OCP can be viewed as a certain set of first-order arithmetic formulas. We shall briefly give the syntax of these formulas; the semantics will be obvious. Since we only consider the interpretation of OCP formulas in the standard structure of natural numbers $\mathbb{N}$, the problem of deciding the truth of a closed OCP formula is well defined:

Problem: TRUTHOCP
INSTANCE: A closed formula $Q \in$ OCP.
QUESTION: Is Q true ?

Let x and y range over (first-order) *variables*. A formula $Q \in$ OCP can have at most one free variable x (i.e., outside the scope of quantifiers); we shall write $Q(x)$ to indicate the free variable (if there is one) of Q; that is, $Q(x)$ either has the one free variable x, or no free variables at all. For a number $k \in \mathbb{N}$, $\lceil k \rceil$ stands for a special term denoting k; we can think of $\lceil k \rceil$ as $SS \ldots S0$, i.e., the successor function S applied k times to 0. We stipulate that $size(\lceil k \rceil) = k{+}1$ (which corresponds to representing numbers in unary).

The formulas Q of OCP are defined inductively as follows; at the same time we inductively define their size (keeping in mind the unary representation of $\lceil k \rceil$):

(a) $x = 0$	$size(Q) = 1$
(b) $\lceil k \rceil \mid x$ ("k divides x"; $k{>}0$)	$size(Q) = k{+}1$
(c) $\lceil k \rceil \nmid x$ ("k does not divide x"; $k{>}0$)	$size(Q) = k{+}1$
(d) $Q_1(x) \wedge Q_2(x)$	$size(Q) = size(Q_1) + size(Q_2) + 1$
(e) $Q_1(x) \vee Q_2(x)$	$size(Q) = size(Q_1) + size(Q_2) + 1$
(f) $\exists y \leq x : Q'(y)$ (x and y distinct)	$size(Q) = size(Q') + 1$
(g) $\forall x : Q'(x)$	$size(Q) = size(Q') + 1$

We shall need to consider the truth value of a formula $Q(x)$ in a valuation assigning a number $n \in \mathbb{N}$ to the (possibly) free variable x; this is given by the formula $Q[n/x]$ obtained by replacing each free occurrence of the variable x in Q by n. Slightly abusing notation, we shall denote this by $Q(n)$. (Symbols like i, j, k, n range over natural numbers, not variables.) For example, if $Q(x)$ is the formula $\exists y \leq x : ((3 \mid y) \wedge (2 \nmid y))$, then $Q(5)$ is true while $Q(2)$ is false; and if $Q(x)$ is a closed formula, then the truth value of $Q(n)$ is independent of n.

2.2 DP-hardness of TRUTHOCP

Recall the following problem:

> **Problem**: SAT-UNSAT
> INSTANCE: A pair (φ, ψ) of boolean formulas in conjunctive normal form (CNF).
> QUESTION: Is it the case that φ is satisfiable and ψ is unsatisfiable ?

This problem is **DP**-complete, which corresponds to an intermediate level in the polynomial hierarchy, harder than both Σ_1^p and Π_1^p but still contained in Σ_2^p and Π_2^P (cf., e.g., [13]). Our aim here is to show that SAT-UNSAT is polynomial-time reducible to TRUTHOCP. In particular, we show how, given a boolean formula φ in CNF, we can in polynomial time construct a (closed) formula of OCP which claims that φ is satisfiable, and also a formula of OCP which claims that φ is unsatisfiable (Theorem 2).

First we introduce some notation. Let $Var(\varphi) = \{x_1, \ldots, x_m\}$ denote the set of (boolean) variables in φ. Further let π_j (for $j \geq 1$) denote the j^{th} prime number. For every $n \in \mathbb{N}$ define the assignment $\nu_n : Var(\varphi) \to \{true, false\}$ by

$$\nu_n(x_j) = \begin{cases} true, & \text{if } \pi_j \mid n, \\ false, & \text{otherwise.} \end{cases}$$

Note that for an arbitrary assignment ν there is $n \in \mathbb{N}$ such that $\nu_n = \nu$; it suffices to take $n = \Pi\{\, \pi_j : 1 \leq j \leq m \text{ and } \nu(x_j) = true \,\}$. By $\|\varphi\|_\nu$ we denote the truth value of φ under the assignment ν.

Lemma 1. *There is a polynomial-time algorithm which, given a boolean formula φ in CNF, constructs* OCP*-formulas $Q_\varphi(x)$ and $\overline{Q}_\varphi(x)$ such that both size(Q_φ) and size$(\overline{Q}_\varphi)$ are in $\mathcal{O}(|\varphi|^3)$, and such that for every $n \in \mathbb{N}$*

$$Q_\varphi(n) \text{ is true} \quad \text{iff} \quad \overline{Q}_\varphi(n) \text{ is false} \quad \text{iff} \quad \|\varphi\|_{\nu_n} = true.$$

Proof. Let $Var(\varphi) = \{x_1, \ldots, x_m\}$. Given a literal ℓ (that is, a variable x_j or its negation $\overline{x}_j$), define the OCP-formula $Q_\ell(x)$ as follows:

$$Q_{x_j}(x) \;=\; \lceil \pi_j \rceil \mid x \quad \text{and} \quad Q_{\overline{x}_j}(x) \;=\; \lceil \pi_j \rceil \nmid x.$$

Clearly, $Q_\ell(n)$ is true iff $Q_{\overline{\ell}}(n)$ is false iff $\|\ell\|_{\nu_n} = true$.

- Formula $Q_\varphi(x)$ is obtained from φ by replacing each literal ℓ with $Q_\ell(x)$. It is clear that $Q_\varphi(n)$ is true iff $\|\varphi\|_{\nu_n} = true$.
- Formula $\overline{Q}_\varphi(x)$ is obtained from φ by replacing each $\wedge$, $\vee$, and ℓ with $\vee$, $\wedge$, and $Q_{\overline{\ell}}(x)$, respectively. It is readily seen that $\overline{Q}_\varphi(n)$ is true iff $\|\varphi\|_{\nu_n} = false$.

It remains to evaluate the size of Q_φ and $\overline{Q}_\varphi$. Here we use a well-known fact from number theory (cf, e.g., [2]) which says that π_m is in $\mathcal{O}(m^2)$. Hence $size(Q_\ell)$ is in $\mathcal{O}(|\varphi|^2)$ for every literal ℓ of φ. As there are $\mathcal{O}(|\varphi|)$ literal occurrences and $\mathcal{O}(|\varphi|)$ boolean connectives in φ, we can see that $size(Q_\varphi)$ and $size(\overline{Q}_\varphi)$ are indeed in $\mathcal{O}(|\varphi|^3)$. □

We now come to the main result of the section.

Theorem 2. *Problem* SAT-UNSAT *is reducible in polynomial time to* TRUTHOCP. *Therefore,* TRUTHOCP *is **DP**-hard.*

Proof. We give a polynomial-time algorithm which, given an instance (φ, ψ) of SAT-UNSAT, constructs a closed OCP-formula Q, with $size(Q)$ in $\mathcal{O}(|\varphi|^3 + |\psi|^3)$, such that Q is true iff φ is satisfiable and ψ is unsatisfiable.

Expressing the unsatisfiability of ψ is straightforward: by Lemma 1, ψ is unsatisfiable iff the OCP-formula

$$\forall x : \overline{Q}_\psi(x)$$

is true. Thus, let Q_2 be this formula.

Expressing the satisfiability of φ is rather more involved. Let $g = \pi_1\pi_2 \ldots \pi_m$, where $Var(\varphi) = \{x_1, \ldots, x_m\}$. Clearly φ is satisfiable iff there is some $n \leq g$ such that $||\varphi||_{\nu_n} = true$. Hence φ is satisfiable iff

the OCP-formula $\exists y \leq x : Q_\varphi(y)$ is true for any valuation assigning $i \geq g$ to x.

As it stands, it is unclear how this might be expressed; however, we can observe that the equivalence still holds if we replace the condition "$i \geq g$" with "i is a multiple of g". In other words, φ is satisfiable iff for every $i \in \mathbb{N}$ we have that either $i = 0$, or $g \nmid i$, or there is some $n \leq i$ such that $Q_\varphi(n)$ is true. This can be written as

$$\forall x \ : \ x = 0 \ \vee \ (\lceil \pi_1 \rceil \nmid x \ \vee \cdots \vee \ \lceil \pi_m \rceil \nmid x) \ \vee \ \exists y \leq x : Q_\varphi(y)$$

We thus let Q_1 be this formula.

Hence, (φ, ψ) is a positive instance of the SAT-UNSAT problem iff the formula

$$Q = Q_1 \wedge Q_2$$

is true. To finish the proof, we observe that $size(Q)$ is indeed in $\mathcal{O}(|\varphi|^3 + |\psi|^3)$. □

2.3 TRUTHOCP is in Π_2^p

The conclusions we draw for our verification problems are that they are **DP**-hard, as we reduce the **DP**-hard problem TRUTHOCP to them. We cannot improve this lower bound by much using the reduction from TRUTHOCP, as TRUTHOCP is in Π_2^p. In this section we sketch the proof of this fact.

Proposition 3. TRUTHOCP *is in* Π_2^p.

Proof. We start by first proving that for every formula $Q(x)$ of OCP there is a d with $0 < d \leq 2^{size(Q)}$ such that $Q(i) = Q(i-d)$ for every $i > 2^{size(Q)}$. Hence, $\forall x : Q(x)$ holds iff $\forall x \leq 2^{size(Q)} : Q(x)$ holds. (Note that $\forall x \leq 2^{size(Q)} : Q(x)$ is not a formula of OCP.)

We prove the existence of d for every formula $Q(x)$ by induction on the structure of $Q(x)$. If $Q(x)$ is $x = 0$ then we can take $d = 1$; and if $Q(x)$ is $\lceil k \rceil \mid x$ or $\lceil k \rceil \nmid x$ then we can take $d = k$.

If $Q(x)$ is $Q_1(x) \wedge Q_2(x)$ or $Q_1(x) \vee Q_2(x)$, then we may assume by the induction hypothesis the existence of the relevant d_1 for Q_1 and d_2 for Q_2. We can then take $d = d_1 d_2$ to give the desired property that $Q(i) = Q(i-d)$ for every $i > 2^{size(Q)}$.

If $Q(x)$ is $\exists y \leq x : Q'(y)$ (x and y distinct) then by the induction hypothesis there is a d' with $0 < d' \leq 2^{size(Q')}$ such that $Q'(i) = Q'(i-d')$ for every $i > 2^{size(Q')}$. It follows that if $Q'(i)$ is true for some i, then it is true for some $i \leq 2^{size(Q')} < 2^{size(Q)}$. Furthermore, if $Q'(i)$ is true for some i then $Q(j)$ is true for every $j \geq i$; on the other hand, if $Q'(i)$ is false for every i, then $Q(j)$ is false for every j. Thus we can take $d = 1$.

If $Q(x)$ is $\forall y : Q'(y)$, then x is not free in $Q'(y)$, so the truth value of $Q(i)$ does not depend on i and we can take $d = 1$.

Next we note that every OCP-formula $Q(x)$ can be transformed into a formula $\widehat{Q}(x)$ (which need not be in OCP) in (pseudo-)prenex form

$$
\begin{aligned}
&(\forall x_1 \leq 2^{size(Q_1)}) \cdots (\forall x_k \leq 2^{size(Q_k)}) \\
&\qquad\qquad (\exists y_1 \leq z_1) \cdots (\exists y_\ell \leq z_\ell)\, \mathcal{F}(x_1, \ldots, x_k, y_1, \ldots, y_\ell)
\end{aligned}
$$

where

- $\forall x_i : Q_i(x_i)$ is a subformula of $Q(x)$;
- each $z_i \in \{x_1, \ldots, x_k,\, y_1, \ldots, y_{i-1}\}$; and
- $\mathcal{F}(x_1, \ldots, x_k,\, y_1, \ldots, y_\ell)$ is a $\wedge, \vee$-combination of atomic subformulas of $Q(x)$.

This can be proved by induction on the structure of $Q(x)$. The only case requiring some care is the case when $Q(x)$ is of the form $\exists y \leq x : Q'(y)$, because $\exists y \forall z : P(y, z)$ and $\forall z \exists y : P(y, z)$ are not equivalent in general, but they are in our case, as z never depends on y due to restrictions in OCP. Note that the size of $\widehat{Q}(x)$ is polynomial in $size(Q)$ (assuming that $2^{size(Q_1)}, \ldots, 2^{size(Q_k)}$ are encoded in binary).

We can construct an alternating Turing machine which first uses its universal states to assign all possible values (bounded as mentioned above) to $x_1, \ldots, x_k$, then uses its existential states to assign all possible values to $y_1, \ldots, y_\ell$, and finally evaluates (deterministically) the formula $\mathcal{F}(x_1, \ldots, x_k,\, y_1, \ldots, y_\ell)$. It is clear that this alternating Turing machine can be constructed so that it works in time which is polynomial in $size(Q)$. This implies the membership of TRUTHOCP in Π_2^p. □

3 Application to One-Counter Automata Problems

As we mentioned above, the language OCP was designed with one-counter automata in mind. The problem TRUTHOCP can be relatively smoothly reduced to various verification problems for such automata, by providing relevant constructions ("implementations") for the cases (a)-(g) of the OCP definition, and thus it constitutes a useful tool for proving lower complexity bounds (**DP**-hardness) for these problems. We shall demonstrate this for the OCN $\leftrightarrow$ OCN problem, where $\leftrightarrow$ is any relation satisfying that $\sim\ \subseteq\ \leftrightarrow\ \subseteq\ \sqsubseteq$, and then also for the OCA $\sqsubseteq$ FS, FS $\sqsubseteq$ OCA, and OCA $\simeq$ FS problems.

For the purposes of our proofs, we adopt a "graphical" representation of one-counter automata as finite graphs with two kinds of edges (solid and dashed ones) which are labelled by pairs of the form $(a,i) \in \Sigma \times \{-1,0,1\}$; instead of $(a,-1)$, $(a,1)$, and $(a,0)$ we write simply $-a$, $+a$, and a, respectively. A *solid* edge from p to q labelled by (a,i) indicates that the represented one-counter automaton can make a transition $p(k) \xrightarrow{a} q(k+i)$ whenever $i \geq 0$ or $k > 0$. A *dashed* edge from p to q labelled by (a,i) (where i must not be -1) represents a zero-transition $p(0) \xrightarrow{a} q(i)$. Hence, graphs representing one-counter nets do not contain any dashed edges, and graphs corresponding to finite-state systems use only labels of the form $(a,0)$ (remember that finite-state systems are formally understood as special one-counter nets). Also observe that the graphs cannot represent non-decrementing transitions which are enabled *only* for positive counter values; this does not matter since we do not need such transitions in our proofs. The distinguished initial control state(s) is (are) indicated by a black circle.

3.1 Results for One-Counter Nets

In this section we show that, for any relation $\leftrightarrow$ satisfying $\sim\ \subseteq\ \leftrightarrow\ \subseteq\ \sqsubseteq$, the problem of deciding whether two (states of) one-counter nets are in $\leftrightarrow$ is **DP**-hard. We first state an important technical result, but defer its proof until after we derive the desired theorem as a corollary.

Proposition 4. *There is an algorithm which, given a formula $Q = Q(x) \in$ OCP as input, halts after $\mathcal{O}(size(Q))$ steps and outputs a one-counter net with two distinguished control states p and p' such that for every $k \in \mathbb{N}$ we have:*

– *if* $Q(k)$ *is true then* $p(k) \sim p'(k)$*;*
– *if* $Q(k)$ *is false then* $p(k) \not\sqsubseteq p'(k)$*.*

(Note that if Q is a closed formula, then this implies that $p(0) \sim p'(0)$ if Q is true, and $p(0) \not\sqsubseteq p'(0)$ if Q is false.)

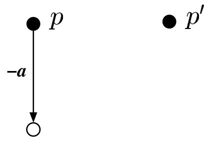
Theorem 5. *For any relation $\leftrightarrow$ such that $\sim\ \subseteq\ \leftrightarrow\ \subseteq\ \sqsubseteq$, the following problem is* ***DP****-hard:*

INSTANCE*: A one-counter net with two distinguished control states p and p'.*
QUESTION*: Is $p(0) \leftrightarrow p'(0)$?*

Proof. Given an instance of TRUTHOCP, i.e., a *closed* formula $Q \in$ OCP, we use the (polynomial) algorithm of Proposition 4 to construct a one-counter net with the two distinguished control states p and p'. If Q is true, then $p(0) \sim p'(0)$, and hence $p(0) \leftrightarrow p'(0)$; and if Q is false, then $p(0) \not\sqsubseteq p'(0)$, and hence $p(0) \not\leftrightarrow p'(0)$. $\square$

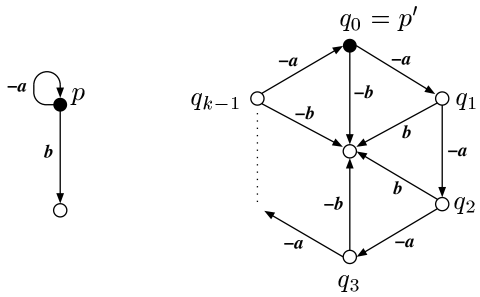
Proof of Proposition 4: We proceed by induction on the structure of Q. For each case, we show an *implementation*, i.e., the corresponding one-counter net $\mathcal{N}_Q$ with two distinguished control states p and p'. Constructions are sketched by figures which use our notational conventions; the distinguished control states are denoted by black dots (the left one p, the right one p'). It is worth noting that we only use two actions, a and b.

(a) $Q(x) = (x = 0)$: A suitable (and easily verifiable) implementation looks as follows:



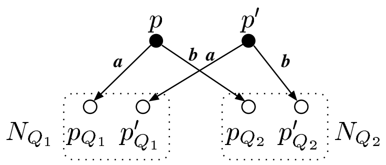
(b,c) $Q(x) = \lceil k \rceil \mid x$ or $Q(x) = \lceil k \rceil \nmid x$, where $k{>}0$: Given $J \subseteq \{0, 1, 2, \ldots, k{-}1\}$, let $R_J(x) = (x \bmod k) \in J$. We shall show that this formula can be implemented in our sense; taking $J = \{0\}$ then gives us the construction for case (b), and taking $J = \{1, \ldots, k{-}1\}$ gives us the construction for case (c).

An implementation of $R_J(x)$, where $1, 2 \in J$ but $0, 3, k{-}1 \notin J$, looks as follows:



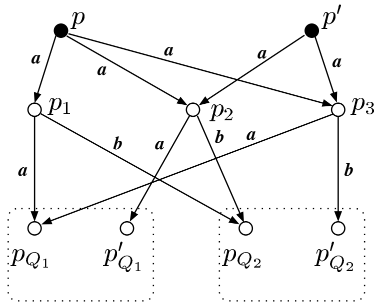
In this picture, each node q_i has an outgoing edge going to a "dead" state; this edge is labelled b if $i \in J$ and labelled $-b$ if $i \notin J$. It is straightforward to check that the proposed implementation of $R_J(x)$ is indeed correct.

(d) $Q(x) = Q_1(x) \wedge Q_2(x)$: We can assume (by induction) that implementations $\mathcal{N}_{Q_1}$ of $Q_1(x)$ and $\mathcal{N}_{Q_2}$ of $Q_2(x)$ have been constructed. $\mathcal{N}_Q$ is constructed, using $\mathcal{N}_{Q_1}$ and $\mathcal{N}_{Q_2}$, as follows:



The dotted rectangles represent the graphs associated to $\mathcal{N}_{Q_1}$ and $\mathcal{N}_{Q_2}$ (where the only depicted control states are the distinguished ones). Verifying the correctness of this construction is straightforward.

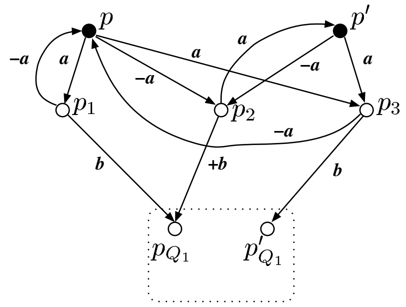
(e) $Q(x) = Q_1(x) \vee Q_2(x)$: As in case (d), the construction uses the implementations of $Q_1(x)$ and $Q_2(x)$; but the situation is slightly more involved in this case:



To verify correctness, we first consider the case when $Q(k)$ is true. By induction, either $p_{Q_1}(k) \sim p'_{Q_1}(k)$ or $p_{Q_2}(k) \sim p'_{Q_2}(k)$. In the first case, $p_{Q_1}(k) \sim p'_{Q_1}(k)$ implies that $p_1(k) \sim p_2(k)$, which in turn implies that $p(k) \sim p'(k)$; similarly, in the second case, $p_{Q_2}(k) \sim p'_{Q_2}(k)$ implies that $p_1(k) \sim p_3(k)$, which also implies that $p(k) \sim p'(k)$. Hence in either case $p(k) \sim p'(k)$.

Now consider the case when $Q(k)$ is false. By induction, $p_{Q_1}(k) \not\sqsubseteq p'_{Q_1}(k)$ and $p_{Q_2}(k) \not\sqsubseteq p'_{Q_2}(k)$. Obviously, $p_{Q_1}(k) \not\sqsubseteq p'_{Q_1}(k)$ implies that $p_1(k) \not\sqsubseteq p_2(k)$, and $p_{Q_2}(k) \not\sqsubseteq p'_{Q_2}(k)$ implies that $p_1(k) \not\sqsubseteq p_3(k)$. From this we have $p(k) \not\sqsubseteq p'(k)$.

(f) $Q(x) = \exists y \leq x : Q_1(y)$ (where x, y are distinct): We use the following construction:



To verify correctness, we first consider the case when $Q(k)$ is true. This means that $Q_1(i)$ is true for some $i \leq k$, which by induction implies that $p_{Q_1}(i) \sim p'_{Q_1}(i)$ for this $i \leq k$. Our result, that $p(k) \sim p'(k)$, follows immediately from the following:

Claim: For all k, if $p_{Q_1}(i) \sim p'_{Q_1}(i)$ for some $i \leq k$, then $p(k) \sim p'(k)$.

Proof of Claim. By induction on k. For the base case (k=0), if $p_{Q_1}(i) \sim p'_{Q_1}(i)$ for some $i \leq 0$, then $p_{Q_1}(0) \sim p'_{Q_1}(0)$, which implies that $p_1(0) \sim p_3(0)$, and hence that $p(0) \sim p'(0)$. For the induction step (k>0), if $p_{Q_1}(i) \sim p'_{Q_1}(i)$ for some $i \leq k$, then either $p_{Q_1}(k) \sim p'_{Q_1}(k)$, which implies that $p_1(k) \sim p_3(k)$ which in turn implies that $p(k) \sim p'(k)$; or $p_{Q_1}(i) \sim p'_{Q_1}(i)$ for some

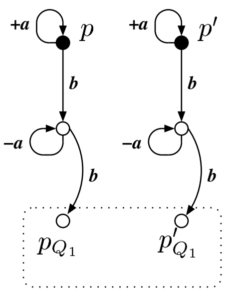
$i{\le}k{-}1$, which by induction implies that $p(k{-}1) \sim p'(k{-}1)$, which implies that $p_1(k) \sim p_2(k{-}1)$, which in turn implies that $p(k) \sim p'(k)$.

Next, we consider that case when $Q(k)$ is false. This means that $Q_1(i)$ is false for all $i{\le}k$, which by induction implies that $p_{Q_1}(i) \not\sqsubseteq p'_{Q_1}(i)$ for all $i{\le}k$. Our result, that $p(k) \not\sqsubseteq p'(k)$, follows immediately from the following:

Claim: For all k, if $p(k) \sqsubseteq p'(k)$ then $p_{Q_1}(i) \sqsubseteq p'_{Q_1}(i)$ for some $i{\le}k$.

Proof of Claim. By induction on k. For the base case ($k{=}0$), if $p(0) \sqsubseteq p'(0)$ then $p_1(0) \sqsubseteq p_3(0)$, which in turn implies that $p_{Q_1}(0) \sqsubseteq p'_{Q_1}(0)$. For the induction step ($k{>}0$), if $p(k) \sqsubseteq p'(k)$ then either $p_1(k) \sqsubseteq p_2(k{-}1)$ or $p_1(k) \sqsubseteq p_3(k)$. In the first case, $p_1(k) \sqsubseteq p_2(k{-}1)$ implies that $p(k{-}1) \sqsubseteq p'(k{-}1)$, which by induction implies that $p_{Q_1}(i) \sqsubseteq p'_{Q_1}(i)$ for some $i{\le}k{-}1$ and hence for some $i{\le}k$; and in the second case, $p_1(k) \sqsubseteq p_3(k)$ implies that $p_{Q_1}(k) \sqsubseteq p'_{Q_1}(k)$.

(g) $Q = \forall x : Q_1(x)$: The implementation in the following figure can be easily verified.



For any $Q \in$ OCP, the described construction terminates after $\mathcal{O}(size(Q))$ steps, because we add only a constant number of new nodes in each subcase except for (b) and (c), where we add $\mathcal{O}(k)$ new nodes (recall that the size of $\lceil k \rceil$ is $k + 1$). □

3.2 Simulation Problems for One-Counter Automata and Finite-State Systems

Now we establish **DP**-hardness of the OCA $\sqsubseteq$ FS, FS $\sqsubseteq$ OCA, and OCA $\simeq$ FS problems. Again, we use the (inductively defined) reduction from TRUTHOCP; only the particular constructions are now slightly different.

By an *implementation* we now mean a 4-tuple $(\mathcal{A}, \mathcal{F}, \mathcal{F}', \mathcal{A}')$ where $\mathcal{A}, \mathcal{A}'$ are one-counter automata, and $\mathcal{F}, \mathcal{F}'$ are finite-state systems; the role of distinguished states is now played by the initial states, denoted q for $\mathcal{A}$, f for F, f' for F', and q' for $\mathcal{A}'$. We again first state an important technical result, and again defer its proof until after we derive the desired theorem as a corollary.

Proposition 6. *There is an algorithm which, given $Q = Q(x) \in$ OCP as input, halts after $\mathcal{O}(size(Q))$ steps and outputs an implementation $(\mathcal{A}, \mathcal{F}, \mathcal{F}', \mathcal{A}')$ (where q, f, f' and q' are the initial control states of $\mathcal{A}$, $\mathcal{F}$, $\mathcal{F}'$ and $\mathcal{A}'$, respectively) such that for every $k \in \mathbb{N}$ we have:*

$Q(k)$ is true iff $q(k) \sqsubseteq f$ iff $f' \sqsubseteq q'(k)$.

(Note that if Q is a closed formula, then this implies that Q is true iff $q(0) \sqsubseteq f$ iff $f' \sqsubseteq q'(0)$.)

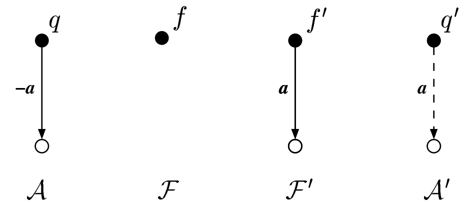
Theorem 7. *Problems* OCA $\sqsubseteq$ FS, FS $\sqsubseteq$ OCA, *and* OCA $\simeq$ FS *are* ***DP****-hard.*

Proof. Recalling that TRUTHOCP is **DP**-hard, **DP**-hardness of the first two problems readily follows from Proposition 6.

DP-hardness of the third problem follows from a simple (general) reduction of OCA $\sqsubseteq$ FS to OCA $\simeq$ FS: given a one-counter automaton $\mathcal{A}$ with initial state q, and a finite-state system $\mathcal{F}$ with initial state f, we first transform $\mathcal{F}$ to $\mathcal{F}_1$ by adding a new state f_1 and transition $f_1 \stackrel{a}{\to} f$, and then create $\mathcal{A}_1$ by taking (disjoint) union of $\mathcal{A}$, $\mathcal{F}_1$ and adding $\overline{f_1} \stackrel{a}{\to} q$, where $\overline{f_1}$ is the copy of f_1 in $\mathcal{A}_1$. Clearly $q(k) \sqsubseteq f$ iff $\overline{f_1}(k) \simeq f_1$. $\square$

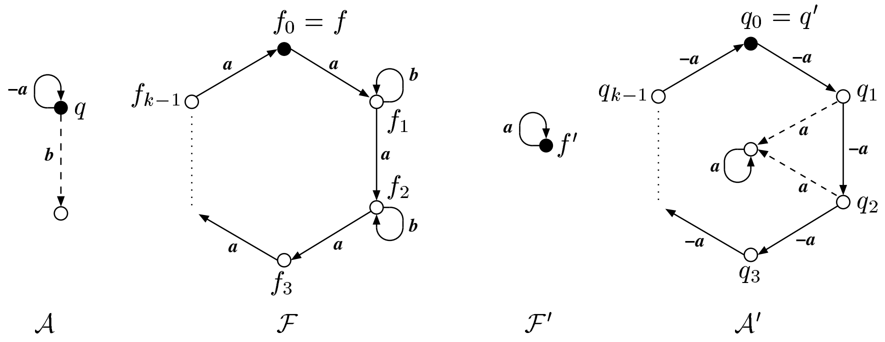
Proof of Proposition 6: We proceed by induction on the structure of Q. In the constructions we use only two actions, a and b; this also means that a state with non-decreasing a and b loops is *universal*, i.e, it can simulate "everything".

(a) $Q = (x = 0)$: A straightforward implementation looks as follows:



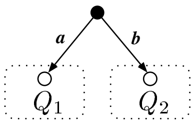
(b,c) $Q = \lceil k \rceil \,|\, x$ or $Q = \lceil k \rceil \nmid x$, where $k>0$: Given $J \subseteq \{0, 1, 2, \ldots, k-1\}$, let $R_J(x) = (x \bmod k) \in J$. We shall show that this formula can be implemented in our sense; taking $J = \{0\}$ then gives us the construction for case (b), and taking $J = \{1, \ldots, k-1\}$ gives us the construction for case (c).

An implementation of $R_J(x)$, where $1, 2 \in J$ but $0, 3, k-1 \notin J$, looks as follows:



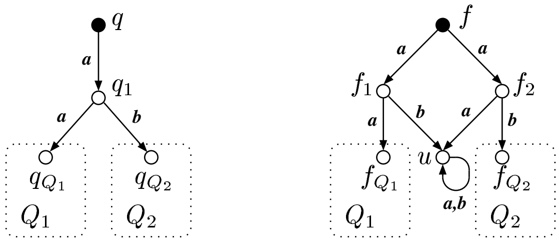
In this picture, node f_i has a b-loop in $\mathcal{F}$, and node q_i has an outgoing dashed a-edge in $\mathcal{A}'$, iff $i \in J$. It is straightforward to check that the proposed implementation of $R_J(x)$ is indeed correct.

(d) $Q(x) = Q_1(x) \wedge Q_2(x)$: The members of the implementation $(\mathcal{A}_Q, \mathcal{F}_Q, \mathcal{F}'_Q, \mathcal{A}'_Q)$ for Q can be constructed from the respective members of the implementations for Q_1, Q_2 (assumed by induction): $\mathcal{A}_Q$ from $\mathcal{A}_{Q_1}$ and $\mathcal{A}_{Q_2}$; $\mathcal{F}_Q$ from $\mathcal{F}_{Q_1}$ and $\mathcal{F}_{Q_2}$; $\mathcal{F}'_Q$ from $\mathcal{F}'_{Q_1}$ and $\mathcal{F}'_{Q_2}$; and $\mathcal{A}'_Q$ from $\mathcal{A}'_{Q_1}$ and $\mathcal{A}'_{Q_2}$. All these cases follow the schema depicted in the following figure:



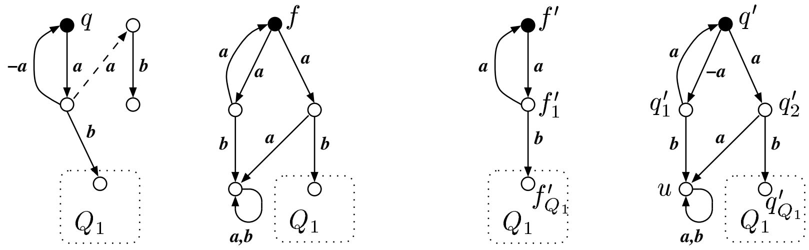
Correctness is easily verifiable.

(e) $Q(x) = Q_1(x) \vee Q_2(x)$: We give constructions just for $\mathcal{A}$ and $\mathcal{F}$ (the constructions for $\mathcal{F}'$ and $\mathcal{A}'$ are almost identical):



For any k, $Q(k)$ is true iff $Q_1(k)$ is true or $Q_2(k)$ is true, which by induction is true iff $q_{Q_1}(k) \sqsubseteq f_{Q_1}$ or $q_{Q_2}(k) \sqsubseteq f_{Q_2}$, which is true iff $q_1(k) \sqsubseteq f_1$ or $q_1(k) \sqsubseteq f_2$, which in turn is true iff $q(k) \sqsubseteq f$.

(f) $Q(x) = \exists y \leq x : Q_1(y)$ (where x, y are distinct): We use the following constructions:



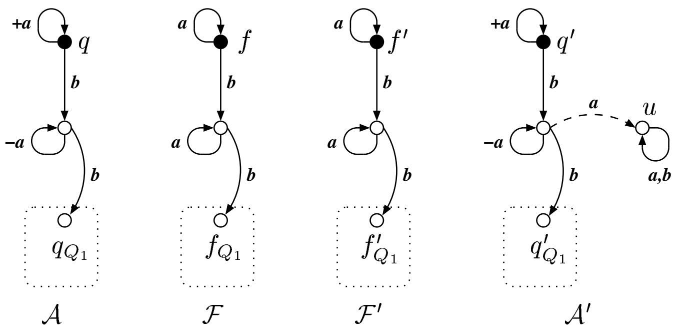
$\mathcal{A}$ $\mathcal{F}$ $\mathcal{F}'$ $\mathcal{A}'$

We prove that the construction is correct for $\mathcal{F}'$ and $\mathcal{A}'$ (the other case being similar). $Q(k)$ is true iff $Q_1(i)$ is true for some $i \leq k$, which by induction is true iff $f'_{Q_1} \sqsubseteq q'_{Q_1}(i)$ for some $i{\leq}k$, which in turn is true iff $f'_1 \sqsubseteq q'_2(i)$ for some $i{\leq}k$. Our result, that this is true iff $f' \sqsubseteq q'(k)$, follows immediately from the following:

Claim: For all k, $f' \sqsubseteq q'(k)$ iff $f'_1 \sqsubseteq q'_2(i)$ for some $i{\leq}k$.

Proof of Claim. By induction on k. For the base case (k=0), the result is immediate. For the induction step (k>0), first note that $f'_1 \sqsubseteq q'_1(k{-}1)$ iff $f' \sqsubseteq q'(k{-}1)$, which by induction is true iff $f'_1 \sqsubseteq q'_2(i)$ for some $i{\leq}k{-}1$. Thus $f' \sqsubseteq q'(k)$ iff $f'_1 \sqsubseteq q'_2(k)$ or $f'_1 \sqsubseteq q'_1(k{-}1)$, which is true iff $f'_1 \sqsubseteq q'_2(k)$ or $f'_1 \sqsubseteq q'_2(i)$ for some $i{\leq}k{-}1$, which in turn is true iff $f'_1 \sqsubseteq q'_2(i)$ for some $i{\leq}k$.

(g) $Q = \forall x : Q_1(x)$: It is easy to show the correctness of the implementation in the following figure.



For any $Q \in$ OCP, the described construction terminates after $\mathcal{O}(size(Q))$ steps, because we add only a constant number of new nodes in each subcase except for (b) and (c), where we add $\mathcal{O}(k)$ new nodes. □

4 Conclusions

Intuitively, the reason why we could not lift the **DP** lower bound to some higher complexity class (e.g., **PSPACE**) is that there is no apparent way to implement a "step-wise guessing" of assignments which would allow us to encode, e.g., the QBF problem. The difficulty is that if we modify the counter value, we were not able to find a way to check that the old and new values encode "compatible" assignments which agree on a certain subset of propositional constants. Each such attempt resulted in an exponential blow-up in the number of control states.

A summary of known results about equivalence-checking with one-counter automata is given below (where $\approx$ denotes weak bisimilarity).

- OCN $\approx$ OCN and OCA $\approx$ OCA remain open.
- OCA $\sqsubseteq$ OCA and OCA $\simeq$ OCA are undecidable.
- OCA $\sim$ OCA, OCN $\sim$ OCN, OCN $\sqsubseteq$ OCN and OCN $\simeq$ OCN are decidable and **DP**-hard, but without any known upper bound.

– OCA $\approx$ FS, OCN $\approx$ FS, OCA $\sqsubseteq$ FS, FS $\sqsubseteq$ OCA and OCA $\simeq$ FS are decidable, **DP**-hard, and in **EXPTIME**. The **EXPTIME** upper bound is due to the fact that all of the mentioned problems can be easily reduced to the model-checking problem with pushdown systems (see, e.g., [5, 10, 9]) and the modal μ-calculus which is **EXPTIME**-complete [16].
– OCA $\sim$ FS, OCN $\sim$ FS, OCN $\sqsubseteq$ FS, FS $\sqsubseteq$ OCN and OCN $\simeq$ FS are in **P**.

To complete the picture, let us also mention that the model-checking problem with a fixed formula $\Diamond[a]\Diamond[b]\mathtt{false}$ of a simple branching-time logic EF (which can be seen as a natural fragment of CTL [3]) is **NP**-hard for OCN processes, which also means that model-checking with $\Box\langle a\rangle\Box\langle b\rangle\mathtt{true}$ (which is the negation of the above given formula) is **coNP**-hard [9]. From this one can readily see that model-checking with $[c]\Diamond[a]\Diamond[b]\mathtt{false} \wedge \langle d\rangle\Box\langle a\rangle\Box\langle b\rangle\mathtt{true}$ is in fact **DP**-hard for OCN processes. It is quite interesting that model checking with Hennessy-Milner logic [12] is still polynomial even for OCA processes (this problem is **PSPACE**-hard for related models like BPA or BPP [11]).

References

[1] P. 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] E. Bach and J. Shallit. *Algorithmic Number Theory. Vol. 1, Efficient Algorithms*. The MIT Press, 1996.
[3] E. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, B, 1991.
[4] P. Jančar. Decidability of bisimilarity for one-counter processes. *Information and Computation*, 158(1):1–17, 2000.
[5] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theoretical Computer Science*, 258(1–2):409–433, 2001.
[6] P. Jančar, A. Kučera, and F. Moller. Simulation and bisimulation over one-counter processes. In *Proceedings of STACS 2000*, volume 1770 of *LNCS*, pages 334–345. Springer, 2000.
[7] P. Jančar, F. Moller, and Z. Sawa. Simulation problems for one-counter machines. In *Proceedings of SOFSEM'99*, volume 1725 of *LNCS*, pages 404–413. Springer, 1999.
[8] A. Kučera. Efficient verification algorithms for one-counter processes. In *Proceedings of ICALP 2000*, volume 1853 of *LNCS*, pages 317–328. Springer, 2000.
[9] A. Kučera. On simulation-checking with sequential systems. In *Proceedings of ASIAN 2000*, volume 1961 of *LNCS*, pages 133–148. Springer, 2000.
[10] A. Kučera and R. Mayr. Simulation preorder on simple process algebras. In *Proceedings of ICALP'99*, volume 1644 of *LNCS*, pages 503–512. Springer, 1999.
[11] R. Mayr. Strict lower bounds for model checking BPA. *Electronic Notes in Theoretical Computer Science*, 18, 1998.
[12] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
[13] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
[14] D. Park. Concurrency and automata on infinite sequences. In *Proceedings* 5^{th} *GI Conference*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.
[15] R. van Glabeek. The linear time - branching time spectrum I. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.
[16] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.