Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Information and Computation 188 (2004) 1–19

Information and Computation

www.elsevier.com/locate/ic

DP lower bounds for equivalence-checking and model-checking of one-counter automata☆

Petr Jančar,[a,1] Antonín Kučera,[b,1] Faron Moller,[c,*] and Zdeněk Sawa[a,1]

[a] *Department of Computer Science, FEI, Technical University of Ostrava, 17. listopadu 15, Ostrava CZ-70833, Czech Republic*
[b] *Faculty of Informatics, Masaryk University, Botanická 68a, Brno CZ-60200, Czech Republic*
[c] *Department of Computer Science, University of Wales Swansea, Singleton Park, Swansea, Wales SA2 8PP, UK*

Abstract

We present a general method for proving **DP**-hardness of problems related to formal verification of 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), and for the model-checking problem with one-counter nets and the branching-time temporal logic EF.

Keywords: One-counter machines; Equivalence-checking; Model-checking

1. Introduction

In concurrency theory, a ***process*** is typically defined to be a state in a ***transition system***, which is a triple $T = (S, \Sigma, \rightarrow)$ 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 $w \in \Sigma^*$.

☆ This paper is based on results which previously appeared in [11,15].

* Corresponding author. Fax: +44-1792-295-708.
E-mail addresses: Petr.Jancar@vsb.cz (P. Jančar), tony@fi.muni.cz (A. Kučera), F.G.Moller@swansea.ac.uk (F. Moller), Zdenek.Sawa@vsb.cz (Z. Sawa).

[1] Supported by the Grant Agency of the Czech Republic, Grant No. 201/03/1161.

0890-5401/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/S0890-5401(03)00171-8

The study of transition systems has a long and illustrious history in the guise of automata theory. A great many classes of automata have been studied extensively, particularly in terms of the languages which they describe. However, automata have found greater importance recently as process generators rather than as language generators; they are now more often used to describe the behaviour of computing systems rather than for describing the syntactic structure of languages.

Still, the standard classes of automata are finding their place in the study of system behaviours. For example, context-free grammars form the basis of the Basic Process Algebra BPA of Bergstra and Klop [3] as well as the Basic Parallel Process algebra BPP of Christensen [5]; these are both well-studied sublanguages of the full Process Algebra PA [3]. Although most analyses in practice are carried out on finite state system models, these wider classes of automata have found various applications. In particular, BPA and pushdown automata (state-extended BPA) have been used for dataflow analysis of recursive procedures, with particular applications to optimizing compilers [7]. This study has recently been extended to one-counter BPA [4].

In this paper we consider 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 $A = (Q, \Sigma, \delta^=, \delta^>, q_0)$ where Q is a finite set of ***control states***, Σ is a finite set of ***actions***,

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

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. A is a ***one-counter net*** if and only if for all pairs $(q, a) \in Q \times \Sigma$ we have that $\delta^=(q, a) \subseteq \delta^>(q, a)$.

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

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

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 immediate 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 T_A reachable from $p(i)$ and $p(j)$ are isomorphic and finite for all $p \in Q$ and $i, j \in \mathbb{N}$.

Remark 1. 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., [23]); however, *simulation* and *bisimulation* equivalence [19,21] are of special importance, as their accompanying theory has found its way into many practical applications.

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

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

s is ***simulated*** by t, written $s \sqsubseteq t$, iff $(s, t) \in R$ for some simulation 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 $\mathcal{P}$ and $\mathcal{Q}$ be classes of processes. The problem of deciding whether a given process s of $\mathcal{P}$ is simulated by a given process t of $\mathcal{Q}$ is denoted by $\mathcal{P} \sqsubseteq \mathcal{Q}$; similarly, the problem of deciding if s and t are simulation equivalent (or bisimilar) is denoted by $\mathcal{P} \simeq \mathcal{Q}$ (or $\mathcal{P} \sim \mathcal{Q}$, respectively). The classes of all one-counter automata, one-counter nets, and finite-state systems are denoted $\mathcal{A}$, $\mathcal{N}$, and $\mathcal{F}$, respectively.

In the *model-checking* approach to formal verification, one defines the desired properties of the implementation as a formula in a suitable temporal logic, and then it is shown that the implementation satisfies the formula. There are many temporal logics which can be classified according to various aspects (see, e.g., [6,22] for an overview). The simplest (branching-time and action-based) temporal logic is ***Hennessy–Milner logic*** (HML) [19]. The syntax is given by

$$\Phi ::= \mathit{true} \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \langle a \rangle \Phi$$

Here a ranges over a countable alphabet of actions. Given a transition system $T = (S, \Sigma, \rightarrow)$ and an HML formula Φ, we inductively define the ***denotation*** of Φ, denoted $[\![\Phi]\!]$, which is the set of all states of T where the formula *holds*:

$$\begin{aligned} [\![\mathit{true}]\!] &= S \\ [\![\Phi_1 \wedge \Phi_2]\!] &= [\![\Phi_1]\!] \cap [\![\Phi_2]\!] \\ [\![\neg\Phi]\!] &= S - [\![\Phi]\!] \\ [\![\langle a \rangle \Phi]\!] &= \{s \in S \mid \exists t \in S : s \xrightarrow{a} t \wedge t \in [\![\Phi]\!]\} \end{aligned}$$

As usual, we write $s \models \Phi$ instead of $s \in [\![\Phi]\!]$. The operator dual to $\langle a \rangle$ is $[a]$ defined by $[a]\Phi \equiv \neg\langle a \rangle \neg\Phi$. The other propositional connectives are introduced in the standard way.

The logic EF is obtained by extending HML with the $\Diamond$ (reachability) operator. Its semantics is defined as follows:

$$[\![\Diamond\Phi]\!] \;=\; \{s \in S \mid \exists t \in S : s \to^* t \;\wedge\; t \in [\![\Phi]\!]\}$$

The formula $\Diamond\Phi$ can be phrased "there **E**xists a **F**uture state such that Φ holds"; this justifies the "EF" acronym. The dual operator to $\Diamond$ is $\Box$, defined by $\Box\Phi \equiv \neg\Diamond\neg\Phi$. The logic EF can also be seen as a natural fragment of CTL [6].

The state of the art. The $\mathcal{N} \sqsubseteq \mathcal{N}$ problem was first considered in [1], where it was shown that if two one-counter net 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 of the positive subcase, and hence decidability (the negative subcase being semidecidable by standard arguments.) A simpler proof was given later in [12] 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 $\mathcal{A} \sqsubseteq \mathcal{A}$ problem is undecidable. The decidability of the $\mathcal{A} \sim \mathcal{A}$ problem was demonstrated in [8] by showing that the greatest bisimulation relation over the 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 [10] where it was shown that one can effectively reduce certain simulation problems to their bisimulation counterparts by applying a technique proposed in [16]. The complexity of bisimilarity-checking with one-counter automata was studied in [14], where the problem $\mathcal{N} \sim \mathcal{N}$ is shown to be **coNP**-hard and the problem of *weak* bisimilarity [19] between $\mathcal{N}$ and $\mathcal{F}$ processes even **DP**-hard; moreover, the problem $\mathcal{A} \sim \mathcal{F}$ was shown to be solvable in polynomial time. Complexity bounds for simulation-checking were given in [15], where it was shown that the problems $\mathcal{N} \sqsubseteq \mathcal{F}$ and $\mathcal{F} \sqsubseteq \mathcal{N}$ (and thus also $\mathcal{N} \simeq \mathcal{F}$) are in **P**, while $\mathcal{A} \sqsubseteq \mathcal{F}$ and $\mathcal{A} \simeq \mathcal{F}$ are **coNP**-hard (and solvable in exponential time). As for model-checking, we can transfer upper complexity bounds from the results which were achieved for *pushdown processes*, because $\mathcal{A}$ can be seen as a (proper) subclass of pushdown automata (cf. Remark 1). Hence, model-checking with logics like EF, CTL, CTL* [6], or even the modal μ-calculus [13], is decidable in exponential time for one-counter automata processes [24]. However, the techniques for lower complexity bounds do not carry over to $\mathcal{A}$. Another simple observation is that model-checking for HML and $\mathcal{A}$ processes is in **P**. This is because the (in)validity of a given HML formula Φ in a state s depends only on those states which are reachable from s along a path consisting of at most d transitions, where d is the nesting depth of the $\langle a\rangle$ operator in Φ. Since the number of states which are reachable from a given one-counter automata process $p(i)$ is clearly polynomial in d and the size of the underlying one-counter automaton, we can easily design a polynomial time model-checking algorithm. (It contrasts with other models like BPA or BPP where model-checking HML is **PSPACE**-complete [17].)

Our contribution. We generalize the technique used in [14] for establishing lower complexity bounds for certain equivalence-checking problems, and present a general method for showing **DP**-hardness of equivalence-checking and model-checking problems for one-counter automata. (The class **DP** [20] consists of those languages which are expressible as a difference of two languages from **NP**, and is generally conjectured 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 OCL ("One-Counter Logic") 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 OCL, which implies that this latter problem is **DP**-hard (Theorem 3); yet

- sufficiently simple so that the problem of deciding the truth of OCL formulas is polynomially reducible to various equivalence-checking and model-checking problems (thus providing the "application part" of the proposed method). The reduction is typically constructed inductively on the structure of OCL formulas, thus making the proofs readable and easily verified.

In Section 3.1 we apply the method to the $\mathcal{N} \leftrightarrow \mathcal{N}$ 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 6). In particular, we improve the **coNP** lower bound for the $\mathcal{N} \sim \mathcal{N}$ problem established in [14]. In Section 3.2 we concentrate on simulation problems between one-counter and finite-state automata, and prove that $\mathcal{A} \sqsubseteq \mathcal{F}$, $\mathcal{F} \sqsubseteq \mathcal{A}$, and $\mathcal{A} \simeq \mathcal{F}$ are all **DP**-hard (Theorem 8). Section 3.3 is devoted to the complexity of model-checking with one-counter processes. As already mentioned, the model-checking problem for HML and one-counter automata processes is in **P**. We show that model-checking with the logic EF is already intractable: it is **DP**-hard even for processes of one-counter nets and a *fixed* EF formula (Theorem 11). In practice, temporal formulas are usually quite small; hence, the fact that the EF formula can be fixed provides stronger evidence of computational intractability. Finally, in Section 4 we draw some conclusions and present a detailed summary of known results.

2. The OCL fragment of arithmetic

In this section, we introduce a fragment of (Presburger) arithmetic, denoted OCL ("One-Counter Logic"). We then show how to encode the problems of satisfiability and unsatisfiability of boolean formulas in OCL, and thus deduce **DP**-hardness of the truth problem for (closed formulas of) OCL. (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 OCL

OCL 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 OCL formulas in the standard structure of natural numbers $\mathbb{N}$, the problem of deciding the truth of a closed OCL formula is well defined:

Problem. TRUTHOCL

INSTANCE: A closed formula $Q \in$ OCL.

QUESTION: Is Q true ?

Let x and y range over (first-order) *variables*. A formula $Q \in$ OCL 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 $k \in \mathbb{N}$, we let $\lceil k \rceil$ stand for a special term denoting k which we can think of as a unary representation of k. In this way, we require that the size of the representation $\lceil k \rceil$ of a number k be on the order of k rather than $\ln k$.

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

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

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 $\exists y \leqslant x : ((3 \mid y) \wedge (2 \nmid y))$, 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 TRUTHOCL*

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 while 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., [20]). Our aim here is to show that SAT–UNSAT is polynomial-time reducible to TRUTHOCL . In particular, we show how, given a boolean formula φ in CNF, we can in polynomial time construct a (closed) formula of OCL which claims that φ is satisfiable, and also a formula of OCL which claims that φ is unsatisfiable (Theorem 3).

First we introduce some notation. Let $Var(\varphi) = \{x_1, \ldots, x_m\}$ denote the set of (boolean) variables in φ. Furthermore, let π_j (for $j \geqslant 1$) denote the jth 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 exists an $n \in \mathbb{N}$ such that $\nu_n = \nu$; it suffices to take $n = \Pi\{\pi_j : 1 \leqslant j \leqslant m \text{ and } \nu(x_j) = true\}$. By $\|\varphi\|_\nu$ we denote the truth value of φ under the assignment ν.

Lemma 2. *There is a polynomial-time algorithm which, given a boolean formula φ in CNF, constructs OCL-formulas $Q_\varphi(x)$ and $\overline{Q}_\varphi(x)$ such that both $size(Q_\varphi)$ and $size(\overline{Q}_\varphi)$ are in $\mathcal{O}(|\varphi|^3)$, and such that for every $n \in \mathbb{N}$*

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

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 OCL-formula $Q_\ell(x)$ as follows:

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

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

- 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 $\|\varphi\|_{v_n} = \mathit{true}$.
- 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 $\|\varphi\|_{v_n} = \mathit{false}$.

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 $\mathit{size}(Q_\ell)$ 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 $\mathit{size}(Q_\varphi)$ and $\mathit{size}(\overline{Q}_\varphi)$ are indeed in $\mathcal{O}(|\varphi|^3)$. □

We now come to the main result of the section.

Theorem 3. *Problem* SAT-UNSAT *is reducible in polynomial time to* TRUTHOCL. *Therefore*, TRUTHOCL *is* ***DP****-hard*.

Proof. We give a polynomial-time algorithm which, given an instance (φ, ψ) of SAT–UNSAT, constructs a closed OCL-formula Q, with $\mathit{size}(Q)$ 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 2, ψ is unsatisfiable iff the OCL-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 $g = \pi_1\pi_2 \cdots \pi_m$, where $Var(\varphi) = \{x_1, \ldots, x_m\}$. Clearly, φ is satisfiable iff there is some $n \leqslant g$ such that $\|\varphi\|_{v_n} = \mathit{true}$. Hence φ is satisfiable iff the OCL-formula $\exists y \leqslant x : Q_\varphi(y)$ is true for any valuation assigning some $i \geqslant g$ 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 "$i \geqslant g$" with "i is a positive 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 $n \leqslant i$ such that $Q_\varphi(n)$ is true. This can be written as

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

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 $\mathit{size}(Q)$ is indeed in $\mathcal{O}(|\varphi|^3 + |\psi|^3)$. □

2.3. TRUTHOCL 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 TRUTHOCL to them. We cannot improve this lower bound by much using the reduction from TRUTHOCL , as TRUTHOCL is in Π_2^p. In this section we sketch the ideas of a proof of this fact.

Theorem 4. TRUTHOCL *is in* Π_2^p.

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

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 $\exists y \leqslant x : Q'(y)$ (x and y distinct) then by the induction hypothesis there is a d' with $0 < d' \leqslant 2^{size(Q')}$ 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 $i \leqslant 2^{size(Q')} < 2^{size(Q)}$ (recall that $size(Q) = size(Q') + 1$). Furthermore, if $Q'(i)$ is true for some i then $Q(j)$ is true for every $j \geqslant i$; 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 OCL-formula $Q(x)$ can be transformed into a formula $\widehat{Q}(x)$ (which need not be in OCL) in (pseudo-)prenex form

$$(\forall x_1 \leqslant 2^{size(Q_1)}) \cdots (\forall x_k \leqslant 2^{size(Q_k)})$$
$$(\exists y_1 \leqslant z_1) \cdots (\exists y_\ell \leqslant z_\ell) \mathcal{F}(x_1, \ldots, x_k, y_1, \ldots, y_\ell)$$

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 $\exists y \leqslant x : Q'(y)$, 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 OCL. 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 TRUTHOCL in Π_2^p. □

3. Application to one-counter automata problems

As we mentioned above, the language OCL was designed with one-counter automata in mind. The problem TRUTHOCL can be relatively smoothly reduced to various verification problems for such automata, by providing relevant constructions ("implementations") for the various cases (a)-(g) of the OCL definition, and thus it constitutes a useful tool for proving lower complexity bounds (**DP**-hardness) for these problems. We shall demonstrate this for the $\mathcal{N} \leftrightarrow \mathcal{N}$ problem, where $\leftrightarrow$ is any relation satisfying that $\sim \subseteq \leftrightarrow \subseteq \sqsubseteq$, and then also for the $\mathcal{A} \sqsubseteq \mathcal{F}$, $\mathcal{F} \sqsubseteq \mathcal{A}$, and $\mathcal{A} \simeq \mathcal{F}$ problems, and finally for the problem of model checking for the logic EF over $\mathcal{N}$ processes.

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 $p(n) \xrightarrow{a} q(n + i)$ whenever $i \geqslant 0$ or $n > 0$. 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 states are indicated by black circles.

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 5. *There is an algorithm which, given a formula $Q = Q(x) \in OCL$ 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 $n \in \mathbb{N}$ we have*:

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

(*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 6. *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 TRUTHOCL , i.e., a *closed* formula $Q \in$ OCL, we use the (polynomial-time) algorithm of Proposition 5 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 $p(0) \not\leftrightarrow p'(0)$. □

Proof of Proposition 5. We proceed by induction on the structure of Q. For each case, we construct an *implementation*, i.e., a corresponding one-counter net N_Q with two distinguished control states p and p'. In each case we demonstrate that the two bi-implications

$$p(n) \sqsubseteq p'(n) \iff Q(n) \iff p(n) \sim p'(n)$$

hold for each $n \in \mathbb{N}$. (We are only required to prove implications for Proposition 5; however, the stronger bi-implications arise with no added difficulty.)

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, in our constructions.

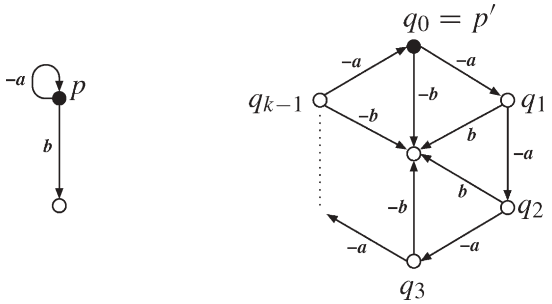
(a) $Q(x) = (x = 0)$: The following provides a suitable construction:

That this construction suffices is readily verified:

$$p(n) \sqsubseteq p'(n) \iff n = 0 \iff p(n) \sim p'(n).$$

(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 the formula $R_J(x)$ has an associated implementation 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).

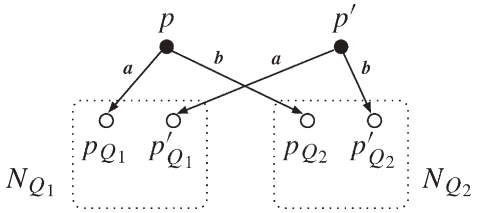
An implementation for $R_J(x)$, where for the point of illustration we have $1, 2 \in J$ but $0, 3, k-1 \notin J$, looks as follows:



In this picture, each node q_i has an outgoing edge leading 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 for $R_J(x)$ is indeed correct:

$$p(n) \sqsubseteq p'(n) \iff (n \bmod k) \in J \iff p(n) \sim p'(n).$$

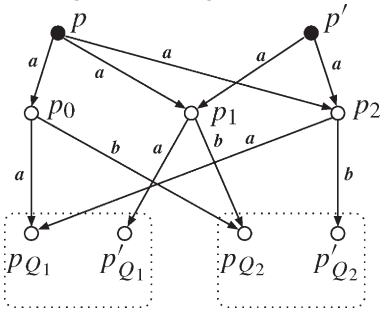
(d) $Q(x) = Q_1(x) \wedge Q_2(x)$: We can assume by induction that implementations N_{Q_1} and N_{Q_2} for $Q_1(x)$ and $Q_2(x)$, respectively, have been constructed. N_Q is then constructed from N_{Q_1} and N_{Q_2} as follows:



The dotted rectangles represent the graphs associated to N_{Q_1} and N_{Q_2} (only the distinguished control states are depicted). Verifying the correctness of this construction is straightforward:

$$\begin{aligned} p(n) \sqsubseteq p'(n) &\iff p_{Q_1}(n) \sqsubseteq p'_{Q_1}(n) \wedge p_{Q_2}(n) \sqsubseteq p'_{Q_2}(n) \\ &\overset{\text{induction}}{\iff} Q_1(n) \wedge Q_2(n) \;=\; Q(n) \\ &\overset{\text{induction}}{\iff} p_{Q_1}(n) \sim p'_{Q_1}(n) \wedge p_{Q_2}(n) \sim p'_{Q_2}(n) \\ &\iff p(n) \sim p'(n) \end{aligned}$$

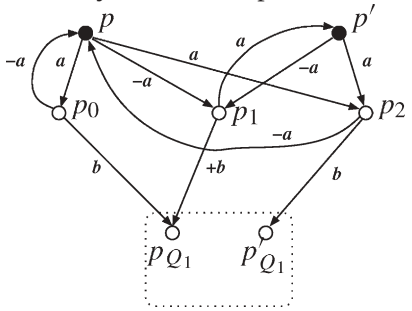
(e) $Q(x) = Q_1(x) \vee Q_2(x)$: As in case (d), the construction uses the inductively assumed implementations for $Q_1(x)$ and $Q_2(x)$; but the situation is slightly more involved in this case:



In this construction, p and p' are identical apart from the transition $p \xrightarrow{a} p_0$. Thus, to show either $p(n) \sqsubseteq p'(n)$ or $p(n) \sim p'(n)$ it is necessary and sufficient to show that the transition $p(n) \xrightarrow{a} p_0(n)$ can be matched either by $p'(n) \xrightarrow{a} p_1(n)$ (which in turn is true iff the transition $p_0(n) \xrightarrow{a} p_{Q_1}(n)$ can be matched by the transition $p_1(n) \xrightarrow{a} p'_{Q_1}(n)$), or by $p'(n) \xrightarrow{a} p_2(n)$ (which in turn is true iff the transition $p_0(n) \xrightarrow{b} p_{Q_2}(n)$ can be matched by the transition $p_2(n) \xrightarrow{b} p'_{Q_2}(n)$). If $Q_1(n)$ is true then the transition $p'(n) \xrightarrow{a} p_1(n)$ works, and if $Q_2(n)$ is true then the transition $p'(n) \xrightarrow{a} p_2(n)$ works; if neither is true (that is, $Q(n)$ is false) then neither transition works.
This reasoning underlies the following argument.

$$\begin{aligned} p(n) \sqsubseteq p'(n) &\iff p_0(n) \sqsubseteq p_1(n) \vee p_0(n) \sqsubseteq p_2(n) \\ &\iff p_{Q_1}(n) \sqsubseteq p'_{Q_1}(n) \vee p_{Q_2}(n) \sqsubseteq p'_{Q_2}(n) \\ &\overset{\text{induction}}{\iff} Q_1(n) \vee Q_2(n) = Q(n) \\ &\overset{\text{induction}}{\iff} p_{Q_1}(n) \sim p'_{Q_1}(n) \vee p_{Q_2}(n) \sim p'_{Q_2}(n) \\ &\iff p_0(n) \sim p_1(n) \vee p_0(n) \sim p_2(n) \\ &\iff p(n) \sim p'(n) \end{aligned}$$

(f) $Q(x) = \exists y \leqslant x : Q_1(y)$ (where x, y are distinct): We use the following construction involving the inductively assumed implementation for $Q_1(x)$:

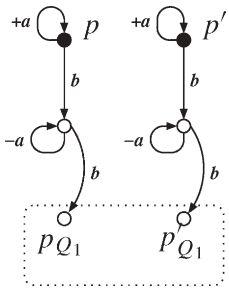


In this construction, p and p' are identical apart from the transition $p \xrightarrow{a} p_0$. Thus, to show either $p(n) \sqsubseteq p'(n)$ or $p(n) \sim p'(n)$ it is necessary and sufficient to show that the transition $p(n) \xrightarrow{a} p_0(n)$ can be matched either by $p'(n) \xrightarrow{a} p_1(n-1)$ (which in turn is true iff the transition $p_0(n) \xrightarrow{a} p(n-1)$ can be matched by the transition $p_1(n-1) \xrightarrow{a} p'(n-1)$), or by $p'(n) \xrightarrow{a} p_2(n)$ (which in turn is true iff the transition $p_0(n) \xrightarrow{b} p_{Q_1}(n)$ can be matched by the transition $p_2(n) \xrightarrow{b} p'_{Q_1}(n)$). If $Q_1(n)$ is true then the transition $p'(n) \xrightarrow{a} p_2(n)$ works, and if $Q_1(i)$ is true for some $i < n$ then the transition $p'(n) \xrightarrow{a} p_1(n-1)$ works; if neither is true (that is, $Q(n)$ is false) then neither transition works.

This reasoning underlies the following argument, which is carried out by a further induction on $n \in \mathbb{N}$; that is, in the case where $n > 0$ we assume that $p(n-1) \sqsubseteq p'(n-1) \iff Q(n-1) \iff p(n-1) \sim p'(n-1)$.

$$\begin{aligned}
p(n) \sqsubseteq p'(n) &\iff \Big(n{>}0 \ \wedge\ p_0(n) \sqsubseteq p_1(n{-}1)\Big) \vee p_0(n) \sqsubseteq p_2(n) \\
&\iff \Big(n{>}0 \ \wedge\ p(n{-}1) \sqsubseteq p'(n{-}1)\Big) \vee p_{Q_1}(n) \sqsubseteq p'_{Q_1}(n) \\
&\overset{\text{induction}}{\iff} \Big(n{>}0 \ \wedge\ \exists y \leqslant n{-}1 : Q_1(y)\Big) \vee Q_1(n) \ =\ Q(n) \\
&\overset{\text{induction}}{\iff} \Big(n{>}0 \ \wedge\ p(n{-}1) \sim p'(n{-}1)\Big) \vee p_{Q_1}(n) \sim p'_{Q_1}(n) \\
&\iff \Big(n{>}0 \ \wedge\ p_0(n) \sim p_1(n{-}1)\Big) \vee p_0(n) \sim p_2(n) \\
&\iff p(n) \sim p'(n)
\end{aligned}$$

(g) $Q = \forall x : Q_1(x)$: The following provides a suitable construction involving the inductively assumed implementation for $Q_1(x)$:



That this construction suffices is readily verified:

$$\begin{aligned}
p(n) \sqsubseteq p'(n) &\iff \forall x : p_{Q_1}(x) \sqsubseteq p'_{Q_1}(x) \\
&\overset{\text{induction}}{\iff} \forall x : Q_1(x) \ =\ Q(n) \\
&\overset{\text{induction}}{\iff} \forall x : p_{Q_1}(x) \sim p'_{Q_1}(x) \\
&\iff p(n) \sim p'(n)
\end{aligned}$$

For any $Q \in \mathrm{OCL}$, the described construction terminates after $\mathcal{O}(\mathit{size}(Q))$ 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 $\mathcal{A} \sqsubseteq \mathcal{F}$, $\mathcal{F} \sqsubseteq \mathcal{A}$, and $\mathcal{A} \simeq \mathcal{F}$ problems. Again, we use the (inductively defined) reduction from TRUTHOCL ; only the particular constructions are now slightly different.

By an *implementation* we now mean a 4-tuple (A, F, F', A') where A, A' are one-counter automata, and F, F' are finite-state systems; the role of distinguished states is now played by the initial states, denoted q for A, f for F, f' for F', and q' for 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 7. *There is an algorithm which, given $Q = Q(x) \in OCL$ as input, halts after $\mathcal{O}(size(Q))$ steps and outputs an implementation (A, F, F', A') (where q, f, f', and q' are the initial control states of A, F, F', and A', respectively) such that for every $n \in \mathbb{N}$ we have*:

$$Q(n) \text{ is true iff } q(n) \sqsubseteq f \text{ iff } f' \sqsubseteq q'(n).$$

(*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 8. *Problems $\mathcal{A} \sqsubseteq \mathcal{F}$, $\mathcal{F} \sqsubseteq \mathcal{A}$, and $\mathcal{A} \simeq \mathcal{F}$ are **DP**-hard.*

Proof. Recalling that TRUTHOCL is **DP**-hard, **DP**-hardness of the first two problems readily follows from Proposition 7.

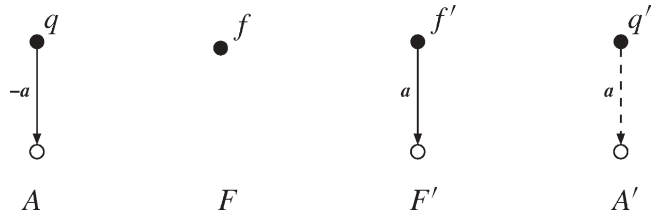
DP-hardness of the third problem follows from a simple (general) reduction of $\mathcal{A} \sqsubseteq \mathcal{F}$ to $\mathcal{A} \simeq \mathcal{F}$: given a one-counter automaton A with initial state q, and a finite-state system F with initial state f, we first transform F to F_1 by adding a new state f_1 and transition $f_1 \stackrel{a}{\to} f$, and then create A_1 by taking (disjoint) union of A, F_1 and adding $\overline{f_1} \stackrel{a}{\to} q$, where $\overline{f_1}$ is the copy of f_1 in A_1. Clearly $q(n) \sqsubseteq f$ iff $\overline{f_1}(n) \simeq f_1$. □

Proof of Proposition 7. We proceed by induction on the structure of Q. For each case, we construct an implementation (A, F, F', A') with distinguished states q, f, f' and q', respectively. In each case we demonstrate that the two bi-implications

$$q(n) \sqsubseteq f \iff Q(n) \iff f' \sqsubseteq q'(n)$$

hold for each $n \in \mathbb{N}$. In the constructions we use only two actions, a and b; this means that a state with non-decreasing a and b loops is *universal*, i.e., it can simulate every state.

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

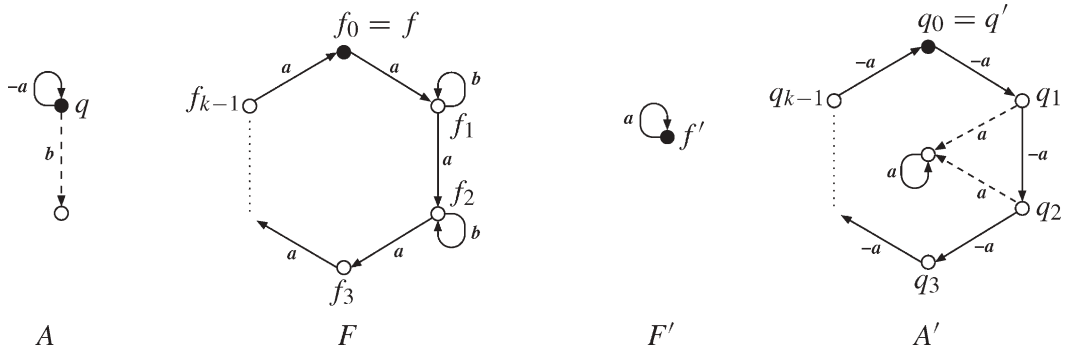


The validity of this implementation is readily verified:

$$q(n) \sqsubseteq f \iff n = 0 \iff f' \sqsubseteq q'(n)$$

(b,c) $Q = \lceil k \rceil \mid x$ 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 the formula $R_J(x)$ has an associated implementation 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).

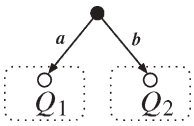
An implementation for $R_J(x)$, where for the point of illustration we have $1, 2 \in J$ but $0, 3, k-1 \notin J$, looks as follows:



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

$$q(n) \sqsubseteq f \iff (n \bmod k) \in J \iff f' \sqsubseteq q'(n)$$

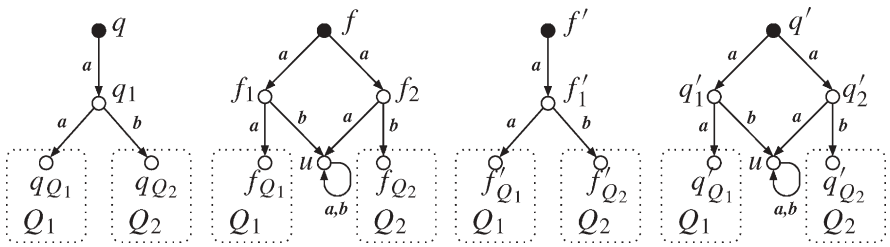
(d) $Q(x) = Q_1(x) \wedge Q_2(x)$: The elements of the implementation (A_Q, F_Q, F'_Q, A'_Q) for Q can be constructed from the respective elements of the implementations for Q_1, Q_2 (assumed by induction): A_Q from A_{Q_1} and A_{Q_2}; F_Q from F_{Q_1} and F_{Q_2}; F'_Q from F'_{Q_1} and F'_{Q_2}; and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



Verifying the correctness of this construction is straightforward:

$$\begin{aligned} q(n) \sqsubseteq f &\iff q_{Q_1}(n) \sqsubseteq f_{Q_1} \wedge q_{Q_2}(n) \sqsubseteq f_{Q_2} \\ &\overset{\text{induction}}{\iff} Q_1(n) \wedge Q_2(n) \;=\; Q(n) \\ &\overset{\text{induction}}{\iff} f'_{Q_1} \sqsubseteq q'_{Q_1}(n) \wedge f'_{Q_2} \sqsubseteq q'_{Q_2}(n) \\ &\iff f' \sqsubseteq q'(n) \end{aligned}$$

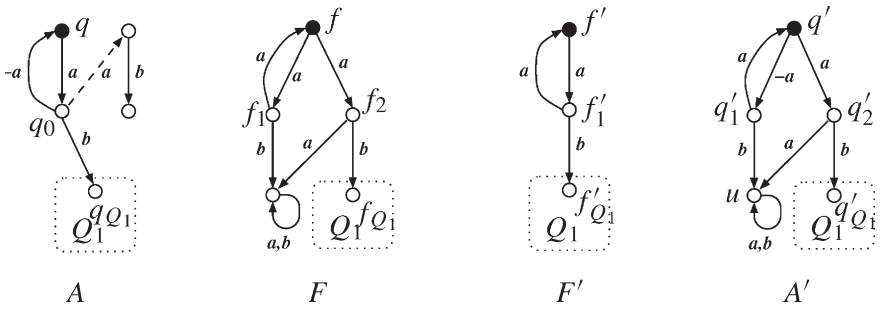
(e) $Q(x) = Q_1(x) \vee Q_2(x)$: As in case (d), the constructions use the inductively assumed implementations for $Q_1(x)$ and $Q_2(x)$; they are as follows:



Verifying the correctness of this construction is straightforward:

$$\begin{array}{rcl} q(n) \sqsubseteq f & \iff & q_1(n) \sqsubseteq f_1 \vee q_1(n) \sqsubseteq f_2 \\ & \iff & q_{Q_1}(n) \sqsubseteq f_{Q_1} \vee q_{Q_2}(n) \sqsubseteq f_{Q_2} \\ & \overset{\text{induction}}{\iff} & Q_1(n) \vee Q_2(n) \;=\; Q(n) \\ & \overset{\text{induction}}{\iff} & f'_{Q_1} \sqsubseteq q'_{Q_1}(n) \vee f'_{Q_2} \sqsubseteq q'_{Q_2}(n) \\ & \iff & f'_1 \sqsubseteq q'_1(n) \vee f'_1 \sqsubseteq q'_2(n) \\ & \iff & f' \sqsubseteq q'(n) \end{array}$$

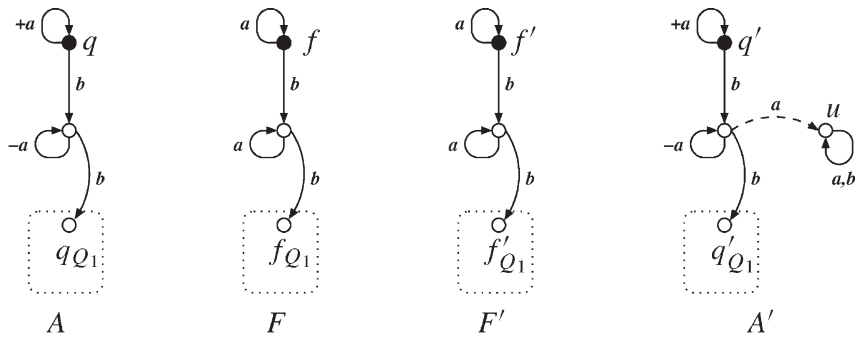
(f) $Q(x) = \exists y \leqslant x : Q_1(y)$ (where x, y are distinct): We use the following constructions involving the inductively assumed implementations for $Q_1(x)$:



We demonstrate the validity of this construction using a further induction on $n \in \mathbb{N}$; that is, in the case where $n > 0$ we assume that $q(n-1) \sqsubseteq f \iff Q(n-1) \iff f' \sqsubseteq q'(n-1)$.

$$\begin{array}{rcl} q(n) \sqsubseteq f & \iff & q_0(n) \sqsubseteq f_1 \vee q_0(n) \sqsubseteq f_2 \\ & \iff & \Big(n{>}0 \;\wedge\; q(n{-}1) \sqsubseteq f\Big) \vee q_{Q_1}(n) \sqsubseteq f_{Q_1} \\ & \overset{\text{induction}}{\iff} & \Big(n{>}0 \;\wedge\; \exists y \leqslant n{-}1 : Q_1(y)\Big) \vee Q_1(n) \;=\; Q(n) \\ & \overset{\text{induction}}{\iff} & \Big(n{>}0 \;\wedge\; f' \sqsubseteq q'(n{-}1)\Big) \vee f'_{Q_1} \sqsubseteq q'_{Q_1}(n) \\ & \iff & \Big(n{>}0 \;\wedge\; f'_1 \sqsubseteq q'_1(n{-}1)\Big) \vee f'_1 \sqsubseteq q'_2(n) \\ & \iff & f' \sqsubseteq q'(n) \end{array}$$

(g) $Q = \forall x : Q_1(x)$: The following provides a suitable construction involving the inductively assumed implementations for $Q_1(x)$:



That this construction suffices is readily verified:

$$\begin{aligned} q(n) \sqsubseteq f &\iff \forall x : q_{Q_1}(x) \sqsubseteq f_{Q_1} \\ &\overset{\text{induction}}{\iff} \forall x : Q_1(x) = Q(n) \\ &\overset{\text{induction}}{\iff} \forall x : f'_{Q_1} \sqsubseteq q'_{Q_1}(x) \\ &\iff f' \sqsubseteq q'(n) \end{aligned}$$

For any $Q \in$ OCL, 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. □

3.3. Model-checking the logic EF for one-counter nets

We prove that the model-checking problem for the logic EF and $\mathcal{N}$ processes is **DP**-hard, even for a fixed EF formula. We start with the following proposition:

Proposition 9. *There is an algorithm which, given $Q = Q(x) \in OCL$ as input, halts after $\mathcal{O}(size(Q))$ steps and outputs a one-counter net with a distinguished state q and an EF formula Φ_Q such that for every $k \in \mathbb{N}$ we have*:

$$Q(k) \text{ is true iff } q(k) \models \Phi_Q.$$

The constructed EF formula Φ_Q is not yet fixed; actually, it is not clear if the proof of Proposition 9 can be modified so that it returns the same EF formula for every $Q \in$ OCL. However, it is quite straightforward to modify the construction so that it produces the same EF formula for all those $Q \in$ OCL which can be obtained by applying the construction of (the proof of) Theorem 3 to some instance (φ, ψ) of SAT–UNSAT . Thus we obtain

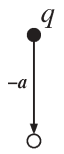
Proposition 10. *Let Q be an OCL formula which can be obtained by applying the construction of Theorem* 3. *There is a (fixed) EF formula Φ and an algorithm which, given Q on input, halts after $\mathcal{O}(size(Q))$ steps and outputs a one-counter net with a distinguished state q such that for every $n \in \mathbb{N}$ we have*:

$$Q(n) \text{ is true iff } q(n) \models \Phi.$$

Theorem 11. *The model-checking problem for the logic EF and $\mathcal{N}$ processes is **DP**-hard, even for a fixed EF formula.*

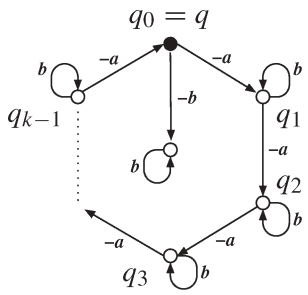
Proof of Proposition 9. We proceed by induction on the structure of Q. All steps are easy to verify and do not require detailed comments.

(a) $Q = (x = 0)$:



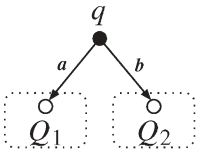
$\Phi_Q = [a]false$

(b,c) $Q = \lceil k \rceil \mid x$ or $Q = \lceil k \rceil \nmid x$, where $k>0$:



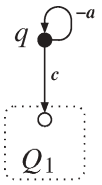
$\Phi_Q = \Diamond[b]false$ or $\Phi_Q = \Box\langle b\rangle true$

(d,e) $Q(x) = Q_1(x) \wedge Q_2(x)$ or $Q(x) = Q_1(x) \vee Q_2(x)$



$\Phi_Q = [a]\Phi_{Q_1} \wedge [b]\Phi_{Q_2}$ or $\Phi_Q = \langle a\rangle\Phi_{Q_1} \vee \langle b\rangle\Phi_{Q_2}$

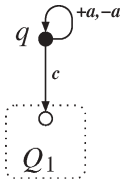
(f) $Q(x) = \exists y \leqslant x : Q_1(y)$ (where x, y are distinct):



$\Phi_Q = \Diamond\langle c\rangle\Phi_{Q_1}$

Here c is a fresh (i.e., previously unused) action.

(g) $Q = \forall x : Q_1(x)$:

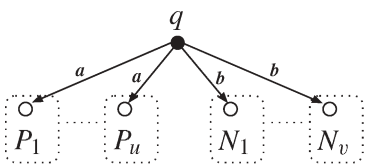


$\Phi_Q = \Box[c]\Phi_{Q_1}$

Again, c is a fresh action. □

Proof of Proposition 10. Note that the algorithm of Theorem 3 produces OCL formulas with an "almost fixed" structure: for a given instance (φ, ψ) of SAT–UNSAT, it basically plugs the φ and ψ (in a slightly modified form) into a fixed template. Therefore, we just need to modify the steps (d,e) of the previous algorithm.

(d,e) (i) $Q(x) = \bigvee_{i=1}^{u} P_i(x) \vee \bigvee_{j=1}^{v} N_j(x)$ where $u + v \geqslant 2$, and every P_i and N_j is of the form $\lceil k_i \rceil \mid x$ and $\lceil k'_j \rceil \nmid x$, respectively.

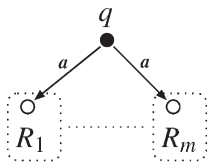


$\Phi_Q = \langle a\rangle\Phi_P \vee \langle b\rangle\Phi_N$

Here $\Phi_P = \Diamond[b]\mathit{false}$ and $\Phi_N = \Box\langle b\rangle\mathit{true}$ are the (fixed) formulas constructed for $P_i(x)$ and $N_j(x)$, respectively. Also note that if, e.g., $u = 0$, then the node q in the above graph has no a-successors, but the formula Φ_Q keeps its form.

(ii) $Q(x) = \bigwedge_{i=1}^{u} P_i(x) \wedge \bigwedge_{j=1}^{v} N_j(x)$ where $u + v \geqslant 2$, and every P_i and N_j is of the form $\lceil k_i \rceil \mid x$ and $\lceil k'_j \rceil \nmid x$, respectively. We construct the same net as in (i) and put $\Phi_Q = [a]\Phi_P \wedge [b]\Phi_N$.

(iii) $Q(x) = R_1(x) \vee \cdots \vee R_m(x)$ where $m \geqslant 2$ and every $R_i(x)$ is a conjunction of the form discussed in (ii).



$\Phi_Q = \langle a\rangle\Phi_R$

Here $\Phi_R = [a]\Phi_P \wedge [b]\Phi_N$ is the (fixed) formula constructed for $R_i(x)$.

(iv) $Q(x) = R_1(x) \wedge \cdots \wedge R_m(x)$ where $m \geqslant 2$ and every $R_i(x)$ is a disjunction of the form discussed in (i). We construct the same net as in (iii) and put $\Phi_Q = [a]\Phi_R$ where $\Phi_R = \langle a\rangle\Phi_P \vee \langle b\rangle\Phi_N$ is the (fixed) formula constructed for $R_i(x)$.

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.

Known results about equivalence-checking with one-counter automata are summarized in the following table where rows correspond to different equivalences, respectively, preorders, ($\approx$ denotes weak bisimilarity) and columns correspond to different pairs of checked systems.

	$\mathcal{A} \leftrightarrow \mathcal{A}$	$\mathcal{N} \leftrightarrow \mathcal{N}$	$\mathcal{A} \leftrightarrow \mathcal{F}$	$\mathcal{N} \leftrightarrow \mathcal{F}$
$\sim$	decidable [8] **DP**-hard	decidable [8] **DP**-hard	in **P** [14]	in **P** [14]
$\approx$	undecidable [18]	undecidable [18]	in **EXPTIME** **DP**-hard [14]	in **EXPTIME** **DP**-hard [14]
$\simeq$	undecidable [12]	decidable [1,12] **DP**-hard	in **EXPTIME** **DP**-hard	in **P** [15]
$\sqsubseteq$	undecidable [12]	decidable [1,12] **DP**-hard	in **EXPTIME** **DP**-hard	in **P** [15]
$\sqsupseteq$	undecidable [12]	decidable [1,12] **DP**-hard	in **EXPTIME** **DP**-hard	in **P** [15]

The **EXPTIME** upper bound of problems $\mathcal{A} \approx \mathcal{F}$, $\mathcal{N} \approx \mathcal{F}$, $\mathcal{A} \sqsubseteq \mathcal{F}$, $\mathcal{F} \sqsubseteq \mathcal{A}$, and $\mathcal{A} \simeq \mathcal{F}$ 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., [9,16]) and the modal μ-calculus which is **EXPTIME**-complete [24].

Known results for model-checking of one-counter automata can be summarized as follows:

- The model-checking problem for HML and $\mathcal{A}$ processes is in **P**.
- Model-checking with any logic which subsumes the logic EF and which is subsumed by the modal μ-calculus (it applies to, e.g., EF, CTL, CTL*, μ-calculus) is **DP**-hard and in **EXPTIME**. The lower complexity bound holds even for a fixed formula.

References

[1] P. Abdulla, K. Čerāns, Simulation is decidable for one-counter nets, in: Proceedings of CONCUR'98, Lecture Notes in Computer Science, vol. 1466, Springer, Berlin, 1998, pp. 253–268.
[2] E. Bach, J. Shallit, Algorithmic number theory, Efficient Algorithms, The MIT Press, Cambridge, MA, 1996.
[3] J. Bergstra, J. Klop, Algebra of communicating processes with abstraction, Theoretical Computer Science 37 (1985) 77–121.
[4] A. Bouajjani,P. Habermehl, R. Mayr, Automatic verification of recursive procedures with one integer parameter, in: Proceedings of MFCS 2001, Lecture Notes in Computer Science, vol. 2136, Springer, Berlin, 2001, pp. 198–211.
[5] S. Christensen, Decidability and decomposition in process algebras, Ph.D. Thesis, University of Edinburgh, 1993.
[6] E. Emerson, Temporal and modal logic, Handbook of Theoretical Computer Science B (1991) 995–1072.
[7] J. Esparza, J. Knoop, An automata-theoretic approach to interprocedural data-flow analysis, in: Proceedings of FoSSaCS'99, Lecture Notes in Computer Science, vol. 1578, Springer, Berlin, 1999, pp. 14–30.
[8] P. Jančar, Decidability of bisimilarity for one-counter processes, Information and Computation 158 (1) (2000) 1–17.
[9] P. Jančar, A. Kučera, R. Mayr, Deciding bisimulation-like equivalences with finite-state processes, Theoretical Computer Science 258 (1-2) (2001) 409–433.
[10] P. Jančar, A. Kučera, F. Moller, Simulation and bisimulation over one-counter processes, Proceedings of STACS 2000, Lecture Notes in Computer Science, vol. 1770, springer, Berlin, 2000, pp. 334–345.
[11] P. Jančar, A. Kučera, F. Moller, Z. Sawa. Equivalence-checking with one-counter automata: a generic method for proving lower bounds, in: Proceedings of FoSSaCS 2002, Lecture Notes in Computer Science, vol. 2303, Springer, Berlin, 2002, pp. 172–186.
[12] P. Jančar, F. Moller, Z. Sawa. Simulation problems for one-counter machines, in: Proceedings of SOFSEM'99, Lecture Notes in Computer Science, vol. 1725, Sprimger, Berlin, 1999, pp. 404–413.
[13] D. Kozen, Results on the propositional μ-calculus, Theoretical Computer Science 27 (1983) 333–354.
[14] A. Kučera, Efficient verification algorithms for one-counter processes, in: Proceedings of ICALP 2000, Lecture Notes in Computer Science, vol. 1853, Springer, Berlin, 2000, pp. 317–328.
[15] A. Kučera, On simulation-checking with sequential systems, in: Proceedings of ASIAN 2000, Lecture Notes in Computer Science, vol. 1961, Springer, Berin, 2000, pp. 133–148.
[16] A. Kučera, R. Mayr, Simulation preorder over simple process algebras, Information and Computation 173 (2) (2002) 184–198.
[17] R. Mayr, Strict lower bounds for model checking BPA, ENTCS 18 (1998).
[18] R. Mayr, Undecidability of weak bisimulation equivalence for 1-counter processes, in: Proceedings of ICALP 2003, Lecture Notes in Computer Science, vol. 2719, Springer, Berlin, 2003, pp. 570–583.
[19] R. Milner, Communication and Concurrency, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[20] C. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, MA, 1994.
[21] D. Park, Concurrency and automata on infinite sequences, in: Proceedings of the 5th GI Conference, Lecture Notes in Computer Science, vol. 104, Springer, Berlin, 1981, pp. 167–183.
[22] C. Stirling, Modal and temporal logics, Handbook of Logic in Computer Science 2 (1992) 477–563.
[23] R. van Glabbeek, The linear time—branching time spectrum, Handbook of Process Algebra (1999) 3–99.
[24] I. Walukiewicz, Pushdown processes: games and model-checking, Information and Computation 164 (2) (2001) 234–263.