Model Checking Succinct and Parametric One-Counter Automata

Stefan Göller[1], Christoph Haase[2], Joël Ouaknine[2], and James Worrell[2]

[1] Universität Bremen, Institut für Informatik, Germany
[2] Oxford University Computing Laboratory, UK

Abstract. We investigate the decidability and complexity of various model checking problems over one-counter automata. More specifically, we consider *succinct* one-counter automata, in which additive updates are encoded in binary, as well as *parametric* one-counter automata, in which additive updates may be given as unspecified parameters. We fully determine the complexity of model checking these automata against CTL, LTL, and modal μ-calculus specifications.

1 Introduction

Counter automata, which comprise a finite-state controller together with a number of counter variables, are a fundamental and widely-studied computational model. One of the earliest results about counter automata, which appeared in a seminal paper of Minsky's five decades ago, is the fact that two counters suffice to achieve Turing completeness [19].

Following Minsky's work, much research has been directed towards studying restricted classes of counter automata and related formalisms. Among others, we note the use of restrictions to a single counter, on the kinds of allowable tests on the counters, on the underlying topology of the finite controller (such as flatness [8, 17]), and on the types of computations considered (such as reversal-boundedness [15]). Counter automata are also closely related to Petri nets and pushdown automata.

In Minsky's original formulation, counters were represented as integer variables that could be incremented, decremented, or tested for equality with zero by the finite-state controller. More recently, driven by complexity-theoretic considerations on the one hand, and potential applications on the other, researchers have investigated additional primitive operations on counters, such as additive updates encoded in binary [3, 17] or even in *parametric* form, i.e., whose precise values depend on parameters [4, 14]. We refer to such counter automata as *succinct* and *parametric* respectively, the former being viewed as a subclass of the latter. Natural applications of such counter machines include the modeling of resource-bounded processes, programs with lists, recursive or multi-threaded programs, and XML query evaluation; see, e.g., [15, 3, 6].

In most cases, investigations have centered around the decidability and complexity of the *reachability* problem, i.e., whether a given control state can be reached starting from the initial configuration of the counter automaton. Various instances of the reachability problem for succinct and parametric counter automata are examined, for example, in [9, 12, 14].

The aim of the present paper is to study the decidability and complexity of *model checking* for succinct and parametric one-counter automata. In view of Minsky's result, we restrict our attention to *succinct one-counter automata (SOCA)* and *parametric one-counter automata (POCA)*. On the specification side, we focus on the three most prominent formalisms in the literature, namely the temporal logics CTL and LTL, as well as the modal μ-calculus. For a counter automaton $\mathbb{A}$ and a specification φ, we therefore consider the question of deciding whether $\mathbb{A} \models \varphi$, in case of POCA for all values of the parameters, and investigate both the *data* complexity (in which the formula φ is fixed) as well as the *combined* complexity of this problem. Our main results are summarized in Table 1.

One of the motivations for our work was the recent discovery that reachability is decidable and in fact NP-complete for both SOCA and POCA [12]. We were also influenced by the work of Demri and Gascon on model checking extensions of LTL over non-succinct, non-parametric one-counter automata [9], as well as the recent result of Göller and Lohrey establishing that model checking CTL on such counter automata is PSPACE-complete [11].

We note some interesting differences between our results and corresponding questions regarding finite automata. For the latter, the (combined) model checking problems for CTL, the μ-calculus, and LTL are respectively known to be P-complete, in $\mathsf{NP} \cap \mathsf{coNP}$, and PSPACE-complete. Somewhat surprisingly, for

		SOCA	POCA
CTL, μ-calculus	**data**	EXPSPACE-complete	Π_1^0-complete
	combined		
LTL	**data**	coNP-complete	
	combined	PSPACE-complete	coNEXP-complete

Table 1. The complexity of CTL, the modal μ-calculus, and LTL on SOCA and POCA.

SOCA and POCA, the complexity ordering is reversed and LTL becomes easier to model check than either CTL or the μ-calculus.

On a technical level, the most intricate result is the EXPSPACE-hardness of CTL model checking for SOCA, which requires several steps. We first show that EXPSPACE is 'exponentially LOGSPACE-serializable', adapting the known proof that PSPACE is LOGSPACE-serializable. Unfortunately, and in contrast to [11], this does not immediately provide an EXPSPACE lower bound. In a subsequent delicate stage of the proof, we show how to 'split' the counter in order simultaneously to perform PSPACE computations in the counter and manipulate numbers of exponential size in a SOCA of polynomial size.

Our paper is organized as follows. In Section 2 we introduce general notations. Section 3 deals with CTL and the modal μ-calculus. LTL model checking is content of Section 4. Finally, in Section 5 we give conclusions.

2 Preliminaries

By $\mathbb{Z}$ we denote the *integers* and by $\mathbb{N} = \{0, 1, 2, \ldots\}$ the denote the *naturals*. For each $i, j \in \mathbb{Z}$ we define $[i, j] = \{k \in \mathbb{Z} \mid i \leq k \leq j\}$ and $[j] = [1, j]$. For each $i, n \in \mathbb{N}$, let $\text{bit}_i(n)$ denote the i^{th} least significant bit of the binary representation of n. Hence $n = \sum_{i \in \mathbb{N}} 2^i \cdot \text{bit}_i(n)$. By $\text{bin}_m(n) = \text{bit}_0(n) \cdots \text{bit}_{m-1}(n)$ we denote the first m least significant bits written from *left to right*. When m is not important we just write $\text{bin}(n)$. Let p_i denote the i^{th} prime number for each $i \geq 1$, i.e. $p_1 = 2$, $p_2 = 3$ and so on. We define $\log(n) = \min\{i \geq 1 \mid 2^i > n\}$, in other words $\log(n)$ denotes the number of bits that are needed to represent n in binary. All polynomials $p : \mathbb{N} \to \mathbb{N}$ that occur in this paper are assumed to satisfy $p(n) \geq n$ for each $n \in \mathbb{N}$. For each word $v = a_1 \cdots a_n \in \Sigma^n$ over some finite alphabet Σ and each $i, j \in [n]$ define $v[i, j] = a_i \cdots a_j$ and $v(i) = v[i, i]$.

Turing machines and complexity theory: In the following, we introduce $f(n)$ space-bounded deterministic Turing machines (DTMs) in a suitable way for proving lower bounds. These contain precisely one input tape and one working tape. In our setting, the working alphabet is assumed to be $\{0, 1, \triangleright, \triangleleft\}$, where $\triangleright$ is the *left marker* and $\triangleleft$ is the *right marker*. The working tape of the initial configuration of such a DTM on an input $w \in \Sigma^n$ is assumed to be $\triangleright 0^{f(n)} \triangleleft$, whereas its input tape is $\triangleright w \triangleleft$. Before we define DTMs, let $\Upsilon = \{-1, 0, +1\}$ denote a set of directions.

Formally, a *$f(n)$-space bounded deterministic Turing machine (DTM)* is a tuple $\mathcal{M} = (S, \Sigma, s_0, F, \mu)$, where S is a finite set of *states*, Σ is a *finite input alphabet* with $\triangleright, \triangleleft \notin \Sigma$, $s_0 \in S$ is an *initial state*, $F \subseteq S$ is a set of *final states*, and with $\Sigma_{\mathcal{M}} = \Sigma \uplus \{\triangleright, \triangleleft\}$ we have that

$$\mu \quad : \quad S \times \Sigma_{\mathcal{M}} \times \{0, 1, \triangleright, \triangleleft\} \;\to\; S \times \Upsilon^2 \times \{0, 1, \triangleright, \triangleleft\}$$

is the *transition function*, where $\mu(s, b_1, b_2) = (s', \delta_1, \delta_2, b)$ means that $\mathcal{M}$ is currently in state s, its input head reads b_1, its working tape reads b_2, and $\mathcal{M}$ changes to state s', moves its input head in direction δ_1, moves its working head in direction δ_2, and writes the bit b. Moreover, we require that $\mathcal{M}$ writes a marker precisely when it reads a marker, more formally: $b_2 = m$ if and only if $b = m$ for each $m \in \{\triangleright, \triangleleft\}$. As expected, a *configuration of $\mathcal{M}$* is a tuple $(s, \triangleright w \triangleleft, \triangleright w' \triangleleft, i, j)$, where $s \in S$ is the current state, $w \in \Sigma^n$

is an input, $w' \in \{0,1\}^{f(n)}$ is the content of the working tape of $\mathcal{M}$, $i \in [0, n+1]$ is the current input head position, and $j \in [0, f(n)+1]$ is the current working head position. We define the *language* $L(\mathcal{M})$ *of* $\mathcal{M}$ to consist of all words $w \in \Sigma^*$ such that the *initial configuration* $(s_0, \triangleright w \triangleleft, \triangleright 0^{f(|w|)} \triangleleft, 0, 0)$ reaches a configuration $(s_0, \triangleright w \triangleleft, \triangleright w' \triangleleft, i, j)$ satisfying $s \in F$.

By LOGSPACE, PSPACE, and EXPSPACE we denote the class of all problems that can be decided by DTM that is logarithmically, polynomially, exponentially space bounded, respectively. Recall that Σ_1^0 (resp. Π_1^0) is the class of all languages that are (resp. whose complements are) recursively enumerable.

Transition systems: In the following, we fix a countable set of atomic propositions $\mathcal{P}$. A *transition system* is a tuple $T = (S, \{S_\rho \mid \rho \in \mathcal{P}\}, \rightarrow)$, where S is a set of *states*, $S_\rho \subseteq S$ for each $\rho \in \mathcal{P}$ and S_ρ is non-empty for finitely many $\rho \in \mathcal{P}$, and finally $\rightarrow \subseteq S \times S$ is a set of *transitions*. We prefer to use the infix notation $s_1 \rightarrow s_2$ to abbreviate $(s_1, s_2) \in \rightarrow$. An *infinite path* is an infinite sequence $\pi = s_0 \rightarrow s_1 \cdots$. For each infinite path $\pi = s_0 \rightarrow s_1 \rightarrow \cdots$ and $i \in \mathbb{N}$, we denote by π^i the path $s_i \rightarrow s_{i+1} \cdots$ and by $\pi(i)$ the state s_i. Define the *trace of* π as $\tau(\pi) : \mathbb{N} \rightarrow 2^{\mathcal{P}}$ where $\tau(i) = \{\rho \in \mathcal{P} \mid s_i \in S_\rho\}$ for each $i \in \mathbb{N}$.

Succinct and parametric one-counter automata: A *succinct one-counter automaton (SOCA)* is a tuple $\mathbb{S} = (Q, \{Q_\rho \mid \rho \in \mathcal{P}\}, E, \lambda)$, where Q is a finite set of *control states*, $Q_\rho \subseteq Q$ for each $\rho \in \mathcal{P}$ and Q_ρ is non-empty for finitely many $\rho \in \mathcal{P}$, $E \subseteq Q \times Q$ is a finite set of *transitions*, and $\lambda : E \rightarrow \mathbb{Z} \cup \{\mathsf{zero}\}$ labels the edges with decrements, increments, and zero tests. A *parametric one-counter automaton (POCA)* is a tuple $\mathbb{P}(X) = (Q, \{Q_\rho \mid \rho \in \mathcal{P}\}, E, \lambda)$, where the first three components are same as for a SOCA, where X is a finite set of *parameters over the natural numbers*, and where $\lambda : E \rightarrow (\mathbb{Z} \cup \{\mathsf{zero}\} \cup \{\circ x \mid \circ \in \{+,-\}, x \in X\})$. For each assignment $\sigma : X \rightarrow \mathbb{N}$ the *induced SOCA* is defined as $\mathbb{P}^\sigma = (Q, \{Q_\rho \mid \rho \in \mathcal{P}\}, E, \lambda')$ where $\lambda'(e) = \circ\sigma(x)$ whenever $\lambda(e) = \circ x$ for some $\circ \in \{+,-\}$ and $\lambda'(e) = \lambda(e)$ otherwise. If $X = \{x\}$ is a singleton, then we also write $\mathbb{P}(x)$ for $\mathbb{P}(X)$. The *size* of a POCA is defined as $|\mathbb{P}| = |Q| + |X| + |E| \cdot \max\{\log(|a|) \mid a \in \lambda(E) \cap \mathbb{Z}\}$. Hence, we represent each appearing integer in binary. The size of a SOCA is defined analogously. A SOCA $\mathbb{S} = (Q, \{Q_\rho \mid \rho \in \mathcal{P}\}, E, \lambda)$ describes a transition system $T(\mathbb{S}) = (Q \times \mathbb{N}, \{Q_\rho \times \mathbb{N} \mid \rho \in \mathcal{P}\}, \rightarrow)$, where for each $q_1, q_2 \in Q$ and each $n_1, n_2 \in \mathbb{N}$ we have $q_1(n_1) \rightarrow q_2(n_2)$ if and only if either $\lambda(q_1, q_2) = n_2 - n_1$, or ($n_1 = n_2 = 0$ and $\lambda(q_1, q_2) = \mathsf{zero}$).

3 CTL Model Checking

In Section 3.1 we introduce the syntax and semantics of CTL. Since the upper bounds for the modal μ-calculus follow from known results and since the lower bounds follow from CTL, we do not introduce the modal μ-calculus formally. In Section 3.2 we state an EXPSPACE upper bound for the combined complexity of CTL and the modal μ-calculus on SOCA and a Π_1^0 upper bound on POCA. In Section 3.3 we recall basic notions from complexity theory such as serializability and results on Chinese remainder representation. In Section 3.4 we prove the main result of this section, namely that the data complexity of CTL on SOCA is EXPSPACE-hard. Finally, we prove a Π_1^0 lower bound for the data complexity of CTL on POCA in Section 3.5.

3.1 CTL: Syntax and Semantics

Formulas φ of CTL are given by the following grammar, where ρ ranges over $\mathcal{P}$:

$$\varphi \quad ::= \quad \rho \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}(\varphi \mathsf{U} \varphi) \mid \mathsf{E}(\varphi \mathsf{WU} \varphi)$$

We have the following abbreviations: $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$, $\mathtt{tt} = \rho \vee \neg\rho$ for some atomic proposition $\rho \in \mathcal{P}$, and $\mathsf{EF}\varphi = \mathsf{E}(\mathtt{tt} \mathsf{U} \varphi)$. The *size* of a CTL formula is inductively defined as follows: $|\rho| = 1$ for each $\rho \in \mathcal{P}$, $|\neg\varphi| = |\mathsf{EX}\varphi| = |\varphi| + 1$, $|\varphi_1 \wedge \varphi_2| = |\mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)| = |\mathsf{E}(\varphi_1 \mathsf{WU} \varphi_2)| = |\varphi_1| + |\varphi_2| + 1$. Given a transition system $T = (S, \{S_\rho \mid \rho \in \mathcal{P}\}, \rightarrow)$, a state $s \in S$, and some CTL formula φ, define $(T, s) \models \varphi$ by induction on the structure of φ as follows:

$$
\begin{aligned}
(T,s) \models \rho &\iff s \in S_p \quad \text{for each } \rho \in \mathcal{P} \\
(T,s) \models \varphi_1 \wedge \varphi_2 &\iff (T,s) \models \varphi_1 \text{ and } (T,s) \models \varphi_2 \\
(T,s) \models \neg\varphi &\iff (T,s) \not\models \varphi \\
(T,s) \models \mathsf{EX}\varphi &\iff (T,t) \models \varphi \text{ for some } t \in S \text{ with } s \to t \\
(T,s) \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2) &\iff \exists s_0, \cdots, s_n \in S, n \geq 0 : s_0 = s, (T,s_n) \models \varphi_2 \text{ such that} \\
&\quad\ \forall i \in [0, n-1] : (T,s_i) \models \varphi_1 \text{ and } s_i \to s_{i+1} \\
(T,s) \models \mathsf{E}(\varphi_1 \mathsf{W} \mathsf{U} \varphi_2) &\iff (T,s) \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2) \text{ or} \\
&\quad\ \exists s_0, s_1, \ldots \in S : \forall i \geq 0 : (T,s_i) \models \varphi_1 \text{ and } s_i \to s_{i+1}
\end{aligned}
$$

Let us define the *CTL model checking problem* on SOCA and POCA respectively.

CTL MODEL CHECKING ON SOCA

INPUT: SOCA $\mathbb{S} = (Q, \{Q_\rho \mid \rho \in \mathcal{P}\}, E, \lambda)$, $q \in Q$, $n \in \mathbb{N}$ in binary, and a CTL formula φ.
QUESTION: $(T(\mathbb{S}), q(n)) \models \varphi$?

CTL MODEL CHECKING ON POCA

INPUT: POCA $\mathbb{P}(X) = (Q, \{Q_\rho \mid \rho \in \mathcal{P}\}, E, \lambda)$, $q \in Q$, $n \in \mathbb{N}$ in binary, and a CTL formula φ.
QUESTION: $(T(\mathbb{P}^\sigma), q(n)) \models \varphi$ for every $\sigma : X \to \mathbb{N}$?

3.2 Upper bounds

Let us first state upper bounds on model checking CTL and the modal μ-calculus on SOCA. *One-counter automata (OCA)* are SOCA in which the numbers that occur in the transition labels are represented in unary instead in binary. The following theorem gives an upper bound on model checking the *modal μ-calculus* on OCA. Consult [1] for more details on the modal μ-calculus.

Theorem 1 ([21]). *The combined complexity of the modal μ-calculus on OCA is in* PSPACE.

Since every SOCA can be transformed into an exponentially larger OCA and since each CTL formula can be translated into an alternation-free μ-calculus formula with a linear blowup, the following corollary is immediate.

Corollary 2. *The combined complexity of CTL and the modal μ-calculus on SOCA is in* EXPSPACE.

The following upper bound for CTL on POCA is straightforward.

Corollary 3. *The combined complexity of CTL and the modal μ-calculus on POCA is in Π_1^0.*

Proof. Let $\mathbb{P}(X) = (Q, \{Q_\rho \mid \rho \in \mathcal{P}\}, E, \lambda)$ be a POCA, $q \in Q$, and $n \in \mathbb{N}$, and φ be a formula. From Corollary 2 it follows that for each given $\sigma : X \to \mathbb{N}$ the question $(T(\mathbb{P}^\sigma), q(n)) \models \varphi$ is a decidable predicate. Hence, the question if $(T(\mathbb{P}^\sigma), q(n)) \models \varphi$ for each $\sigma : X \to \mathbb{N}$ is a Π_1^0-predicate since we can encode the set of all such assignments into the naturals via Gödel encoding. □

3.3 Serializability and Chinese remainder representation

For a language $L \subseteq \Sigma^*$ let $\chi_L : \Sigma^* \to \{0,1\}$ denote the *characteristic function of L*, i.e. $\chi_L(x) = 1$ if $x \in L$ and $\chi_L(x) = 0$ otherwise, for each $x \in \Sigma^*$. We define the *lexicographic order on n-bit strings* as $\preceq_n \subseteq \{0,1\}^n \times \{0,1\}^n$, where $x \preceq y$ if and only if $\text{bin}_n(x) \leq \text{bin}_n(y)$, e.g. $101 \preceq_3 011$.

Definition 4. *Let $\mathcal{C}$ be a complexity class. We say a language L is* $\mathcal{C}$-serializable via some language $R \subseteq \{0,1\}^*$ *if there is some polynomial $p(n)$, and some language $U \in \mathcal{C}$ such that for all $x \in \{0,1\}^n$*

$$
x \in L \qquad \iff \qquad \chi_U\left(x, 0^{p(n)}\right) \quad \cdots \quad \chi_U\left(x, 1^{p(n)}\right) \in R,
$$

where with '$\cdots$' we refer to $\preceq_{p(n)}$ in each of the second components.

The following theorem was proven in [10].

Theorem 5 (Theorem 22 in [10]). *For every L in* PSPACE *there is some regular language R such that L is logspace-uniformly* AC^0*-serializable via R.*

As an immediate consequence, we obtain the following corollary.

Corollary 6. *For every L in* PSPACE *there is some regular language R such that L is* LOGSPACE*-serializable via R.*

We remark that our definition of serializability is adopted from [10] and differs slightly from the standard notion as used in [5, 13, 23]. We aim at lifting Corollary 6 via an appropriate notion of serializability to EXPSPACE.

Definition 7. *Let $\mathcal{C}$ be some complexity class. A language L is* exponentially $\mathcal{C}$-serializable via some language $R \subseteq \{0,1\}^*$ *if there is some polynomial $p(n)$ and some language $U \in \mathcal{C}$ such that for all $x \in \{0,1\}^n$*

$$x \in L \quad \iff \quad \chi_U\left(x, 0^{2^{p(n)}}\right) \quad \cdots \quad \chi_U\left(x, 1^{2^{p(n)}}\right) \in R,$$

where with '$\cdots$' we refer to $\preceq_{2^{p(n)}}$ in each of the second components.

The following proposition is folklore and immediate.

Proposition 8. *Let $L \subseteq \Sigma^*$ be in* EXPSPACE*. Then there is some polynomial q such that the padded language $\{x\$^{2^{q(|x|)}} \mid x \in L\} \subseteq (\Sigma \uplus \{\$\})^*$ is in* PSPACE*.*

Theorem 9. *For every language L in* EXPSPACE *there is some regular language R such that L is exponentially* LOGSPACE*-serializable via R.*

Proof. Let $L \subseteq \Sigma^*$ be some language in EXPSPACE. Then by Proposition 8 for some polynomial q the language $L' = \{x\$^{2^{q(|x|)}} \mid x \in L\}$ is in PSPACE. Due to LOGSPACE-serializability of PSPACE by Corollary 6, there exists some polynomial p', some regular language R' and some $U' \in$ LOGSPACE such that for each $y \in (\Sigma \uplus \{\$\})^*$ we have

$$y \in L' \quad \iff \quad \chi_{U'}\left(y, 0^{p'(|y|)}\right) \ \cdots \ \chi_{U'}\left(y, 1^{p'(|y|)}\right) \in R', \tag{1}$$

where '$\cdots$' refers to the lexicographic order $\preceq_{p'(|y|)}$. To prove the lemma we have to find some regular language R, some polynomial p and some $U \in$ LOGSPACE such that for all $x \in \Sigma^*$ we have

$$x \in L \quad \iff \quad \chi_U\left(x, 0^{2^{p(|x|)}}\right) \cdots \chi_U\left(x, 1^{2^{p(|x|)}}\right) \in R,$$

where here '$\cdots$' refers to the lexicographic order $\preceq_{2^{p(|x|)}}$. For now, we choose p sufficiently fast growing, let us postpone this to the end of the proof. Let us describe the language U and consider inputs (x, w) with $x \in \Sigma^n$ and $w \in \{0,1\}^{2^{p(n)}}$; moreover let $y = x\$^{2^{q(|x|)}}$ be the padding of x. Our language U consists of all such pairs (x, w) such that w can be factorized as $w = bz0^l$ for some $b \in \{0,1\}$ and some $l \geq 0$ such that $b = 0$ or $(y, z) \in U'$. Roughly speaking, the goal of U is to simulate U' on the corresponding padded word, however the length of the second component w has to be a power of two, in other words the last l bits of w contain redundant information. We will be able to filter out this redundant information via the bit b. Let us make this more precise. Let

$$\begin{aligned} \gamma' &= \chi_{U'}\left(y, 0^{p'(|y|)}\right) \ \cdots \ \chi_{U'}\left(y, 1^{p'(|y|)}\right) \quad \text{and} \\ \gamma &= \chi_U\left(x, 0^{2^{p(|x|)}}\right) \ \cdots \ \chi_U\left(x, 1^{2^{p(|x|)}}\right). \end{aligned}$$

Then by definition of U, the first $2 \cdot p'(|y|)$ bits of γ are $1\gamma'(1)1\gamma'(2)\cdots 1\gamma'(p'(|y|))$, where the remaining bits of γ are all 0. Hence, when reading γ in *pairs of bits*, precisely when the first of the two bits is set to 1 we read some relevant information.

We have to give some regular language R such that $\gamma' \in R'$ if and only if $\gamma \in R$. Recall that regular languages are closed under shuffle product $\|$ and homomorphisms. We define $R = \varphi(R' \| \{a\}^*)$, where $\varphi : \{0,1,a\} \to \{0,1\}^*$ is the following homomorphism: $\varphi(a) = 00$, $\varphi(0) = 10$, and $\varphi(1) = 11$.

As expected, we choose p to be an arbitrary polynomial satisfying $2^{p(n)} \geq 1 + p'(2^{q(n)} + n)$ for all $n \geq 0$. Finally, let us show that U is in LOGSPACE. Let (x, w) be an input to U and let again $y = x\$^{2^{q(|x|)}}$ be the padding of x. Then it is straightforward to decide in logarithmic space if w can be factorized as $w = bzv$ such that $b \in \{0,1\}$, $|z| = p'(2^{q(|x|)} + |x|)$ and $v \in \{0\}^*$. The only thing that might remain to be verified in logarithmic space is whether $(y, z) \in U'$, which in turn boils down to simulating some logarithmic space bounded TM for U on input (y, z). But this is possible since $|y| + |z| \leq 2 \cdot (|x| + |w|)$ as

$$|y| + |z| = |x| + 2^{q(|x|)} + p'(2^{q(|x|)+|x|}) \leq 2 \cdot p'(2^{q(|x|)} + |x|) \leq 2 \cdot 2^{p(|x|)} \leq 2 \cdot (|x| + |w|).$$

□

Chinese remainder representation: For every $m, M \in \mathbb{N}$ we denote by $\mathrm{CRR}_m(M)$ *the Chinese remainder representation* of M as the Boolean tuple $(b_{i,c})_{i \in [m], 0 \leq c < p_i}$, where $b_{i,c} = 1$ if $M \bmod p_i = c$ and $b_{i,c} = 0$ otherwise. The following theorem tells us that in logarithmic space we can compute the binary representation of a natural number that is given in Chinese remainder representation. It is a consequence of the result that division is in logspace-uniform NC^1, proven in [7].

Theorem 10 ([7] Theorem 3.3). *The following problem is in* LOGSPACE*:*
INPUT: $\mathrm{CRR}_m(M), j \in [m], b \in \{0,1\}$.
QUESTION: $bit_j(M \bmod 2^m) = b$*?*

3.4 EXPSPACE-hardness of the data complexity of CTL on SOCA

In the rest of this section, we give the proof of EXPSPACE-hardness of the data complexity of CTL on SOCA. Let $L \subseteq \{0,1\}^*$ be an arbitrary language in EXPSPACE. Then by Theorem 9, there is some regular language $R \subseteq \{0,1\}^*$ such that L is exponentially LOGSPACE-serializable via R. Hence there is some language $U \in$ LOGSPACE and some polynomial p such that for all $x \in \{0,1\}^n$ we have

$$x \in L \qquad \Longleftrightarrow \qquad \chi_U\left(x, 0^{2^{p(n)}}\right) \quad \cdots \quad \chi_U\left(x, 1^{2^{p(n)}}\right) \in R, \tag{2}$$

where '$\cdots$' refers to the lexicographic order $\preceq_{2^{p(n)}}$ on the bit strings on the right-hand side.
For the rest of this section, let us fix an input $x_0 \in \{0,1\}^n$. Let $N = p(n)$ and let $A = (Q, \{0,1\}, q_0, \delta, F)$ be some deterministic finite automaton with $L(A) = R$. Let us briefly recall what A consists of: Q is a finite set of *states*, $\{0,1\}$ is the *input alphabet*, $q_0 \in Q$ is the *initial state*, $\delta : Q \times \{0,1\} \to Q$ is the *transition function*, and $F \subseteq Q$ is the set of *final states*. Let us describe equivalence (2) differently: We have $x_0 \in L$ if and only if the program in Figure 1 returns `true`.

$q \in Q$; $q := q_0$;
$d \in \mathbb{N}$; $d := 0$;
$b \in \{0,1\}$;
`while` $d \neq 2^{2^N}$ `loop`
    $b := \chi_U(x_0, \mathrm{bin}_{2^N}(d))$;
    $q := \delta(q, b)$;
    $d := d + 1$;
`endloop`
`return` $q \in F$;

Fig. 1. A program that returns `true` if and only if $x_0 \in L$

Our goal is to mimic the execution of the program in Figure 1 by a fixed CTL formula and by a SOCA that depends on our input x_0. Before we start with the reduction, let us discuss the obstacles that arise:

(A) We need some way of storing d on the counter. Of course there are a lot of ways to do this, but since we want to access all bits of d in the assignment $b := \chi_U(x_0, \mathrm{bin}_{2^N}(d))$, the most natural way is probably to represent d in binary. However, for this 2^N bits are required. More problematically, we need to be able to check if d is equal to 2^{2^N}. This cannot be achieved by a transition in a SOCA that subtracts 2^{2^N}, since the representation of this number requires exponentially many bits in n.

(B) As in [11], a solution to obstacle **(A)** is to store d in Chinese remainder representation with the first 2^N prime numbers. A polynomial number of bits (in n) suffice to represent each of the occurring prime numbers, but we need exponentially many of them. Thus, we cannot equip a polynomial size SOCA with transitions for each prime number, simply because there are too many of them.

(C) The assignment $b := \chi_U(x_0, \mathrm{bin}_{2^N}(d))$ implies that we need to simulate on the counter a logarithmically space bounded DTM for the language U on an exponentially large input (in n). Speaking in terms of the input size n, this means that we need to provide polynomially many bits on the counter that can be used to describe the working tape for this DTM. However, we need to provide some on-the-fly mechanism for reading the input.

To achieve this, let us give a high-level description of how we proceed. In a first step, we carefully design a data structure on the counter and describe the intuition behind it. In a second step, we give five queries which we aim at implementing via fixed CTL formulas and by SOCA that can be computed from x_0 in logarithmic space.

The data structure and how to access it: Let $K = n + 2^N + 1$ denote the number of bits that are required to store an input for U plus one. Let $\alpha = \log K$ denote the number of bits that we require for storing a pointer to an input for U and let β be the number of bits that suffice for storing the K^{th} prime. Hence $\alpha = O(N)$ and by the Prime Number Theorem, it follows that $\beta \in O(\log(K \log(K))) = O(N)$. The number α and such a sufficiently large number β can be computed from x_0 in logarithmic space.

Let us describe how we will interpret the counter in our reduction. Assume that the counter value is $v \in \mathbb{N}$. Instead of treating v as a natural number, we are interested only in the l least significant bits V of the binary representation of v, where l is some number that is exponentially bounded in n; the precise value of l will be made clear below. Assume $V = \mathrm{bit}_0(v) \cdots \mathrm{bit}_{l-1}(v)$. We read V to be factorized into blocks of bits

$$V \quad = \quad I\ M\ C\ J\ X\ Y\ Z\ B, \qquad (\bigstar)$$

where

- $I \in \{0,1\}^\alpha$ represents a prime number index,
- $M \in \{0,1\}^\beta$ represents I^{th} prime number p_I,
- $C \in \{0,1\}^\beta$ represents some residue class modulo M,
- $J \in \{0,1\}^\alpha$ represents a pointer to some bit of B,
- each X, Y and Z consist of polynomially many bits (in n) and represent the working tape of three space bounded DTMs that we will comment on later in more detail, and
- $B \in \{0,1\}^{n+2^N+1}$ with $B = xB'$ for some $x \in \{0,1\}^n$ and some $B' \in \{0,1\}^{2^N+1}$, B represents the current input for U, where x is reserved to represent our input x_0 and where B' represents the counter d from program in Figure 1 from above. The block B' consists of $2^N + 1$ bits since we want to be able to test if $d = 2^{2^N}$.

Let us introduce some more notation for addressing leftmost (starting to count from 0) and rightmost bit positions in each of the above sequences of bits in V. For each such sequence Θ let $\Theta_\Leftarrow$ and $\Theta_\Rightarrow$ denote the respective positions of the leftmost and rightmost bits of Θ within the bit string V, e.g. $I_\Rightarrow = \alpha - 1$ and $C_\Leftarrow = \alpha + \beta$.

Important remark: Throughout the rest of this section v will denote an arbitrary natural number, moreover I, M, C, J, X, Y, Z, and B will implicitly be coupled with v. Note that all of the bit strings have polynomial length in n except for B. Moreover, we identify each of the blocks with the natural number they represent.

A very simple but important gadget that we need is to decide, for each $b \in \{0,1\}$ if the i^{th} bit of v is b.

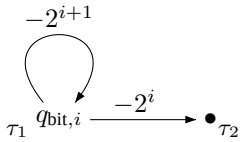
Lemma 11. *For each bit $b \in \{0,1\}$ there exists a fixed CTL formula $\varphi_{bit,b}$ such that the following is computable in logarithmic space:*

INPUT: $i \in \mathbb{N}$ in unary.

OUTPUT: A SOCA $\mathbb{S}_{bit,i}$ and some control state $q_{bit,i}$ such that

$$(T(\mathbb{S}_{bit,i}), q_{bit,i}(v)) \models \varphi_{bit,b} \quad \Longleftrightarrow \quad bit_i(v) = b.$$

Proof. The SOCA $\mathbb{S}_{\text{bit},i}$ has the two atomic propositions τ_1 and τ_2 and is depicted below:



The simple way to check if $\text{bit}_i(v) = 1$ (resp. $\text{bit}_i(v) = 0$) is to repeatedly subtract 2^{i+1} from v until no longer possible and then to check if 2^i can (resp. cannot) be subtracted. Hence we put

$$\begin{aligned} \varphi_{\text{bit},1} &= \tau_1 \wedge \mathsf{EF}(\tau_1 \wedge \neg\mathsf{EX}\tau_1 \wedge \mathsf{EX}\tau_2) \qquad \text{and} \\ \varphi_{\text{bit},0} &= \tau_1 \wedge \mathsf{EF}(\tau_1 \wedge \neg\mathsf{EX}\tau_1 \wedge \neg\mathsf{EX}\tau_2). \end{aligned}$$

□

Queries that we need to implement: We will implement the following five queries Q1 to Q5 by instances of the model checking problem, where each query builds on top of its preceding queries.

(Q1) When assuming $C < M$, does $B \equiv C \bmod M$ hold?

(Q2) Is M the I^{th} prime number, i.e. $M = p_I$?

(Q3) What is $\text{bit}_J(B)$?

(Q4) Does $(B[1,n], B[n+1, n+2^N]) \in U$ hold?

(Q5) Does $x_0 \in L$ hold?

We implement each of the five queries by providing fixed CTL formulas and SOCA that can be computed from x_0 in logarithmic space. EXPSPACE-hardness of the data complexity of CTL on SOCA will hence follow from the implementation of Q5. First, let us give an implementation of Q1.

Lemma 12. *There exists some fixed CTL formula φ_{mod} such that we can compute from x_0 in logarithmic space some SOCA $\mathbb{S}_{mod}$ and some control state q_{mod} such that $(T(\mathbb{S}_{mod}), q_{mod}(v)) \models \varphi_{mod}$ if and only if $B \equiv C \bmod M$.*

Proof. The SOCA $\mathbb{S}_{\text{mod}}$ contains the three atomic propositions ρ_0, ρ_1, and ρ_2 and is depicted in Figure 2. The CTL formula φ_{mod} expresses that we traverse the upper sequence of diamonds and thereby repeatedly subtract M from B. The number of diamonds both in the first and the second row equals β, the number of bits of M and of C. In the upper row, one diamond corresponds to one bit of M. In case bit the rightmost bit of M (in other words bit $M_{\Rightarrow}$ of the counter) is 1, which we can verify by a transition to the initial control state of the SOCA $\mathbb{S}_{\text{bit},M_{\Rightarrow}}$, we subtract $2^{B_{\Leftarrow}+\beta-1}$ from B, otherwise we do not modify the counter value. After that, we move on to the second diamond, which represents the second rightmost bit of M, and so on. Hence, traversing a cycle that starts and ends in our initial control state q_{mod} will correspond to subtracting M once. Then we traverse the lower sequence of diamonds and similarly subtract C, but this this time only once. Finally, after having traversed the lower sequence of diamonds and seeing the atomic proposition ρ_2, we check if $B = 0$ by trying to subtract $2^{B_{\Leftarrow}}$. Finally, let us give the formula φ_{mod} :

$$\varphi_{\text{mod}} \quad = \quad \mathsf{E}\left(\bigwedge_{i\in\{0,1\}} \rho_i \rightarrow \mathsf{EX}\varphi_{\text{bit},i}\right) \mathsf{U}(\rho_2 \wedge \neg\mathsf{EX}\mathtt{tt})$$

□

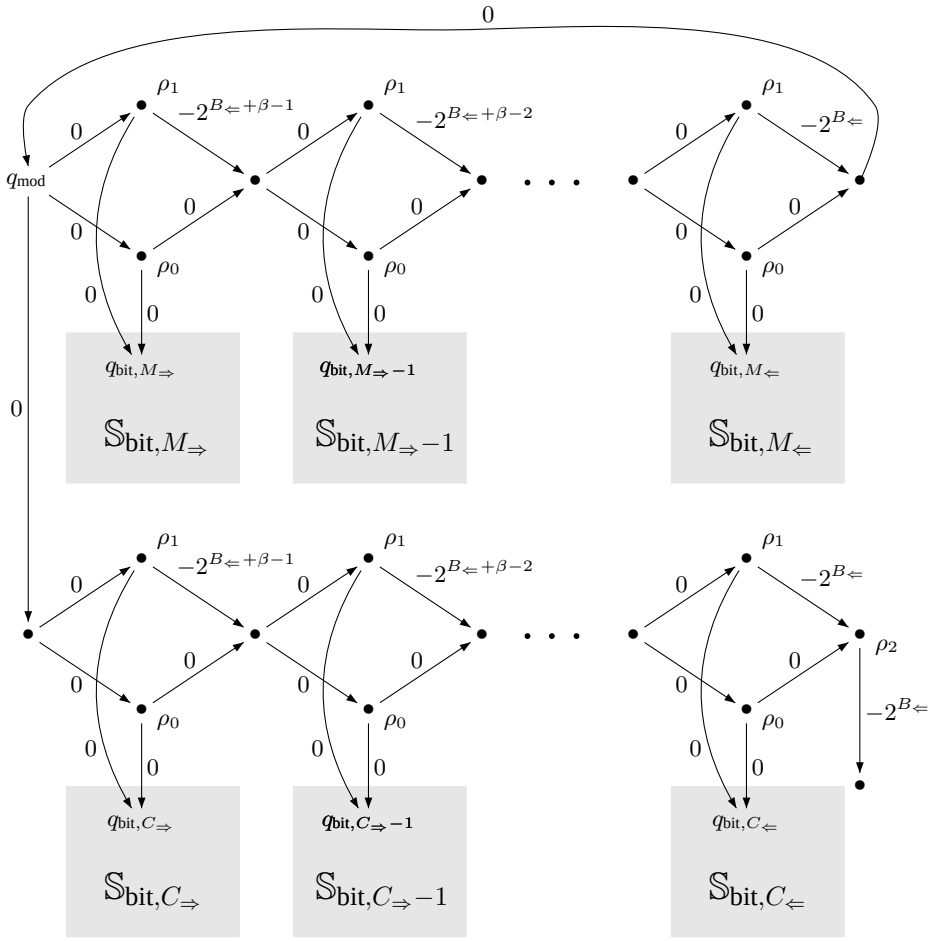


Fig. 2. The SOCA $\mathbb{S}_{\text{mod}}$
for checking if $B \equiv C \bmod M$.

We need the following proposition for implementing Q2. Its proof is straightforward.

Proposition 13. *The following problem is in* PSPACE*:*
INPUT: $bin(i)\$bin(m)$
QUESTION: $m = p_i$*?*

Lemma 14. *There is some fixed CTL formula φ_{prime} such that from x_0 we can compute in logarithmic space some SOCA $\mathbb{S}_{prime}$ and some control state q_{prime} such that $(T(\mathbb{S}_{prime}), q_{prime}(v)) \models \varphi_{prime}$ if and only if $M = p_I$.*

Proof. By Proposition 13 there is some $q(n)$ space-bounded DTM $\mathcal{M} = (S, \Sigma, s_0, F, \mu)$ over the alphabet $\Sigma = \{0, 1, \$\}$ that decides, given $\text{bin}(i)\$\text{bin}(m)$, whether $m = p_i$, where q is some polynomial. The idea is to simulate $\mathcal{M}$ on input $z = I\$M$ using X from ($\bigstar$) as a working tape. For this, we define our block X to consist of $l = q(|I| + |M| + 1) = q(\alpha + \beta + 1)$ many bits. Moreover, we assume w.l.o.g. that $\mathcal{M}$'s behaviour on input $I\$M$ is independent of the content of its initial working tape: This can be achieved by adding extra states to $\mathcal{M}$ that first write 0^l onto the working tape. Hence, the initial configuration of $\mathcal{M}$ on input z is $(s_0, \triangleright z \triangleleft, \triangleright w \triangleleft, 0, 0)$ for some $w \in \{0,1\}^l$. Before we give our SOCA $\mathbb{S}_{\text{prime}}$, we describe the computation of $\mathcal{M}$ on input $I\$M$ as a pseudo program that terminates if and only if $M = p_I$:

$s \in S$; $s := s_0$; (current state of $\mathcal{M}$)
$i \in [0, \alpha + \beta + 2]$; $i := 0$; (current input head position)
$h \in [0, l + 1]$; $h := 0$; (current working head position)
$a \in \{0, 1, \$, \triangleright, \triangleleft\}$; $a := \triangleright$; (current input symbol)
$b \in \{0, 1, \triangleright, \triangleleft\}$; $b := \triangleright$; (current working tape symbol)

`while` $s \notin F$ `loop`

$$a := \begin{cases} \triangleright & \text{if } i = 0 \\ \triangleleft & \text{if } i = \alpha + \beta + 2 \\ \$ & \text{if } i = \alpha + 1 \\ \text{bit}_i(I) & \text{if } i \in [1, \alpha] \\ \text{bit}_{i-\alpha-1}(M) & \text{otherwise} \end{cases}$$

$$b := \begin{cases} \triangleright & \text{if } h = 0 \\ \triangleleft & \text{if } h = l + 1 \\ \text{bit}_h(X) & \text{otherwise} \end{cases}$$

Let $\mu(s, a, b) = (s', \delta_1, \delta_2, b')$.
`if` $h \in [1, l]$ `then` $\text{bit}_h(X) := b'$; `fi`
$s := s'$;
$i := i + \delta_1$;
$h := h + \delta_2$;
`endloop`

Let us describe our SOCA $\mathbb{S}_{\text{prime}}$. The control states Q of $\mathbb{S}_{\text{prime}}$ will contain

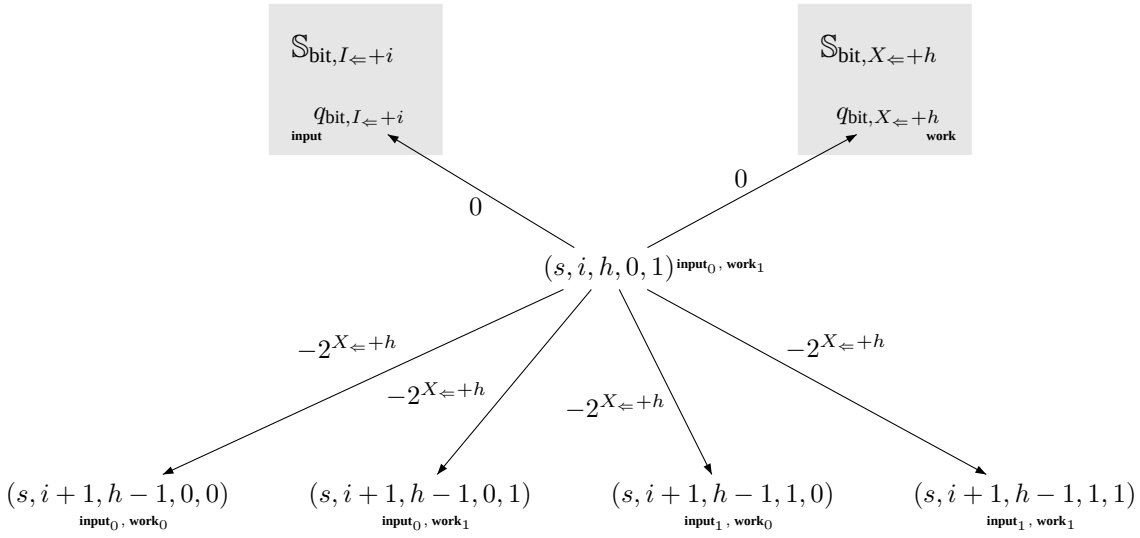
$$S \times [0, \alpha + \beta + 2] \times [0, l + 2] \times \{0, 1, \$, \triangleright, \triangleleft\} \times \{0, 1, \triangleright, \triangleleft\}, \qquad \text{where}$$

the first component corresponds to the variable s, the second component corresponds to the variable i, the third component corresponds to the variable h, the fourth component corresponds to the variable a, and finally the fifth component corresponds to the variable b of the above program. Moreover, our SOCA will contain the SOCA $\mathbb{S}_{\text{bit},I_\Leftarrow}, \mathbb{S}_{\text{bit},I_\Leftarrow+1} \ldots, \mathbb{S}_{\text{bit},I_\Rightarrow}$ and $\mathbb{S}_{\text{bit},X_\Leftarrow}, \mathbb{S}_{\text{bit},X_\Leftarrow+1} \ldots, \mathbb{S}_{\text{bit},X_\Rightarrow}$ in order to test if certain bits in I and X are set correctly. Hence Q contains the control states $q_{\text{bit},I_\Leftarrow}, q_{\text{bit},I_\Leftarrow+1} \ldots, q_{\text{bit},I_\Rightarrow}$ and $q_{\text{bit},X_\Leftarrow}, q_{\text{bit},X_\Leftarrow+1} \ldots, q_{\text{bit},X_\Rightarrow}$ as well.

We will provide the atomic propositions $\mathcal{P} = \{F, \text{input}, \text{work}, \text{input}_0, \text{input}_1, \text{work}_0, \text{work}_1\}$, where

$$\begin{aligned}
Q_F &= \{(s,i,h,a,b) \in Q \mid s \in F\},\\
Q_{\text{input}} &= \{q_{\text{bit},k} \mid k \in [I_{\Leftarrow}, I_{\Rightarrow}]\},\\
Q_{\text{work}} &= \{q_{\text{bit},k} \mid k \in [X_{\Leftarrow}, X_{\Rightarrow}]\},\\
Q_{\text{input}_0} &= \{(s,i,h,a,b) \in Q \mid a = 0\},\\
Q_{\text{input}_1} &= \{(s,i,h,a,b) \in Q \mid a = 1\},\\
Q_{\text{work}_0} &= \{(s,i,h,a,b) \in Q \mid b = 0\},\\
Q_{\text{work}_1} &= \{(s,i,h,a,b) \in Q \mid b = 1\}.
\end{aligned}$$

We do not give all transitions of $\mathbb{S}_{\text{prime}}$, but illustrate some cases by way of example. Let us give the outgoing transitions of the control state $(s,i,h,0,1)$, where $i \in [I_{\Leftarrow}, I_{\Rightarrow}]$ and $h \in [X_{\Leftarrow}, X_{\Rightarrow}]$, i.e. we currently scan the i^{th} bit of I and do not read a marker on our working tape. Moreover, let us assume $\mu(s,0,1) = (s', +1, -1, 0)$, i.e. we change to state s', we move the input head to the right, the working head to the left, and modify the current content on the working tape from 1 to 0. We realize the latter by subtracting $2^{X_{\Leftarrow}+h}$ from the counter, however we allow transitions to states $(s', i+1, h-1, a, b)$, *for each* $a, b \in \{0,1\}$, hence we guess the input tape symbol and the working tape symbol of the successor configuration. The CTL formula φ_{prime} will guarantee that our guessing was correct by accessing the control states $q_{\text{bit},k}$ where $k \in [I_{\Leftarrow}, I_{\Rightarrow}] \cup [M_{\Leftarrow}, M_{\Rightarrow}]$.



The other cases can be dealt with analogously. We put $q_{\text{prime}} = (s_0, 0, 0, \triangleright, \triangleright)$. Our final formula is

$$\varphi_{\text{prime}} \quad = \quad \mathsf{E}\left(\bigwedge_{k\in\{0,1\}} (\text{input}_k \rightarrow \mathsf{EX}(\text{input} \wedge \varphi_{\text{bit},k})) \wedge (\text{work}_k \rightarrow \mathsf{EX}(\text{work} \wedge \varphi_{\text{bit},k}))\right) \mathsf{U}\ F.$$

Then $(T(\mathbb{S}_{\text{prime}}), q_{\text{prime}}(v)) \models \varphi_{\text{prime}}$ if and only if $I\$M \in L(\mathcal{M})$ if and only if $M = p_I$. □

Let us now give an implementation of query Q3.

Lemma 15. *For each bit $b \in \{0,1\}$ there exists a fixed CTL formula $\varphi_{BIT,b}$ such that from x_0 we can compute in logarithmic space some SOCA $\mathbb{S}_{BIT}$ and some control state q_{BIT} such that $(T(\mathbb{S}_{BIT}), q_{BIT}(v)) \models \varphi_{BIT,b}$ if and only if $bit_J(B) = b$.*

Proof (Sketch). By Lemma 14, there is a fixed CTL formula φ_{prime} such that we can compute from x_0 a SOCA $\mathbb{S}_{\text{prime}}$ that allows to check if $M = p_I$. Moreover, by Lemma 12 there is a fixed CTL formula φ_{mod} and a SOCA $\mathbb{S}_{\text{mod}}$ that allows to check whether $B \equiv C \text{mod } M$. In other words, we can deal with the Chinese remainder representation of B, but our goal is to access the the J^{th} bit in the binary representation of B. So let us assume that $\mathcal{R} = \text{CRR}_K(B) = (b_{i,c})_{1 \leq i \leq K, 0 \leq c < p_i}$ is the Chinese remainder representation of B. Note that we do not have $\mathcal{R}$ stored anywhere on the counter. However, the bit strings I and C serve as pointers to access the bit $b_{I,C}$ of $\mathcal{R}$. By Theorem 10, given $\mathcal{R}$ (on-the-fly by our pointers I and C), our bit string J, and the bit b, we can decide if $\text{bit}_J(B) = b$ in logarithmic space. So let $\mathcal{M} = (S, \Sigma, s_0, F, \mu)$ be some $k \cdot \log(m)$ space-bounded Turing machine machine for this, where $k \geq 1$ is some constant. Hence, in order to decide if $\text{bit}_J(B) = b$ we need to simulate $\mathcal{M}$ on input $\langle \mathcal{R}, J, b \rangle$. For this we store the space that $\mathcal{M}$ requires on our reserved sequence of bits Y from ($\bigstar$), hence Y consists of $l = k \cdot \log(|\mathcal{R}Jb|) = O(\beta^2 + \alpha + 1) = O(N^2)$ many bits. The definition of SOCA $\mathbb{S}_{\text{BIT},b}$ works analogously to the construction of $\mathbb{S}_{\text{prime}}$ in the proof of Lemma 14 i.e. we introduce in $\mathbb{S}_{\text{BIT},b}$ control states that remember the current bit of the input head, the current bit of the working head, and the position of the working head. The only difference is that a pointer to the input head cannot be completely stored in the control states. As mentioned above, for this we employ the bit blocks I and C, and also the sequence M for storing the I^{th} prime number. In order to obtain bit $b_{I,C}$, we allow transitions to our SOCA $\mathbb{S}_{\text{mod}}$ (which we can compute in logarithmic space from x_0 by Lemma 12) and checking (i) via the fixed formula φ_{prime} whether $M = p_I$, and then (ii) via the fixed formula φ_{mod} whether $B \equiv C \text{ mod } M$. Pointers to the remaining parts of the input for $\mathcal{M}$, namely J and b, can directly be handled by the control states of $\mathbb{S}_{\text{BIT},b}$, in analogy to the proof of Lemma 14. By way of example we explain the behavior of $\mathbb{S}_{\text{BIT}}$ when the input head of $\mathcal{M}$ currently scans bit $b_{I,C}$ of $\mathcal{R}$ and when getting to the successor configuration requires moving the input head to the left. Then we simply decrement C by 1 which corresponds to subtracting $2^{C \Leftarrow}$ from the counter. If, however, C currently equals 0, we need to decrement I by 1, overwrite M with prime p_{I-1}, and finally overwrite C with $p_{I-1} - 1$. Decrementing I by 1 can simply achieved by subtracting $2^{I \Leftarrow}$ from the counter. Overwriting M with prime p_{I-1} can be done by repeatedly subtracting 1 from M (i.e. subtracting $2^{M \Leftarrow}$ from the counter) until M equals p_{I-1}; checking if $M = p_{I-1}$ can be done via the fixed CTL formula φ_{prime} and the SOCA $\mathbb{S}_{\text{prime}}$ by Lemma 14. The other cases work analogously. □

The following lemma implements query Q4.

Lemma 16. *For each $b \in \{0, 1\}$ there is some fixed CTL formula $\varphi_{U,b}$ such that from x_0 we can compute in logarithmic space some SOCA $\mathbb{S}_U$ and some control state q_U such that $(T(\mathbb{S}_U), q_U(v)) \models \varphi_{U,b}$ if and only if $b = \chi_U(B[1, n], B[n + 1, n + 2^N])$.*

Proof (sketch). The proof is similar to the proof of Lemma 15. Since U is in LOGSPACE, there is some $k \cdot \log(m)$-space bounded Turing machine that decides U, where $k \geq 1$ is some constant. In order to decide if $b = \chi_U(B[1, n], B[n + 1, n + 2^N])$ we need to simulate $\mathcal{M}$ on input $\langle B[1, n], B[n + 1, n + 2^N] \rangle$. We store the space that $\mathcal{M}$ requires on its working tape in the sequence of bits Z from ($\bigstar$), hence Z consists of $k \cdot \log(n + 2^N + 1) = O(N)$ many bits. We use the bit sequence J as a pointer for accessing bits of B. For reading the J^{th} bit of B we make use of the CTL formula $\varphi_{\text{BIT},b}$ and use the SOCA $\mathbb{S}_{\text{BIT}}$ from Lemma 15. The rest of the proof follows along the same lines as the proof of Lemma 14. □

The following lemma implements query Q5 and concludes the EXPSPACE-hardness of the data complexity of CTL on SOCA.

Lemma 17. *There is some fixed CTL formula φ_L such that from x_0 we can compute in logarithmic space some SOCA $\mathbb{S}_L$ and some control state q_L such that $(T(\mathbb{S}_L), q_L(0)) \models \varphi_L$ if and only if $x_0 \in L$.*

Proof. First note that our CTL formula φ_L will be evaluated in state $q_L(0)$. Recall that our bit sequence B has length $n + 2^N + 1$ where B is factorized as $B = xB'$ for some $x \in \{0, 1\}^n$ and some $B' \in \{0, 1\}^{2^N+1}$. The SOCA $\mathbb{S}_L$ and the CTL formula φ_L will mimic the execution of the program from Figure 1. In the bit string x we store the value x_0. The bit string B' represents the variable d of the program, hence we will initialize B' with 0. Note that incrementing B' by 1 corresponds to adding $2^{B \Leftarrow +n}$ to the counter. Thus, checking when d becomes 2^{2^N} for the first time boils down to checking when the $n + 2^N + 1^{\text{st}}$ bit of B

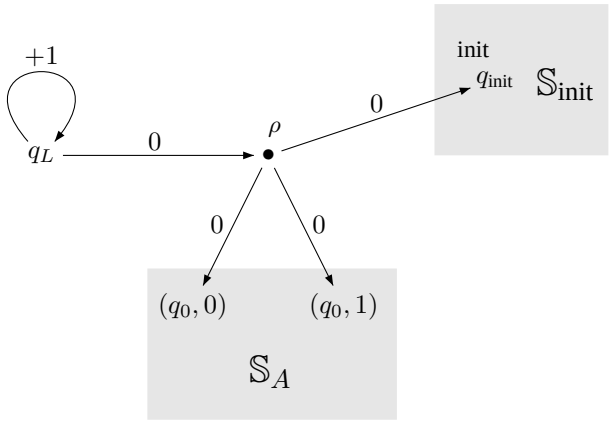
becomes 1 for the first time. By Lemma 15 the CTL formula $\varphi_{\text{BIT},1}$ and the SOCA $\mathbb{S}_{\text{BIT}}$ allow to test if the J^{th} bit of B equals 1. Therefore we store in J the number $n + 2^N + 1$. The following claim tells us that we can test whether we have initialized the counter correctly. Its proof is simple and therefore omitted.

Claim: There is a fixed CTL formula φ_{init} such that from x_0 we can compute in logarithmic space some SOCA $\mathbb{S}_{\text{init}}$ and some control state q_{init} such that $(T(\mathbb{S}_{\text{init}}), q_{\text{init}}(v)) \models \varphi_{\text{init}}$ if and only if $J = n + 2^N + 1$ and $B = x_0 0^{2^N+1}$.

Recall that $A = (Q, \{0,1\}, q_0, \delta, F)$ is the deterministic finite automaton of the program in Figure 1 that needs to be simulated. Let us, before giving $\mathbb{S}_L$, define the auxiliary SOCA $\mathbb{S}_A$ to be connected with the control state q_U of $\mathbb{S}_U$ and with the control state q_{BIT} of $\mathbb{S}_{\text{BIT}}$ along with the additional control states $S = Q \times \{0,1\}$ and the following transitions:

$$(q,b) \xrightarrow{+2^{B} \Leftarrow +n} (\delta(q,b), b'), \qquad (q,b) \xrightarrow{0} q_{\text{BIT}}, \qquad \text{and} \qquad (q,b) \xrightarrow{0} q_U \qquad \text{for each } b, b' \in \{0,1\}.$$

Moreover, $\mathbb{S}_A$ contains the atomic propositions $\{\text{bit}_0, \text{bit}_1, F\}$, where $S_{\text{bit}_b} = Q \times \{b\}$ for each $b \in \{0,1\}$ and $S_F = F \times \{0,1\}$. Before giving φ_L, let us depict our SOCA $\mathbb{S}_L$ which has two additional labels ρ and init:



Let us define the auxiliary formula $\chi = \mathsf{EX}\varphi_{\text{BIT}}$ that allows to test if $B' = 2^{2^N}$. Our final formula is

$$\varphi_L \quad = \quad \mathsf{EF}\left(\rho \wedge \mathsf{EX}(\text{init} \wedge \varphi_{\text{init}}) \wedge \mathsf{E}\left(\neg\chi \wedge \bigwedge_{b \in \{0,1\}} \text{bit}_b \rightarrow \mathsf{EX}\varphi_{U,b}\right) \mathsf{U}\ (\chi \wedge F)\right)$$

We have $x_0 \in L$ if and only if the program from Figure 1 returns `true` if and only if $(T(\mathbb{S}_L), q_L(0)) \models \varphi_L$. □

Theorem 18. *The data and combined complexity of CTL and the modal μ-calculus on SOCA is* EXPSPACE*-complete.*

3.5 Π_1^0-Hardness of the Data Complexity of CTL on POCA

In this section we show that there already exists a fixed CTL formula for which model checking of POCA is Π_1^0-hard. We reduce from the complement of the emptiness problem for *two-counter automata*, which is Σ_1^0-complete [19]. Similar to a SOCA, a *two-counter automaton* is a tuple $\mathbb{A} = (Q, E, \lambda)$, where Q is a finite set of *control states*, $E \subseteq Q \times Q$ is a set of *edges*, however $\lambda : E \to \{\mathsf{zero}_1, \mathsf{zero}_2\} \cup \{\mathsf{add}_j(a) \mid j \in \{1,2\}, a \in \{-1,1\}\}$. The *configuration graph induced by* $\mathbb{A}$ is defined to be $G(\mathbb{A}) = (Q \times \mathbb{N} \times \mathbb{N}, \rightarrow)$, where $(p, i_1, i_2) \rightarrow (q, i_1 + d_1, i_2 + d_2)$ if and only if the following conditions are all satisfied:

- $(p,q) \in E$,
- $\lambda(p,q) = \mathsf{add}_j(a)$ implies $d_j = a$ and $d_{3-j} = 0$ for each $j \in \{1,2\}$, and

– $\lambda(p,q) = \mathsf{zero}_j$ implies $d_j = i_j = 0$ and $d_{3-j} = 0$ for each $j \in \{1,2\}$.

Let us define the following problem.

EMPTINESS FOR TWO-COUNTER AUTOMATA

INPUT: Two-counter automaton $\mathbb{A} = (Q, E, \lambda)$, control states $q_0, q_1 \in Q$.
QUESTION: Are there $m, n \in \mathbb{N}$ such that $(q_0, 0, 0) \to^* (q_1, m, n)$ in $G(\mathbb{A})$?

The idea of our reduction is as follows: Given a two-counter automaton $\mathbb{A}$, we construct a POCA $\mathbb{P}(x)$ with one parameter in such a way that the two counters from $\mathbb{A}$ are encoded into the single counter from $\mathbb{P}(x)$ as follows: for the counter value n of $\mathbb{P}(x)$, we have that $n \bmod x$ encodes the value of the first and $n \operatorname{div} x$ encodes the value of the second counter of $\mathbb{A}$. Hence, testing the first counter for zero corresponds to checking whether $n \equiv 0 \bmod x$ while testing the second counter for zero corresponds to checking whether $n \geq x$. Incrementation (resp. decrementation) on the first counter can be mimicked by adding (resp. subtracting) 1, whereas on counter two this corresponds to adding (resp. subtracting) x. Of course, we need to ensure that we do not overflow. For example, if $n \equiv -1 \bmod x$ and we would simulate to increment the first counter in the above manner, this would correspond to setting the first counter to zero and simultaneously incrementing the second counter. However, if the emptiness problem is solvable, then x can be instantiated with a large enough value such that such an overflow does not occur. Our CTL formula will ensure that we do not overflow. Before we give the reduction from the emptiness problem for two-counter automata, let us introduce some gadgets that we need in our reduction.

Lemma 19. *There exists some fixed CTL formula φ_{test} for which the following holds:*

(1) There exists some fixed POCA $\mathbb{P}_{\not\equiv 0}(x)$ *with some control state $q_{\not\equiv 0}$ such that for each $\sigma : \{x\} \to \mathbb{N}$*

$$(\mathbb{P}^{\sigma}_{\not\equiv 0}, q_{\not\equiv 0}(n)) \models \varphi_{\mathsf{test}} \iff n \not\equiv 0 \text{ mod } \sigma(x).$$

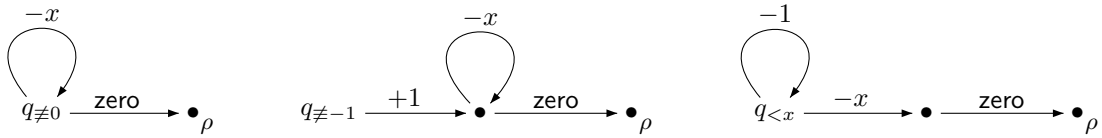
(2) There exists some fixed POCA $\mathbb{P}_{\not\equiv -1}(x)$ *with some control state $q_{\not\equiv -1}$ such that for each $\sigma : \{x\} \to \mathbb{N}$*

$$(\mathbb{P}^{\sigma}_{\not\equiv -1}, q_{\not\equiv -1}(n)) \models \varphi_{\mathsf{test}} \iff n \not\equiv -1 \text{ mod } \sigma(x).$$

(3) There exists some fixed POCA $\mathbb{P}_{<x}(x)$ *with some control state $q_{<x}$ such that for each $\sigma : \{x\} \to \mathbb{N}$*

$$(\mathbb{P}^{\sigma}_{<x}, q_{<x}(n)) \models \varphi_{\mathsf{test}} \iff n < \sigma(x).$$

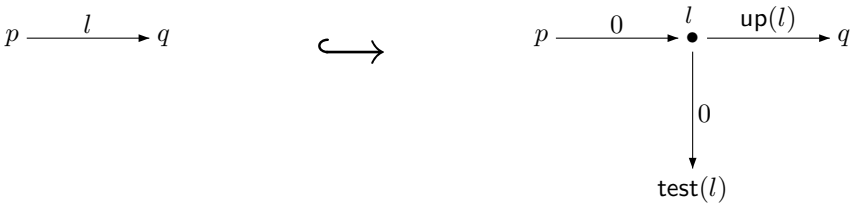
Proof. We choose $\varphi_{\mathsf{test}} = \neg\mathsf{EF}\rho$, where ρ is an atomic proposition. Below we depict $\mathbb{P}_{\not\equiv 0}(x)$, $\mathbb{P}_{\not\equiv -1}(x)$, and $\mathbb{P}_{<x}(x)$, respectively.



Correctness of the lemma is immediate. □

Let us fix some instance of the emptiness problem for two-counter automata: Let $\langle \mathbb{A}, q_0, q_1 \rangle$ be an instance to the emptiness problem, where $\mathbb{A} = (Q, E, \lambda)$ is some two-counter automaton. We will give a fixed CTL formula φ such that from $\langle \mathbb{A}, q_0, q_1 \rangle$ we can construct some POCA $\mathbb{P}(x)$ such that $(q_0, 0, 0) \to^* (q_1, m, n)$ for some $m, n \in \mathbb{N}$ if and only if $(\mathbb{P}^{\sigma}, q_0(0)) \models \varphi$ for some $\sigma : \{x\} \to \mathbb{N}$.

The control states and transitions will of $\mathbb{P}(x)$ can be described by the following graph transformation rule that maps each pair of control states $(p,q) \in E$ of $\mathbb{A}$ with $\lambda(p,q) = l$ into corresponding control states in $\mathbb{P}(x)$, by possibly accessing the different POCA of Lemma 19.



where

$$\mathsf{test}(l) \quad = \quad \begin{cases} (\mathbb{P}_{\not\equiv 0}(x), q_{\not\equiv 0}) & \text{if} \quad l \in \{\mathsf{add}_1(-1), \mathsf{zero}_1\} \\ (\mathbb{P}_{\not\equiv -1}(x), q_{\not\equiv -1}) & \text{if} \quad l = \mathsf{add}_1(+1) \\ (\mathbb{P}_{<x}(x), q_{<x}) & \text{if} \quad l = \mathsf{zero}_2, \end{cases}$$

and where

$$\mathsf{up}(l) \quad = \quad \begin{cases} a & \text{if} \quad l = \mathsf{add}_1(a) \\ \circ x & \text{if} \quad l = \mathsf{add}_2(\circ 1) \text{ for some } \circ \in \{+, -\} \\ 0 & \text{otherwise.} \end{cases}$$

Let us moreover ensure that every control state of the kind q_τ with $\tau \in \{\not\equiv 0, \not\equiv -1, \geq x\}$ has the label ρ_τ. Before we give our final formula, let us introduce the following constraint formulas that guarantee that an overflow never occurs:

$$\begin{aligned} \psi_1 &= \mathsf{add}_1(+1) \longrightarrow \mathsf{EX}(\rho_{\not\equiv -1} \wedge \varphi_{\mathsf{test}}) \\ \psi_2 &= \mathsf{add}_1(-1) \longrightarrow \mathsf{EX}(\rho_{\not\equiv 0} \wedge \varphi_{\mathsf{test}}) \\ \psi_3 &= \mathsf{zero}_2 \longrightarrow \mathsf{EX}(\rho_{<x} \wedge \varphi_{\mathsf{test}}) \\ \psi_4 &= \mathsf{zero}_1 \longrightarrow \mathsf{EX}(\rho_{\not\equiv 0} \wedge \neg\varphi_{\mathsf{test}}) \end{aligned}$$

Note that we do not need any constraint when changing the second counter. Let us introduce a label f at control state q_1. Our final CTL formula φ is

$$\varphi \quad = \quad \mathsf{E}\left(\bigwedge_{i \in [4]} \psi_i \,\mathsf{U}\, f \right).$$

We have that $(q_0, 0, 0) \to^* (q_1, m, n)$ for some $m, n \in \mathbb{N}$ in $G(\mathbb{A})$ if and only if $(T(\mathbb{P}^\sigma(x)), q_0(0)) \models \varphi$ for some $\sigma : \{x\} \to \mathbb{N}$. We obtain the following theorem.

Theorem 20. *The data and combined complexity of CTL and the modal μ-calculus on POCA is Π_1^0-complete.*

4 LTL Model Checking

Formulas of LTL are inductively defined according to the following grammar, where ρ ranges over $\mathcal{P}$:

$$\varphi \quad ::= \quad \rho \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \varphi$$

The Boolean abbreviations and the formula $\mathtt{tt}$ are defined in the same way as in CTL. The *finally modality* $\mathsf{F}\varphi$ is an abbreviation for $\mathtt{tt}\mathsf{U}\varphi$ and the *globally modality* $\mathsf{G}\varphi$ abbreviates $\neg\mathsf{F}\neg\varphi$.

The semantics of LTL is given in terms of infinite paths in a transition system. Let $T = (S, \{S_\rho \mid \rho \in \mathcal{P}\}, \to)$ be a transition system, let $\pi : s_0 \to s_1 \to \cdots$ be an infinite path in T, and let φ be an LTL formula, we define $(T, \pi) \models \varphi$ by induction of the structure of φ, where ρ ranges over $\mathcal{P}$:

$$\begin{aligned} (T, \pi) \models \rho &\iff \pi(0) \in S_\rho \\ (T, \pi) \models \neg\varphi &\iff (T, \pi) \not\models \varphi \\ (T, \pi) \models \varphi_1 \wedge \varphi_2 &\iff (T, \pi) \models \varphi_1 \text{ and } (T, \pi) \models \varphi_2 \\ (T, \pi) \models \mathsf{X}\varphi &\iff (T, \pi^1) \models \varphi \\ (T, \pi) \models \varphi_1 \mathsf{U} \varphi_2 &\iff \exists j \geq 0 : (T, \pi^j) \models \varphi_2 \text{ and } \forall 0 \leq i < j : (T, \pi^i) \models \varphi_1 \end{aligned}$$

Let us now define the LTL model checking problem on SOCA and POCA.

LTL MODEL CHECKING ON SOCA

INPUT: SOCA $\mathbb{S} = (Q, \{Q_\rho \mid \rho \in \mathcal{P}\}, E, \lambda)$, $q \in Q$, $n \in \mathbb{N}$ in binary, and an LTL formula φ.

QUESTION: $(T(\mathbb{S}), \pi) \models \varphi$ for every infinite path π with $\pi(0) = q(n)$?

LTL MODEL CHECKING ON POCA

INPUT: POCA $\mathbb{P}(X) = (Q, \{Q_\rho \mid \rho \in \mathcal{P}\}, E, \lambda)$, $q \in Q$, $n \in \mathbb{N}$ in binary, and an LTL formula φ.
QUESTION: $(T(\mathbb{P}^\sigma), \pi) \models \varphi$ for every $\sigma : X \to \mathbb{N}$ and every infinite path π with $\pi(0) = q(n)$?

4.1 Upper bounds

A standard approach to LTL model checking is the automata-based approach, in which systems are modeled as non-deterministic Büchi automata (NBA). Given an NBA A and an LTL formula φ, the idea is to translate φ into an NBA $A_{\neg\varphi}$ such that $A_{\neg\varphi}$ accepts all words that violate φ. Then, by checking for emptiness of the product automaton $A' = A \times A_{\neg\varphi}$, it can be decided whether or not A generates traces that violate φ. Emptiness can be decided by checking for recurrent reachability of a control state in the transition system induced by A', which can be performed non-deterministically in space logarithmically in the size of A'. Vardi and Wolper showed in [22] that the size of $A_{\neg\varphi}$ is $2^{O(|\varphi|)}$, which yields that checking for emptiness of A' is in PSPACE. The PSPACE lower bound for finite state model checking was originally shown in [18] and hence carries over to SOCA.

Constructing the product automaton of two NBA is a standard technique in model checking, see e.g. [2] for a detailed treatment of this. Moreover, this construction can be adapted in a straight-forward way into the setting of POCA by equipping a POCA with a set of final states and defining emptiness with respect to Büchi acceptance condition in the standard way. For brevity, we do not give any further details. We then have that given a POCA $\mathbb{P}$, an LTL formula φ, a state q and $n \in \mathbb{N}$ we can construct a POCA $\mathbb{P}'$ as the product of $\mathbb{P}$ and $A_{\neg\varphi}$ such that for all assignments σ we have that $\mathbb{P}'$ is empty with respect to Büchi acceptence condition if and only if $(T(\mathbb{P}^\sigma), q(n)) \models \varphi$. Moreover, the size of $\mathbb{P}'$ is $2^{O(|\varphi|)}|\mathbb{P}|$.

It has been shown in [12] that checking emptiness with respect to Büchi acceptance condition is coNP-complete for both SOCA and POCA, and in [9] that it is NL-complete for OCA. We use these results in order to establish upper bounds for our LTL model checking problems. Let us first consider the case of a fixed formula φ. In the terminology of the previous paragraph, $|\mathbb{P}'| = O(|\mathbb{P}|)$ and hence the data complexity of LTL on SOCA and POCA is in coNP. The matching coNP lower bound of the data complexity of LTL on SOCA follows from the NP-hardness of the reachability problem for SOCA.

Proposition 21. *The data complexity of LTL model checking on SOCA and POCA is* coNP*-complete.*

As seen above, if both $\mathbb{P}$ and the formula φ are part of the input then $|\mathbb{P}'| = 2^{O(|\varphi)} \cdot |\mathbb{P}|$, and hence coNEXP is an upper bound for the combined complexity of LTL model checking on both SOCA and POCA. However, we can improve this upper bound in the case of SOCA. Given a SOCA $\mathbb{S}$, let m be the absolute value of the maximal increment or decrement performed on the transitions in $\mathbb{S}$ and let $\mathbb{S}'$ be the product automaton of $\mathbb{S}$ and $A_{\neg\varphi}$. Let $\mathbb{S}''$ be obtained from $\mathbb{S}'$ by replacing every transition labeled with z with a sequence of fresh transitions and control locations of length z, where each transition is labeled with $+1$ respectively -1, depending on the sign of z. We conclude that $|\mathbb{S}''| = m \cdot |\mathbb{S}'| = m \cdot 2^{O(|\varphi|)} \cdot |\mathbb{S}|$. Combining the result from [9] together with the fact that LTL model checking of non-deterministic finite state automata is PSPACE-hard [18], we obtain the following proposition.

Proposition 22. *The combined complexity of LTL on SOCA is* PSPACE*-complete.*

It remains to prove that the combined complexity of LTL model checking on POCA is coNEXP-hard, which we will show in the next section.

4.2 Combined Complexity of LTL on POCA

We are now going to show coNEXP-hardness of LTL model checking on POCA via a reduction from (the complement of) Succinct 3-SAT, which is a NEXP-complete problem [20]. An input of Succinct 3-SAT is given by a Boolean circuit $\mathbb{C}$ that encodes a Boolean formula $\psi_\mathbb{C}$ in 3-CNF, i.e. $\psi_\mathbb{C} = \bigwedge_{j \in [0,M]} (\ell_1^j \vee \ell_2^j \vee \ell_3^j)$ for some $M = O(2^{|\mathbb{C}|})$, and $\psi_\mathbb{C}$ is free over Boolean variables $y_1, \ldots, y_N$ for some $N = O(2^{|\mathbb{C}|})$. Let

$j \in [0, M]$ be the index of a clause encoded in *binary* and $k \in \{1, 2, 3\}$. On input $(j \cdot k)$, the output of $\mathbb{C}$ is $(i \cdot b)$, where $i \in [N]$ is the index of the Boolean variable that appears in literal ℓ_k^j, and where $b = 0$ when ℓ_k^j is negative and $b = 1$ when ℓ_k^j is positive.

SUCCINCT 3-SAT

INPUT: Boolean circuit $\mathbb{C}$
QUESTION: Is $\psi_{\mathbb{C}}$ satisfiable?

In order to establish coNEXP-hardness for the combined complexity of LTL model checking, given an input $\mathbb{C}$ of Succinct 3-SAT, we construct a POCA $\mathbb{P}(x)$ and an LTL formula φ such that $\psi_{\mathbb{C}}$ is satisfiable if and only if there is an assignment σ such that $(T(\mathbb{P}^\sigma), q_{\mathsf{start}}(0)) \models \varphi$ for some distinguished state q_{start} of $\mathbb{P}(x)$.

First, let us provide a suitable encoding of truth assignments by natural numbers. The encoding we use has also been employed for establishing lower bounds for model checking OCA [16]. Recall that p_i denotes the i^{th} prime number. Every natural number x defines a truth assignment $\nu_x : \{y_1, \ldots, y_N\} \to \{0, 1\}$ such that $\nu_x(y_i) = 1$ iff p_i divides x. By the Prime Number Theorem, $p_N = O(N \log N)$ and hence $O(|\mathbb{C}|)$ bits are sufficient to represent p_N. Of course, since we need exponentially many prime numbers they cannot be hard-wired into $\mathbb{P}(x)$.

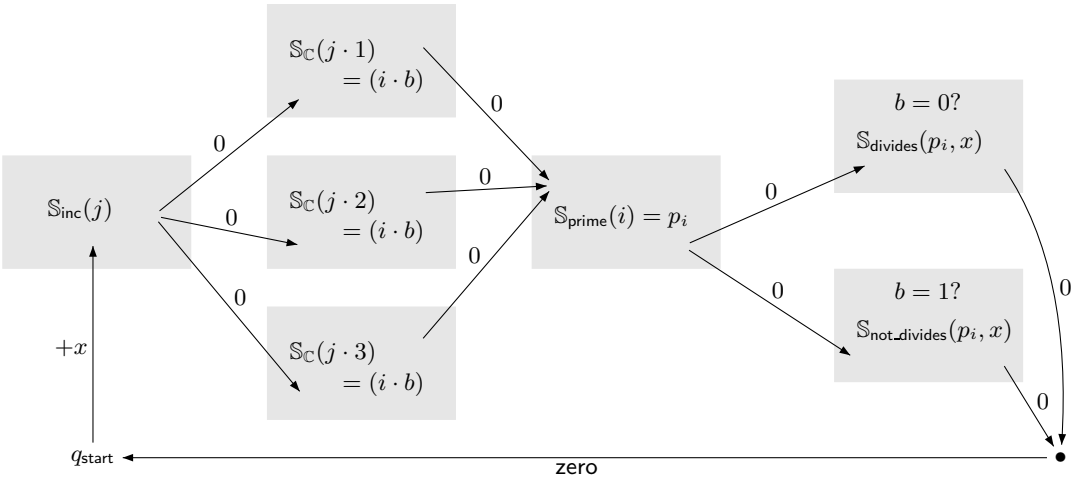


Fig. 3. High-level description of the automaton $\mathbb{P}(x)$ used for the reduction.

Let us now take a look at $\mathbb{P}(x)$, which is sketched in Figure 3. It uses one parameter x and employs several gadgets. The only gadgets manipulating the counter are $\mathbb{S}_{\mathsf{divides}}$ and $\mathbb{S}_{\mathsf{not\_divides}}$. The remaining gadgets are designed such that they communicate via designated propositional variables, and not as in Section 3.4 with the help of the counter. First, $\mathbb{P}(x)$ loads the value of the parameter x on the counter. Think of x encoding a truth assignment of $\psi_{\mathbb{C}}$. Next, $\mathbb{P}(x)$ traverses through $\mathbb{S}_{\mathsf{inc}}$, which initially chooses an arbitrary index j identifying a clause of $\psi_{\mathbb{C}}$. Every time $\mathbb{S}_{\mathsf{inc}}$ is traversed afterwards, it increments j modulo $M + 1$ and hereby moves on to the next clause. Now $\mathbb{P}(x)$ branches non-deterministically into a gadget $\mathbb{S}_{\mathbb{C}}$ in order to compute $(i \cdot b)$ from $\mathbb{C}$ on input $(j \cdot 1), (j \cdot 2)$, resp. $(j \cdot 3)$. The index i is then used as input to a gadget $\mathbb{S}_{\mathsf{prime}}$, which computes p_i. Then if $b = 0$, it is checked in $\mathbb{S}_{\mathsf{not\_divides}}$ that p_i does not divide the value of x, and likewise in $\mathbb{S}_{\mathsf{divides}}$ that p_i divides the value of x if $b = 1$. For checking the latter, the counter needs to be modified. After the checks have been finished, we restore the value x on the counter and the process continues with clause $j + 1$ mod $M + 1$.

It remains to show how the gadgets and the communication between them can be realized. The first observation is that the computations of $\mathbb{S}_{\mathsf{inc}}$, $\mathbb{S}_{\mathbb{C}}$ and $\mathbb{S}_{\mathsf{prime}}$ can be realized by space bounded DTM using no more than $p(|\mathbb{C}|)$ tape cells for their in- and output tape for some polynomial p that is fixed once $\mathbb{C}$ is provided. Indeed, it is easily seen that incrementing modulo $M + 1$, evaluating $\mathbb{C}$ and computing the i^{th}

prime number can be done by such a DTM. Thus, we now show how from a generic DTM $\mathcal{M}$, we can construct in logarithmic space a SOCA $\mathbb{S}_{\mathcal{M}}$ and some LTL formulas that mimic computations of $\mathcal{M}$ on traces of $\mathbb{S}_{\mathcal{M}}$.

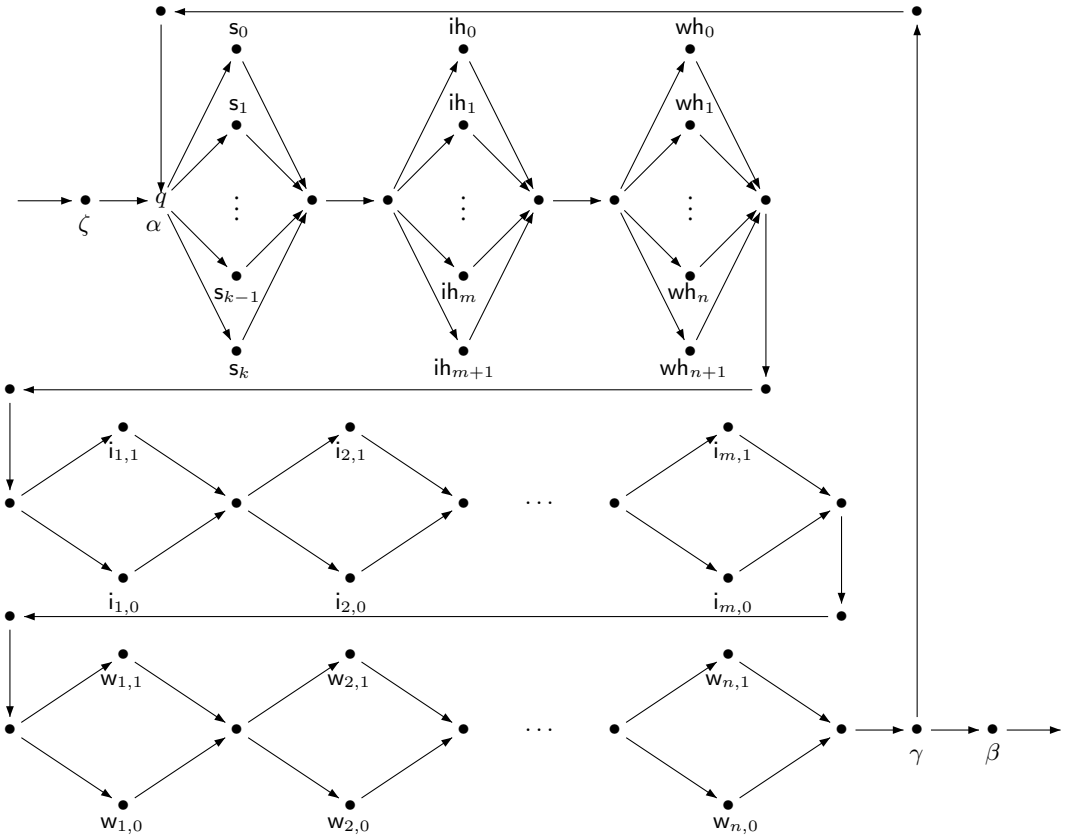


Fig. 4. SOCA $\mathbb{S}_{\mathcal{M}}$ for the simulation of space bounded Turing machines.

Let $\mathcal{M} = (S, \Sigma, s_0, F, \mu)$ be a DTM with a fixed input tape with m tape cells, and n working tape cells, and let $S = \{s_0, \ldots, s_k\}$. Figure 4 shows the SOCA $\mathbb{S}_{\mathcal{M}}$ that we use for the simulation of $\mathcal{M}$. All transitions of $\mathbb{S}_{\mathcal{M}}$ are implicitly labelled with 0. A simulation starts when $\mathbb{S}_{\mathcal{M}}$ is entered at the location labeled with ζ and is finished when the location labeled with β is reached.

The sequence of propositions occurring on a trace starting from and ending in q encodes a configuration of $\mathcal{M}$. In detail, s_i indicates that $\mathcal{M}$ is in state s_i; ih_i that the input tape head scans cell i; wh_i that the working tape head scans cell i; $\mathsf{i}_{i,b}$ that the i^{th} bit of the input tape is set to b; and $\mathsf{w}_{i,b}$ that the i^{th} bit of the working tape is set to b, where i is in the respective range and $b \in \{0, 1\}$.

Let us introduce some LTL formulas that allow for testing properties of the current configuration. Think of all of them as being evaluated in q. The formula $\mathsf{state}_i = \mathsf{X}\mathsf{s}_i$ for each $i \in [0, k]$ expresses that the current state is s_i. By the formula $\mathsf{inhead}_i = \mathsf{XXXX}\mathsf{ih}_i$ we express that the input head is at position i, where $i \in [m]$. Similarly, define the formulas wohead_i, $\mathsf{work}_{j,b}$, and $\mathsf{input}_{i,b}$ for expressing that the working head is at position i, that the i^{th} bit of the input tape is b, and that the j^{th} bit of the working tape is b, respectively, where $i \in [m]$,$j \in [n]$, and $b \in \{0, 1\}$.

The LTL formula below, assumed to be evaluated in α, ensures that the transition function is correctly encoded into traces of $\mathbb{S}_{\mathcal{M}}$ for states $s \in S \setminus F$ whenever the input respectively working tape head does not scan a start ($\triangleright$) respectively end marker ($\triangleleft$):

$$\bigwedge_{s_l \in S \setminus F} \bigwedge_{i \in [m]} \bigwedge_{j \in [n]} \bigwedge_{b_1, b_2 \in \{0,1\}} \mathsf{state}_l \wedge \mathsf{inhead}_i \wedge \mathsf{input}_{i,b_1} \wedge \mathsf{wohead}_j \wedge \mathsf{work}_{j,b_2} \longrightarrow$$

$$\longrightarrow \left((\mathsf{X}(\neg\alpha \wedge \neg\beta)\mathsf{U}(\alpha \wedge \mathsf{succ}(s_l, i, j, b_1, b_2)) \wedge \bigwedge_{j' \neq j} \bigwedge_{b \in \{0,1\}} (\mathsf{work}_{j',b} \leftrightarrow (\mathsf{X}\neg\alpha\mathsf{U}(\alpha \wedge \mathsf{work}_{j',b}))) \right).$$

Here, whenever $\mu(s_l, b_1, b_2) = (s_h, d_1, d_2, b)$, the formula

$$\mathsf{succ}(s_l, i, j, b_1, b_2) \quad = \quad \mathsf{state}_h \wedge \mathsf{inhead}_{i+d_1} \wedge \mathsf{wohead}_{j+d_2} \wedge \mathsf{work}_{j,b}$$

guarantees that the correct bit is "written" to the working tape and that the state, the input head position, and the working tape position of the next configuration seen indeed match the successor configuration. A similar formula can be constructed for the case when one or both of the input or working heads point to a start respectively end marker. Once we have reached a final state $s_i \in F$, we require that $\mathbb{S}_{\mathcal{M}}$ is left which is expressed by the following formula when evaluated in q:

$$\bigwedge_{s_i \in F} \mathsf{state}_i \longrightarrow (\neg\alpha\mathsf{U}\beta).$$

It is now easily seen that we can construct a formula Φ that is derived from a conjunction of the formulas from above such that the formula $\mathsf{G}(\alpha \rightarrow \Phi)$ constraints paths through $\mathbb{S}_{\mathcal{M}}$ in such a way that their traces yield the encoding of a valid computation of $\mathcal{M}$.

Let us now address towards ensuring that once we enter $\mathbb{S}_{\mathcal{M}}$ we initially traverse it in such a way that the trace corresponds to an initial configuration of $\mathcal{M}$. The formula

$$\mathsf{G}\left(\zeta \longrightarrow \mathsf{X}(\mathsf{state}_0 \wedge \mathsf{inhead}_1 \wedge \mathsf{wohead}_1 \wedge \bigwedge_{1 \leq j \leq n} \mathsf{work}_{j,0}) \right)$$

makes sure that the heads of the input and working tape point to the first tape cell, that the working tape is filled with 0s and that we are in the initial state. In case the input tape can be initialized with an arbitrary content, we are done. Otherwise, suppose that we want to transfer the first j bits of the output of a TM $\mathcal{M}'$ from its corresponding SOCA $\mathbb{S}_{\mathcal{M}'}$ to the input of $\mathbb{S}_{\mathcal{M}}$. For $b \in \{0,1\}$, let $\bar{b} = 0$ if $b = 1$ and $\bar{b} = 1$ otherwise, and suppose that all atomic propositions are primed in $\mathbb{S}_{\mathcal{M}'}$. The formula

$$\bigwedge_{\substack{1 \leq i \leq j \\ b \in \{0,1\}}} \mathsf{G}((\mathsf{w}'_{i,b} \wedge (\neg\alpha'\mathsf{U}\beta')) \longrightarrow (\neg\mathsf{i}_{i,\bar{b}}\mathsf{U}\gamma)$$

guarantees that we traverse through the first j bits of the component of $\mathbb{S}_{\mathcal{M}}$ representing the input tape of $\mathcal{M}$ in the same way as we traverse the first j bits of the working tape component of $\mathcal{M}'$ in $\mathbb{S}_{\mathcal{M}'}$ when a computation has finished. Coming back to Figure 3, we have thus seen how the SOCA $\mathbb{S}_{\mathsf{inc}}, \mathbb{S}_{\mathbb{C}}$ and $\mathbb{S}_{\mathsf{prime}}$ and the communication between them can be realized. The only major question left open is how we can perform a divisibility respectively non-divisibility test of the counter value with a prime number computed in $\mathbb{S}_{\mathsf{prime}}$. For this, let us consider the SOCA $\mathbb{S}_{\mathsf{divides}}$ from Figure 5.

One cycle through $\mathbb{S}_{\mathsf{divides}}$ subtracts some natural number of bit length $l+1$ from the counter. However, in order to test for divisibility we need to make sure that we remain on the same path in every cycle. In the CTL setting, this problem was resolved by branching into the additional SOCA $\mathbb{S}_{\mathsf{bit}}$. In contrast, in LTL we cannot branch, but use the propositions $\rho_{j,b_j}, j \in [0,l], b \in \{0,1\}$ in order to stay on precisely one path in every cycle. Assuming that the number p for which we want to test for divisibility with the current counter value is encoded as a sequence of propositions w_{j,b_j} of some SOCA $\mathbb{S}_{\mathcal{M}}$, the subsequent formula enforces that we always subtract p in cycles of $\mathbb{S}_{\mathsf{divides}}$:

$$\mathsf{G}\left(\bigwedge_{\substack{0 \leq j \leq l \\ b_j \in \{0,1\}}} ((\mathsf{w}_{j,b_j} \wedge (\neg\alpha\mathsf{U}\beta)) \longrightarrow (\neg\rho_{j,\bar{b_j}}\mathsf{U}\xi)) \right).$$

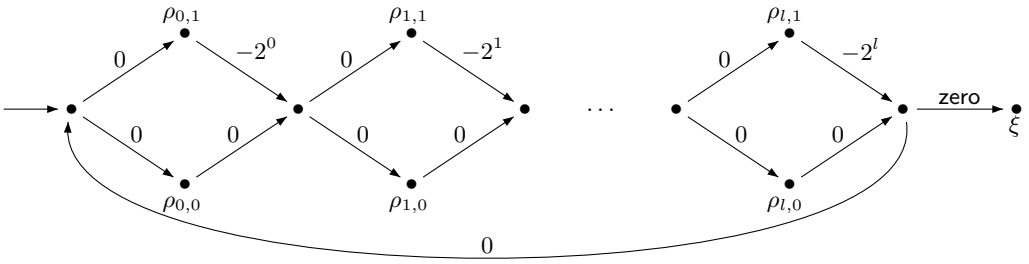


Fig. 5. The SOCA $\mathbb{S}_{\mathsf{divides}}$ for testing the counter for divisibility with some natural number of bit length $l + 1$.

It is straight forward to derive a similar SOCA $\mathbb{S}_{\mathsf{not\_divides}}$ and an appropriate LTL formula for testing non-divisibility of the counter value with a previously computed prime number. Finally, we can also adopt these techniques in order to correctly handle the branching on b performed in Figure 3.

In summary, by taking the disjoint union of all the gadgets from Figure 3, their appendent LTL formula that we described in this section, connecting the gadgets correctly and taking the conjunction of the relevant LTL formulas, we can construct $\mathbb{P}(x)$ and an LTL formula φ in logarithmic space such that there is an assignment σ assigning a natural number to x such that $(T(\mathbb{P}^\sigma), q_{\mathsf{start}}(0)) \not\models \varphi$ if and only if $\psi_{\mathbb{C}}$ given by an input $\mathbb{C}$ of Succinct 3-SAT is satisfiable.

Theorem 23. *The combined complexity of LTL model checking on POCA is* coNEXP*-complete.*

5 Conclusion

In this paper, we have settled the computational complexity of model checking CTL and LTL on SOCA and POCA with respect to data and combined complexity. Our proofs for providing lower bounds have introduced some involved concepts and techniques, which we believe may be of independent interest for providing lower bounds for decision problems in the verification of infinite state systems.

An interesting aspect of future work could be the consideration of *synthesis problems* for POCA. Given a POCA $\mathbb{P}(X)$ and an LTL formula φ, a natural question to pose is whether there *exists an assignment* σ such that $(T(\mathbb{P}^\sigma), \pi) \models \varphi$ for all infinite paths π starting in some state of $T(\mathbb{P}^\sigma)$. For CTL, such a problem is undecidable by Theorem 18, but we claim that it might be decidable for LTL. Moreover, the CTL fragment EF seems to be a good candidate of a branching-time logic for which model checking on POCA could be decidable, but this remains subject to further investigations.

References

1. A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.
2. Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.
3. A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. In *CAV*, volume 4144 of *LNCS*. Springer, 2006.
4. M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. In *Proc. ICALP'06*, volume 4052 of *LNCS*. Springer, 2006.
5. Jin-Yi Cai and Merrick Furst. PSPACE survives constant-width bottlenecks. *International Journal of Foundations of Computer Science*, 2(1):67–76, 1991.
6. Cristiana Chitic and Daniela Rosu. On validation of xml streams using finite state machines. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 85–90, New York, NY, USA, 2004. ACM.
7. Andrew Chiu, George Davida, and Bruce Litow. Division in logspace-uniform NC^1. *Theoretical Informatics and Applications. Informatique Théorique et Applications*, 35(3):259–275, 2001.
8. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *Proc. CAV'98*, volume 1427 of *LNCS*. Springer, 1998.

9. Stéphane Demri and Régis Gascon. The effects of bounding syntactic resources on Presburger LTL. *Journal of Logic and Computation*, (6):1541–1575, December.
10. Stefan Göller and Markus Lohrey. Branchning-time model checking of one-counter processes. Technical report, arXiv.org. `http://arxiv.org/abs/0909.1102`.
11. Stefan Göller and Markus Lohrey. Branchning-time model checking of one-counter processes. In *STACS 2010*. IFIB Schloss Dagstuhl, 2010. to appear.
12. Christoph Haase, Stephan Kreutzer, Joel Ouaknine, and James Worrell. Reachability in parametric one-counter automata. 2010. Submitted. Available via: http://www.comlab.ox.ac.uk/files/2833/iandc.pdf.
13. Ulrich Hertrampf, Clemens Lautemann, Thomas Schwentick, Heribert Vollmer, and Klaus W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings of the Eighth Annual Structure in Complexity Theory Conference*, pages 200–207. IEEE Computer Society Press, 1993.
14. O. H. Ibarra, T. Jiang, N. Trân, and H. Wang. New decidability results concerning two-way counter machines and applications. In *ICALP*, volume 700 of *LNCS*. Springer, 1993.
15. Oscar H. Ibarra and Zhe Dang. On the solvability of a class of diophantine equations and applications. *Theor. Comput. Sci.*, 352(1):342–346, 2006.
16. P. Jančar, A. Kučera, F. Moller, and Z. Sawa. DP lower bounds for equivalence-checking and model-checking of one-counter automata. *Information Computation*, 188(1):1–19, 2004.
17. J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *Proc. ATVA'05*, volume 3707 of *LNCS*. Springer, 2005.
18. Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL '85: Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 97–107, New York, NY, USA, 1985. ACM.
19. Marvin L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics. Second Series*, 74:437–455, 1961.
20. C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
21. Olivier Serre. Parity games played on transition graphs of one-counter processes. In L. Aceto and A. Ingólfsdóttir, editors, *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2006)*, number 3921 in Lecture Notes in Computer Science. Springer, 2006.
22. Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Symposium on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.
23. Heribert Vollmer. A generalized quantifier concept in computational complexity theory. Technical report, arXiv.org, 1998. http://arxiv.org/abs/cs.CC/9809115.