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, 18]), and on the types of computations considered (such as reversal-boundedness [16]). 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 [2, 18] or even in *parametric* form, i.e., whose precise values depend on parameters [3, 15]. We refer to such counter automata as *succinct* and *parametric* resp., the former being viewed as a subclass of the latter. Natural applications of such counter machines include the modelling of resource-bounded processes, programs with lists, recursive or multi-threaded programs, and XML query evaluation; see, e.g., [2, 6, 16].

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, 13, 15].

		SOCA	POCA
CTL, μ-calculus	**data**	EXPSPACE-complete	Π^0_1-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.

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 [13]. 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 [12].

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 [12], this does not immediately provide an EXPSPACE lower bound. In a subsequent delicate stage of the proof, we show how to partition the counter in order simultaneously to perform PSPACE computations in the counter and manipulate numbers of exponential size in a SOCA of polynomial size.

For reasons of space, we have had to abbreviate or omit a number of proofs; full details can however be found in the technical report [10].

2 Preliminaries

By $\mathbb{Z}$ we denote the *integers* and by $\mathbb{N} = \{0, 1, 2, \ldots\}$ 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 $\mathrm{bit}_i(n)$ denote the i^{th} least significant bit of the binary representation of n. Hence $n = \sum_{i \in \mathbb{N}} 2^i \cdot \mathrm{bit}_i(n)$. By $\mathrm{bin}_m(n) = \mathrm{bit}_0(n) \cdots \mathrm{bit}_{m-1}(n)$ we denote the first m least significant bits written from *left to right*. Let p_i denote the i^{th} prime number for each

$i \geq 1$. We define $\log(n) = \min\{i \geq 1 \mid 2^i > n\}$, i.e. $\log(n)$ denotes the number of bits that are needed to represent n in binary. 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 $v(i) = v[i,i]$. For the rest of the paper, we fix a countable set of *atomic propositions* $\mathcal{P}$. A *transition system* is a tuple $T = (S, \{S_\rho \mid \rho \in \mathcal{P}\}, \to)$, 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 $\to \subseteq S \times S$ is a set of *transitions*. We prefer to use the infix notation $s_1 \to s_2$ instead of $(s_1, s_2) \in \to$. An *infinite path* is an infinite sequence $\pi = s_0 \to s_1 \to \cdots$. For each infinite path $\pi = s_0 \to s_1 \to \cdots$ and each $i \in \mathbb{N}$, we denote by π^i the suffix $s_i \to s_{i+1} \to \cdots$ and by $\pi(i)$ the state s_i. A *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 $\lambda : E \to \mathbb{Z} \cup \{\mathsf{zero}\}$. We call a SOCA $\mathbb{S}$ with $\lambda : E \to \{-1, 0, 1\} \cup \{\mathsf{zero}\}$ a *unary one-counter automaton (OCA)*. A *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, X is a finite set of *parameters over the naturals*, and $\lambda : E \to (\mathbb{Z} \cup \{\mathsf{zero}\} \cup \{-x, +x \mid x \in X\})$. For each assignment $\sigma : X \to \mathbb{N}$ the induced SOCA is defined as $\mathbb{P}^\sigma = (Q, E, \lambda')$ where $\lambda'(e) = \sigma(x)$ if $\lambda(e) = x$, $\lambda'(e) = -\sigma(x)$ if $\lambda(e) = -x$, and $\lambda'(e) = \lambda(e)$ otherwise. If $X = \{x\}$ we also write $\mathbb{P}(x)$ instead of $\mathbb{P}(X)$. 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 $T(\mathbb{S}) = (Q \times \mathbb{N}, \{Q_\rho \times \mathbb{N} \mid \rho \in \mathcal{P}\}, \to)$, where for each $q_1, q_2 \in Q$ and each $n_1, n_2 \in \mathbb{N}$ we have $q_1(n_1) \to q_2(n_2)$ iff either $\lambda(q_1, q_2) = n_2 - n_1$, or both $n_1 = n_2 = 0$ and $\lambda(q_1, q_2) = \mathsf{zero}$.

3 CTL Model Checking

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

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

Given a transition system $T = (S, \{S_\rho \mid \rho \in \mathcal{P}\}, \to)$, a state $s \in S$, and some CTL formula φ, define $(T, s) \models \varphi$ by induction on the structure of φ as follows:

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

Subsequently we use the standard abbreviations for disjunction, and implication. Moreover, we define $\mathtt{tt} = \rho \vee \neg\rho$ for some $\rho \in \mathcal{P}$ and $\mathsf{EF}\varphi = \mathsf{E}\ (\mathtt{tt}\ \mathsf{U}\varphi)$. Let us define the

CTL model checking problem on SOCA and POCA resp.

CTL MODEL CHECKING ON SOCA

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

CTL MODEL CHECKING ON POCA

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

3.1 Upper Bounds

Due to space restrictions we do not formally introduce the modal μ-calculus and refer to [1] for more details instead. In [21] Serre showed that the combined complexity of the modal μ-calculus on OCA is in PSPACE. Since every SOCA can be transformed into an OCA of exponential size, and since each CTL formula can be translated into an alternation-free μ-calculus formula with a linear blowup, the following proposition is immediate by adjusting the resulting μ-calculus formula appropriately. Moreover, this immediately implies the containment in the arithmetic hierarchy of the combined complexity of the modal μ-calculus and CTL on POCA.

Proposition 1. *For the modal μ-calculus and CTL the combined complexity on SOCA is in* EXPSPACE*, whereas it is in Π_1^0 on POCA.*

3.2 Hardness of the Data Complexity of CTL on SOCA

Before we prove EXPSPACE-hardness of the data complexity of CTL on SOCA, we introduce some notions and results from complexity theory. Given a language $L \subseteq \Sigma^*$, let $\chi_L : \Sigma^* \to \{0,1\}$ denote the *characteristic function of* L. We define the *lexicographic order on n-bit strings* by $x \preceq_n y$ if and only if $\text{bin}_n(x) \leq \text{bin}_n(y)$, e.g. $011 \preceq_3 101$. We say 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 $U \in \mathcal{C}$ s.t. for all $x \in \{0,1\}^n$

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

where the bit strings on the right-hand side of the concatenation symbol are enumerated in lexicographic order. This definition is a padded variant of the serializability notion used in [11], which in turn is a variant of the serializability notion from [5, 14, 22]. Some subtle technical padding arguments are required to lift AC^0-serializability of PSPACE, proven in Theorem 22 in [11], to exponential LOGSPACE-serializability of EXPSPACE.

Theorem 2. *For every language L in* EXPSPACE *there is some regular language R such that L is exponentially* LOGSPACE*-serializable via R.*

A further concept we use is the Chinese remainder representation of a natural number. For every $m, M \in \mathbb{N}$ we denote by $\text{CRR}_m(M)$ *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 from its Chinese remainder representation. This result is a consequence of [7], where it is shown that division is in logspace-uniform NC^1.

Theorem 3 ([7] Theorem 3.3). *The following problem is in* LOGSPACE*:*
INPUT: $\mathrm{CRR}_m(M), j \in [m], b \in \{0,1\}$.
QUESTION: Is $\mathit{bit}_j(M \mathit{ mod } 2^m) = b$*?*

In the rest of this section, we sketch 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 2, there is some regular language $R \subseteq \{0,1\}^*$ s.t. L is exponentially LOGSPACE-serializable via R. Hence there is some language $U \in$ LOGSPACE s.t. for all $x \in \{0,1\}^n$ we have

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

where the bit strings on the right-hand side of the concatenation symbol are enumerated in lexicographic order. For the rest of this section, let us fix an input $x_0 \in \{0,1\}^n$. Let $N = p(n)$ and $A = (Q, \{0,1\}, q_0, \delta, F)$ be some deterministic finite automaton with $L(A) = R$. Let us describe equivalence (1) differently: We have $x_0 \in L$ iff the program in Fig. 1 returns `true`. We are going to mimic the execution of the program by a fixed CTL formula and a SOCA that can be computed from x_0 in logarithmic space. Before we start with the reduction, let us discuss the obstacles that arise:

```
q ∈ Q; q := q_0;
d ∈ ℕ; d := 0;
b ∈ {0,1};
while d ≠ 2^(2^N) loop
   b := χ_U(x_0 · bin_(2^N)(d));
   q := δ(q, b);
   d := d + 1;
endloop
return q ∈ F;
```

Fig. 1. A program that returns `true` iff $x_0 \in L$.

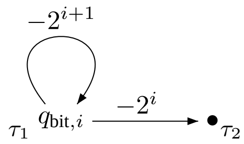
(A) We need some way of storing d on the counter. Of course there are a lot of ways to do this, but since we want to access all bits of d in the assignment $b := \chi_U(x_0 \cdot \mathrm{bin}_{2^N}(d))$, the most natural way is probably to represent d in binary. However, for this 2^N bits are required. 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 [12], 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 $b := \chi_U(x_0 \cdot \mathrm{bin}_{2^N}(d))$ 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.

Let us give a high-level description of the EXPSPACE-hardness proof. In the first step, we carefully design a data structure on the counter and explain the intuition behind it. In the second step, we list five queries which we aim at implementing via fixed CTL formulas and by SOCA that can be computed from the input 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. 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 $\beta = O(\log(K \log(K))) = O(N)$. The number α and such a sufficiently large number β can be computed from x_0 in logarithmic space.

Let us describe the data structure on the counter in our reduction. Assume that the counter value is $v \in \mathbb{N}$. We are interested only in the l least significant bits of the binary representation of v, where l is some number that is exponentially bounded in n; the value of l will be made clear below. Assume $V = \text{bit}_0(v) \cdots \text{bit}_{l-1}(v)$. We imagine V to be factorized into blocks of bits

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



$\varphi_{\text{bit},1} = \tau_1 \wedge \mathsf{EF}(\tau_1 \wedge \neg\mathsf{EX}\tau_1 \wedge \mathsf{EX}\tau_2)$
$\varphi_{\text{bit},0} = \tau_1 \wedge \mathsf{EF}(\tau_1 \wedge \neg\mathsf{EX}\tau_1 \wedge \neg\mathsf{EX}\tau_2)$

Fig. 2. SOCA $\mathbb{S}_{\text{bit},i}$ and CTL formulas $\varphi_{\text{bit},b}$ for checking if $\text{bit}_i(v) = b$.

where $I \in \{0,1\}^\alpha$ is a prime number index; $M \in \{0,1\}^\beta$ is intended to represent the I^{th} prime number p_I; $C \in \{0,1\}^\beta$ is some residue class modulo M; $J \in \{0,1\}^\alpha$ represents a pointer to some bit of B; X, Y and Z consist of polynomially many bits (in n) and are intended to represent the working tape of three space-bounded DTMs that we will comment on later in more detail; and $B \in \{0,1\}^{n+2^N}$ with $B = x_0 B'$ for some $B' \in \{0,1\}^{2^N}$. Our intention is that B represents the current input for U, and in particular B' represents the counting variable d from Fig. 1. 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 via the factorization $(*)$. Note that all of the bit strings have polynomial length in n except for B. Subsequently, we identify each of the blocks with the natural number they represent. A simple but powerful gadget, which will subsequently be used to check for each $b \in \{0,1\}$ if the i^{th} bit of the counter is b, is shown in Fig. 2. We have that $q_{\text{bit},i}(v)$ satisfies $\varphi_{\text{bit},b}$ iff $\text{bit}_i(v) = b$, for each $b \in \{0,1\}$.

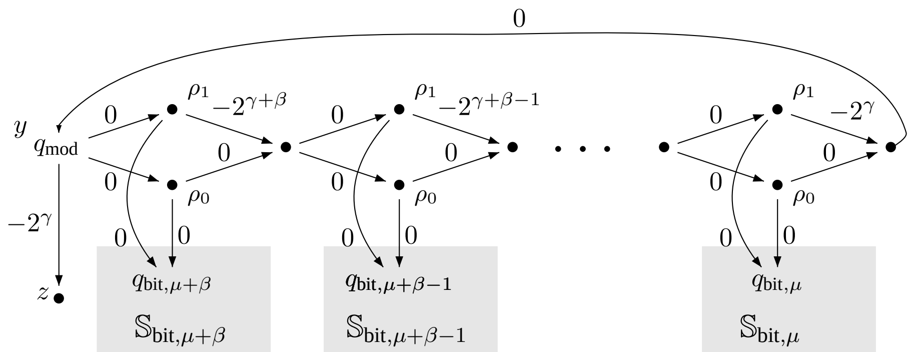
Queries that we need to implement: Next, we list five queries that we aim at answering by instances of the model checking problem. Each query is based on its preceding queries.

(Q1) Assuming $C < M$, does $C \equiv B \bmod M$ hold?
(Q2) Is M the I^{th} prime number, i.e. $M = p_I$?
(Q3) What is $\text{bit}_J(B)$?
(Q4) Does $(B[1,n] \cdot B[n+1, n+2^N]) \in U$ hold?
(Q5) Does $x_0 \in L$ hold?

EXPSPACE-hardness of data complexity of CTL on SOCA will hence follow from the implementation of query Q5. Let $\gamma = O(N)$ denote the absolute value of the leftmost bit position of B in V. Hence, when the γ^{th} bit of V is set to 1 and we subtract 2^γ from the counter, then the leftmost bit of B is set to 0. Similarly, let μ denote the leftmost bit position of M in V. Q1 can now be realized as follows.

Lemma 4. *There is a CTL formula φ_{mod} s.t. we can compute from x_0 in logarithmic space a SOCA $\mathbb{S}_{mod}$ and a control state q_{mod} s.t. $(T(\mathbb{S}_{mod}), q_{mod}(v)) \models \varphi_{mod}$ iff $C \equiv B \bmod M$.*

Proof. For brevity, we illustrate the special case $C = 0$, i.e. $(T(\mathbb{S}_{\text{mod}}), q_{\text{mod}}(v)) \models \varphi_{\text{mod}}$ iff $B \equiv 0 \bmod M$. The SOCA $\mathbb{S}_{\text{mod}}$ contains four atomic propositions ρ_0, ρ_1, y, z and is depicted below. The CTL formula φ_{mod} expresses that we traverse the sequence of diamonds and thereby repeatedly subtract M from B. The number of diamonds equals β, the number of bits of M. One diamond corresponds to one bit of M. In case bit β of M is 1, which we can verify by a transition to the initial control state of the SOCA $\mathbb{S}_{\text{bit},\mu+\beta}$ (see Fig. 2), we subtract $2^{\gamma+\beta}$ from B, otherwise we do not modify the counter value. This process is repeated until we reach the last diamond in which we consider the first bit of M. Finally, the transition from q_{mod} to the control state satisfying z serves for checking if $B = 0$ by trying to subtract 2^γ.



We put $\varphi_{\text{mod}} = \mathsf{E}\left(\bigwedge_{b\in\{0,1\}} \rho_b \to \mathsf{EX}\varphi_{\text{bit},b}\right) \mathsf{U}(y \wedge \neg\mathsf{EX}z)$. $\square$

Let us give some informal ideas on how to implement the queries Q2 to Q5. We strongly recommend the reader to consult the technical report [10] to understand the technical subtleties.

For implementing Q2 we simulate with a SOCA $\mathbb{S}_{\text{prime}}$ some polynomially space-bounded DTM that decides, on the input $\langle I, M\rangle$, whether $M = p_I$. We use the bit string X from $(*)$ for storing the working tape of this DTM on the counter. The current input and working tape symbol and the position of the input and working tape in the counter can directly be hard-wired into the control states of $\mathbb{S}_{\text{prime}}$. We can construct a fixed CTL formula that simulates the computation of this DTM.

Implementing Q3, i.e. deciding the J^{th} bit of B, is more involved. Recall that B consists of $n+2^N$ bits and J consists of $\alpha = O(N)$ bits. Hence checking if $\text{bit}_J(B) = 1$ cannot be done in a similar fashion as in Fig. 2, since J is too big. The solution is the following: By making use of $\mathbb{S}_{\text{prime}}$, one can initialize M with p_I and after that decide if $C \equiv B \bmod p_I$ by making use of $\mathbb{S}_{\text{mod}}$ and φ_{mod} from Lemma 4. Hence, we can access bits of the Chinese remainder representation of B. Let us assume that

$\mathcal{R} = \mathrm{CRR}_K(B) = (b_{i,c})_{1 \leq i \leq K, 0 \leq c < p_i}$ is the Chinese remainder representation of B. Observe that $|\mathcal{R}|$ is exponential in n and $\mathcal{R}$ is not stored anywhere on the counter. However we can use the bit strings I and C as pointers to access the bit $b_{I,C}$ of $\mathcal{R}$. By Theorem 3, given $\mathcal{R}$ (in our case on-the-fly by the pointers I and C), the bit string J and $b \in \{0,1\}$, we can decide if $\mathrm{bit}_J(B) = b$ by simulating a logarithmically space-bounded DTM on the input $\langle \mathcal{R}, J, b \rangle$ of exponential size. In the block Y of $(*)$ we reserve the space that this DTM requires. Q4 can be implemented similarly as Q3 by simulating a logarithmically space-bounded DTM that decides U on input $B[1,n] \cdot B[n+1, n+2^N]$ of exponential size. We use Z for simulating the working tape and the bit sequence J as a pointer to access the bits of B.

For implementing Q5 we simulate the program from in Fig. 1. Recall that our bit sequence B is of length $n + 2^N$. We initialize the first n bits of B with x_0. The remaining bit sequence B' stores the variable d of the program, initialized with 0 and being repeatedly incremented by adding $2^{\gamma+n}$. Thus, checking when d becomes 2^{2^N} for the first time boils down to checking when B' overflows for the first time. This can be checked by initializing J appropriately and being able to access the J^{th} bit via query Q3. The states of the automaton A can directly be handled by the control states of the SOCA. To obtain $\chi_U(x_0, \mathrm{bin}_{2^N}(d))$, we invoke the query Q4 and store this bit in the control state of the SOCA. This concludes our EXPSPACE-hardness proof.

Theorem 5. *The data complexity and the combined complexity of CTL and the modal μ-calculus on SOCA is* EXPSPACE*-complete.*

3.3 Hardness of the data complexity of CTL on POCA

We now show that there exists a fixed CTL formula for which model checking of POCA is Π^0_1-hard by a reduction from the emptiness problem for *two-counter automata*, which is Π^0_1-complete [19]. Similar to a SOCA, a two-counter automaton $\mathbb{A}$ consists of a finite set of control states and transitions between them. However, each transition of $\mathbb{A}$ acts on two counters, which it can in- and decrement and test for zero.

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 $\mathbb{P}(x)$. Given a counter value n of $\mathbb{P}(x)$, $n \bmod x$ encodes the value of the first, and $n \operatorname{div} x$ encodes the value of the second counter of $\mathbb{A}$. Hence, testing whether the first equals 0 corresponds to checking whether $n \equiv 0 \bmod x$, while testing whether the second counter equals 0 corresponds to checking whether $n \geq x$. Incrementing (resp. decrementing) the first counter of $\mathbb{A}$ can be mimicked by adding (resp. subtracting) 1, whereas on counter two this corresponds to adding (resp. subtracting) x. Of course, we need CTL formulas to ensure that we do not overflow when simulating an increment of the first counter of $\mathbb{A}$. For instance, if $n \equiv -1 \bmod x$ and we want to simulate an increment of the first counter of $\mathbb{A}$ in that way, we would actually set the first counter to 0 and simultaneously increment the second counter. However, if $\mathbb{A}$ is not empty, then x can be instantiated with a large enough value such that such an overflow does not occur. Conversely, if $\mathbb{A}$ is empty then there is no such instantiation.

Theorem 6. *The data and combined complexity of CTL on POCA is Π^0_1-complete.*

4 LTL Model Checking

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

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

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

$$(T,\pi) \models \rho \iff \pi(0) \in S_\rho, \rho \in \mathcal{P} \qquad (T,\pi) \models \neg\varphi \iff \pi \not\models \varphi$$
$$(T,\pi) \models \varphi_1 \wedge \varphi_2 \iff \forall i \in \{1,2\} : (T,\pi) \models \varphi_i \quad (T,\pi) \models \mathsf{X}\varphi \iff (T,\pi^1) \models \varphi$$
$$(T,\pi) \models \varphi_1 \mathsf{U} \varphi_2 \iff \exists j \geq 0 : (T,\pi^j) \models \varphi_2 \text{ and } \forall 0 \leq i < j : (T,\pi^i) \models \varphi_1$$

LTL MODEL CHECKING ON SOCA
INPUT: SOCA $\mathbb{S} = (Q, E, \lambda)$, $q \in Q$, $n \in \mathbb{N}$ in binary, an LTL formula φ.
QUESTION: Does $(T(\mathbb{S}), \pi) \models \varphi$ for all infinite paths π with $\pi(0) = q(n)$?

LTL MODEL CHECKING ON POCA
INPUT: POCA $\mathbb{P}(X) = (Q, E, \lambda)$, $q \in Q$, $n \in \mathbb{N}$ in binary, an LTL formula φ.
QUESTION: Does $(T(\mathbb{P}^\sigma), \pi) \models \varphi$ for all $\sigma : X \rightarrow \mathbb{N}$ and for all infinite paths π with $\pi(0) = q(n)$?

4.1 Upper Bounds

A standard approach to LTL model checking is the automata-based approach, in which behaviours of *systems* are modelled as non-deterministic Büchi automata (NBA). Given an NBA A modelling a system and an LTL formula φ, the idea is to translate φ into an NBA $A_{\neg\varphi}$ of size $2^{O(|\varphi|)}$ such that the language of $A \times A_{\neg\varphi}$ is empty iff φ holds on all infinite traces of A. The concept of Büchi automata can easily be adopted to the setting of counter automata. Then a *Büchi-SOCA* is not empty if there is an infinite path on which some designated control states occurs infinitely often. The latter boils down to just checking for recurrent reachability. Moreover, the Büchi-SOCA obtained from the product of a SOCA and an NBA can be defined and constructed in a straightforward way. We omit details for brevity.

It was shown in [13] that checking emptiness is coNP-complete for both Büchi-SOCA and Büchi-POCA, and in [9] that it is NL-complete for Büchi-OCA. We use these results for establishing upper bounds for the LTL model checking problems.

For every fixed LTL formula φ, and every POCA $\mathbb{P}$, the size of $\mathbb{P} \times A_{\neg\varphi}$ is $O(|\mathbb{P}|)$, hence the data complexity of LTL on SOCA and POCA is in coNP. Hardness for coNP follows from NP-hardness of reachability using a fixed formula $\mathtt{tt}\mathsf{U}\rho$ for some $\rho \in \mathcal{P}$.

If both $\mathbb{P}$ and φ are part of the input then $|\mathbb{P} \times A_{\neg\varphi}| = |\mathbb{P}| \cdot 2^{O(|\varphi|)}$, and hence [13] gives a coNEXP upper bound for the combined complexity of LTL model checking on both SOCA and POCA. This upper bound can however be improved for SOCA. Given a SOCA $\mathbb{S}$, let m be the absolute value of the maximum increment or decrement on the transitions in $\mathbb{S}$. Let $\mathbb{S}'$ be the Büchi-SOCA obtained from the product $\mathbb{S} \times A_{\neg\varphi}$ by replacing every transition labeled with z with a sequence of fresh transitions and control states of length z, where, depending on the sign of z, each transition is labeled with $+1$ resp. -1. We have $|\mathbb{S}'| = m \cdot |\mathbb{S}| \cdot 2^{O(|\varphi|)}$, and hence the NL upper bound for emptiness of Büchi-OCA from [9] yields a PSPACE upper bound.

Proposition 7. *The data complexity of LTL model checking on SOCA and POCA is* coNP*-complete, the combined complexity of LTL model checking on SOCA is* PSPACE*-complete, and the combined complexity of LTL model checking on POCA is in* coNEXP.

4.2 Hardness of the Combined Complexity of LTL on POCA

We are now going to sketch a proof of coNEXP-hardness of LTL model checking on POCA via a reduction from the complement of the NEXP-complete Succinct 3-SAT problem [20]. An input of Succinct 3-SAT is given by a Boolean circuit $\mathbb{C}$ that encodes a Boolean formula ψ in 3-CNF, i.e. $\psi = \bigwedge_{0 \leq j < M}(\ell_1^j \vee \ell_2^j \vee \ell_3^j)$. Let $j \in [M]$ be the index of a clause encoded in *binary* and $k \in \{1, 2, 3\}$. Assume that ψ uses N different variables $y_1, \ldots, y_N$. 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 is to decide whether ψ is satisfiable. Fig. 3 depicts on a high-level the POCA $\mathbb{P}(x)$ derived from $\mathbb{C}$ that we are using in our reduction.

As a first step, 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 [17]. Recall that p_i denotes the i^{th} prime number. Every natural number n defines a truth assignment $\nu_n : \{y_1, \ldots, y_N\} \to \{0, 1\}$ such that $\nu_n(y_i) = 1$ iff p_i divides n. 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 $\mathbb{P}(x)$.

Let us now take a look at $\mathbb{P}(x)$. It uses one parameter x and employs several gadgets. Only the gadgets $\mathbb{S}_{\mathsf{divides}}$ and $\mathbb{S}_{\mathsf{not\_divides}}$ manipulate the counter. All gadgets are designed so that they communicate via designated propositional variables, and not as in Section 3.2 with the help of the counter. Starting in q_{start}, $\mathbb{P}(x)$ first loads the value of the parameter x on the counter. Think of x encoding a truth assignment of ψ. Next, $\mathbb{P}(x)$ traverses through $\mathbb{S}_{\mathsf{inc}}$, which initially chooses an arbitrary index j identifying a clause of ψ. Every time $\mathbb{S}_{\mathsf{inc}}$ is traversed afterwards, it increments j modulo N 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 that p_i does not divide x, and likewise that p_i divides x if $b = 1$. Those checks need to modify the counter. After they have finished, we restore the value x on the counter and the process continues with clause $j + 1$ mod N. We can construct an LTL formula φ that ensures that all gadgets work and communicate correctly, and prove that ψ is satisfiable iff there is an assignment σ and an infinite path $\pi = q_{\mathsf{start}}(0) \to \cdots$ such that $(T(\mathbb{P}^\sigma), \pi) \not\models \varphi$. The gadgets $\mathbb{S}_{\mathsf{inc}}$, $\mathbb{S}_{\mathsf{circuit}}$ and $\mathbb{S}_{\mathsf{inc}}$ can be realized by simulating space-bounded Turing machines with SOCA and some appropriate LTL formulas. Here

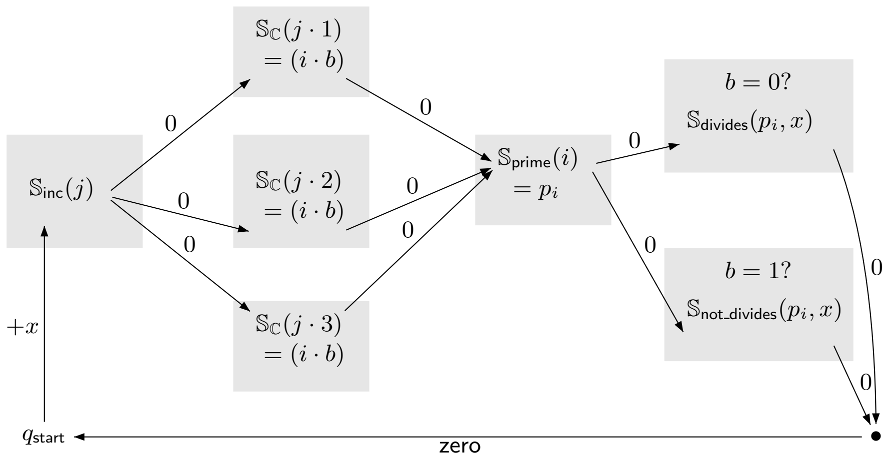


Fig. 3. High-level description of the POCA $\mathbb{P}(x)$ used for the reduction from Succinct 3-SAT.

it is important that our LTL formula φ is not fixed. Divisibility resp. non-divisibility is checked similar as in the CTL case, cf. Lemma 4. We refer the reader to the technical report for further details [10].

Theorem 8. *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, the modal μ-calculus and LTL on SOCA and POCA with respect to data and combined complexity. Our proofs for providing lower bounds have introduced some non-trivial 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 to consider *synthesis problems* for POCA. Given a POCA $\mathbb{P}(X)$ and a formula φ, a natural question to ask is whether there *exists an assignment* σ such that $(T(\mathbb{P}^\sigma), \pi) \models \varphi$ on all infinite paths π starting in some state of $T(\mathbb{P}^\sigma)$. For CTL resp. the modal μ-calculus, such a problem is undecidable by Theorem 6. However for LTL it seems conceivable that this problem can be translated into a sentence of a decidable fragment of Presburger arithmetic with divisibility, similar to those studied in [4].

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. A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. In *Proc. of CAV*, volume 4144 of *LNCS*, pages 517–531. Springer, 2006.

3. M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. In *Proc. of ICALP*, volume 4052 of *LNCS*, pages 577–588. Springer, 2006.
4. Marius Bozga and Radu Iosif. On decidability within the arithmetic of addition and divisibility. In *Proc. of FOSSACS*, volume 3441 of *LNCS*, pages 425–439. Springer, 2005.
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 *Proc. of WebDB*, pages 85–90. ACM, 2004.
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. of CAV*, volume 1427 of *LNCS*, pages 268–279. Springer, 1998.
9. Stéphane Demri and Régis Gascon. The effects of bounding syntactic resources on Presburger LTL. *Journal of Logic and Computation*, 19(6):1541–1575, December 2009.
10. Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Model checking succinct and parametric one-counter automata. Technical report, University of Bremen, 2010. Available via `http://www.informatik.uni-bremen.de/tdki/research/papers/succ.pdf`.
11. Stefan Göller and Markus Lohrey. Branching-time model checking of one-counter processes. Technical report, arXiv.org, 2009. `http://arxiv.org/abs/0909.1102`.
12. Stefan Göller and Markus Lohrey. Branching-time model checking of one-counter processes. In *Proc. of STACS*, pages 405–416. IFIB Schloss Dagstuhl, 2010.
13. 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`.
14. Ulrich Hertrampf, Clemens Lautemann, Thomas Schwentick, Heribert Vollmer, and Klaus W. Wagner. On the power of polynomial time bit-reductions. In *Proc. of CoCo*, pages 200–207. IEEE Computer Society Press, 1993.
15. O. H. Ibarra, T. Jiang, N. Trân, and H. Wang. New decidability results concerning two-way counter machines and applications. In *Proc. of ICALP*, volume 700 of *LNCS*, pages 313–324. Springer, 1993.
16. 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.
17. 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.
18. J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *Proc. of ATVA'05*, volume 3707 of *LNCS*, pages 489–503. Springer, 2005.
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 *Proc. of FOSSACS*, volume 3921 of *LNCS*, pages 337–351. Springer, 2006.
22. Heribert Vollmer. A generalized quantifier concept in computational complexity theory. Technical report, arXiv.org, 1998. `http://arxiv.org/abs/cs.CC/9809115`.