The Complexity of Flat Freeze LTL*

Benedikt Bollig[1], Karin Quaas[2], and Arnaud Sangnier[3]

1 **CNRS, LSV, ENS Paris-Saclay, Université Paris-Saclay**
2 **Universität Leipzig**
3 **IRIF, Université Paris Diderot**

Abstract

We consider the model-checking problem for freeze LTL on one-counter automata (OCAs). Freeze LTL extends LTL with the freeze quantifier, which allows one to store different counter values of a run in registers so that they can be compared with one another. As the model-checking problem is undecidable in general, we focus on the flat fragment of freeze LTL, in which the usage of the freeze quantifier is restricted. Recently, Lechner et al. showed that model checking for flat freeze LTL on OCAs with binary encoding of counter updates is decidable and in 2NEXPTIME. In this paper, we prove that the problem is, in fact, NEXPTIME-complete no matter whether counter updates are encoded in unary or binary. Like Lechner et al., we rely on a reduction to the reachability problem in OCAs with parameterized tests (OCAPs). The new aspect is that we simulate OCAPs by alternating two-way automata over words. This implies an exponential upper bound on the parameter values that we exploit towards an NP algorithm for reachability in OCAPs with unary updates. We obtain our main result as a corollary.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases one-counter automata, freeze LTL, model checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2017.29

1 Introduction

One-counter automata (OCAs) are a simple yet fundamental computational model operating on a single counter that ranges over the non-negative integers. Albeit being a classical model, OCAs are in the focus of ongoing research in the verification community (cf., for example, [9, 18–20, 22, 23, 26]). A large body of work is devoted to model checking OCAs, i.e., the question whether all runs of a given OCA satisfy a temporal-logic specification. A natural way to model runs of OCAs is in terms of *data words*, i.e., words where every position carries two pieces of information: a set of propositions from a finite alphabet, and a datum from an infinite alphabet. In our case, the datum represents the current counter value. Now, reasoning about the sequence of propositions that is produced by a run amounts to model checking OCAs against classical temporal logics like linear-time temporal logic (LTL) and computation-tree logic (CTL) [18–21]. But it is natural to also reason about the unboundedly many counter values that may occur. Several formalisms have been introduced that can handle infinite alphabets, including variants or extensions of monadic second-order logic, LTL, and CTL [3, 10, 12, 15]. Freeze LTL is an extension of LTL that allows one to remember, in terms of the *freeze quantifier*, certain counter values for later comparison in a run of the OCA under consideration (cf. [10, 11, 16, 27]). Unfortunately, satisfiability and model checking OCAs against freeze LTL are undecidable [10, 12].

* This work has been partly supported by the ANR research program PACS (ANR-14-CE28-0002) and by DFG, project QU 316/1-2.

In this paper, we study model checking of OCAs against formulas in *flat freeze LTL*, a fragment of freeze LTL that restricts the freeze quantifier, but allows for unlimited usage of comparisons. A typical property definable in flat freeze LTL is "there exists a counter value that occurs infinitely often". Moreover, the negation of many natural freeze LTL specifications can be expressed in flat freeze LTL. The approach of restricting the syntax of a temporal logic to its flat fragment has also been pursued in [7] for Constraint LTL, and in [4] for MTL.

Demri and Sangnier [13] reduced model checking OCAs against flat freeze LTL to reachability in OCAs *with parameterized tests* (OCAPs). In an OCAP, counter values may be compared with parameters whose values are arbitrary but fixed. Reachability asks whether a given state can be reached under *some* parameter instantiation. Essentially, the translation of a flat freeze LTL formula into an OCAP interprets every freeze quantifier as a parameter, whose value can be compared with counter values arbitrarily often. Decidability of the reachability problem for OCAPs, however, was left open. Recently, Lechner et al. proved decidability by a reduction to satisfiability in Presburger arithmetic [26]. As a corollary, they obtain a 2NEXPTIME upper bound for model checking OCAs against formulas in flat freeze LTL, assuming that counter updates in OCAs are encoded in binary.

Our main technical contribution is an improvement of the result by Lechner et al. We proceed in two main steps. First, we show that reachability for OCAPs (with unary counter updates) can be reduced to non-emptiness of alternating two-way (finite) automata. Interestingly, alternating two-way automata have already been used as an algorithmic framework for game-based versions of pushdown processes [6, 24], one-counter processes [28], and timed systems [1]. Our link already implies decidability of both reachability for OCAPs (in PSPACE when counter updates are unary) and model checking OCAs against flat freeze LTL (in EXPSPACE). However, we can go further. First, we deduce an exponential upper bound on the largest parameter value needed for an accepting run in the given OCAP. Exploiting this bound and a technique by Galil [17], we show that, actually, reachability for OCAPs is in NP. As a corollary, we obtain a NEXPTIME upper bound for model checking OCAs against flat freeze LTL. Using a result from [18], we can also show NEXPTIME-hardness, which establishes the precise complexity of the model-checking problem. Our result applies no matter whether counter updates in the given OCA are encoded in unary or binary.

Outline. In Section 2, we define OCAs, (flat) freeze LTL, and the corresponding model-checking problems. Section 3 is devoted to reachability in OCAPs, which is at the heart of our model-checking procedures. The reduction of model checking to reachability in OCAPs is given in Section 4, where we also present lower bounds. We conclude in Section 5.

2 Preliminaries

2.1 One-Counter Automata with Parameterized Tests

We start by defining one-counter automata *with all extras* such as parameters and succinct encodings of updates. Well-known subclasses are then identified as special cases.

For the rest of this paper, we fix a countably infinite set $\mathbb{P}$ of *propositions*, which will label states of a one-counter automaton. Transitions of an automaton may perform tests to compare the current counter value with zero or with a parameter. Thus, for a finite set $\mathcal{X}$ of parameters, we let $\mathrm{Tests}(\mathcal{X}) = \{\mathsf{zero?}\} \cup \{{<}x\,, {=}x\,, {>}x \mid x \in \mathcal{X}\}$.

▶ **Definition 1.** A *succinct one-counter automaton with parameterized tests (SOCAP)* is a tuple $\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta, \mu)$ where Q is a finite set of *states*, $\mathcal{X}$ is a finite set of *parameters* ranging over $\mathbb{N}$, $q_{\mathsf{in}} \in Q$ is the *initial state*, $\Delta \subseteq Q \times (\mathbb{Z} \cup \mathrm{Tests}(\mathcal{X})) \times Q$ is a finite set

of *transitions*, and $\mu : Q \to 2^{\mathbb{P}}$ maps each state to a *finite* set of propositions. We define the size of $\mathcal{A}$ as $|\mathcal{A}| = |Q| + |\mathcal{X}| + \sum_{q \in Q} |\mu(q)| + |\Delta| \cdot \max\big(\{1\} \cup \big\{\log(|z|) \mid (q, z, q') \in \Delta \cap (Q \times (\mathbb{Z} \setminus \{0\}) \times Q)\big\}\big)$.

Thus, transitions of a SOCAP either compare the counter value with zero or a parameter value, or increment/decrement the counter by a value $z \in \mathbb{Z}$, which is encoded in binary.

Let $\mathcal{C}_{\mathcal{A}} := Q \times \mathbb{N}$ be the set of *configurations* of $\mathcal{A}$. In a configuration $(q, v) \in \mathcal{C}_{\mathcal{A}}$, the first component q is the current state and v is the current counter value which is always non-negative. The semantics of $\mathcal{A}$ is given w.r.t. a *parameter instantiation* $\gamma : \mathcal{X} \to \mathbb{N}$ in terms of a global transition relation $\longrightarrow_\gamma \subseteq \mathcal{C}_{\mathcal{A}} \times \mathcal{C}_{\mathcal{A}}$. For two configurations (q, v) and (q', v'), we have $(q, v) \longrightarrow_\gamma (q', v')$ if there is $(q, \mathsf{op}, q') \in \Delta$ such that one of the following holds:

- $\mathsf{op} \in \mathbb{Z}$ and $v' = v + \mathsf{op}$,
- $\mathsf{op} = \mathsf{zero?}$ and $v = v' = 0$, or
- $\mathsf{op} = \bowtie x$ and $v = v'$ and $v \bowtie \gamma(x)$, for some $\bowtie \in \{<, =, >\}$.

A γ-*run* of $\mathcal{A}$ is a finite or infinite sequence $\rho = (q_0, v_0) \longrightarrow_\gamma (q_1, v_1) \longrightarrow_\gamma \cdots$ of global transitions (a run may consist of one single configuration (q_0, v_0)). We say that ρ is *initialized* if $q_0 = q_{\mathsf{in}}$ and $v_0 = 0$.

We identify some well-known special cases of the general model. Let $\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta, \mu)$ be a SOCAP. We say that

- $\mathcal{A}$ is a **one-counter automaton with parameterized tests (OCAP)** if all transition labels are among $\{+1, 0, -1\} \cup \mathrm{Tests}(\mathcal{X})$ (i.e., counter updates are *unary*);
- $\mathcal{A}$ is a **succinct one-counter automaton (SOCA)** if $\mathcal{X} = \emptyset$;
- $\mathcal{A}$ is a **one-counter automaton (OCA)** if $\mathcal{X} = \emptyset$ and, moreover, all transition labels are among $\{+1, 0, -1\} \cup \{\mathsf{zero?}\}$.

Note that, when $\mathcal{X} = \emptyset$ (i.e., in the case of a SOCA or OCA), we may omit $\mathcal{X}$ and simply refer to $(Q, q_{\mathsf{in}}, \Delta, \mu)$. Also, the global transition relation does not depend on a parameter instantiation anymore so that we may just write $\longrightarrow$ instead of $\longrightarrow_\gamma$. Similarly, for reachability problems (see below), μ is irrelevant so that it can be omitted, too.

One of the most fundamental decision problems we can consider is whether a given state $q_f \in Q$ is reachable. Given some parameter instantiation γ, we say that q_f is γ-*reachable* if there is an initialized run $(q_0, v_0) \longrightarrow_\gamma \cdots \longrightarrow_\gamma (q_n, v_n)$ such that $q_n = q_f$. Moreover, q_f is *reachable*, written $\mathcal{A} \models \mathit{Reach}(q_f)$, if it is γ-reachable for some γ. Now, for a class $\mathcal{C} \in \{\mathrm{SOCAP}, \mathrm{OCAP}, \mathrm{SOCA}, \mathrm{OCA}\}$, the *reachability problem* is defined as follows:

$\mathcal{C}$-Reachability	
Input:	$\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta) \in \mathcal{C}$ and $q_f \in Q$
Question:	Do we have $\mathcal{A} \models \mathit{Reach}(q_f)$?

Towards the *Büchi problem* (or *repeated reachability*), we write $\mathcal{A} \models \mathit{Reach}^\omega(q_f)$ if there are a parameter instantiation γ and an infinite initialized γ-run $(q_0, v_0) \longrightarrow_\gamma (q_1, v_1) \longrightarrow_\gamma \cdots$ such that $q_i = q_f$ for infinitely many $i \in \mathbb{N}$. The Büchi problem asks whether some state from a given set $F \subseteq Q$ can be visited infinitely often:

$\mathcal{C}$-Büchi	
Input:	$\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta) \in \mathcal{C}$ and $F \subseteq Q$
Question:	Do we have $\mathcal{A} \models \mathit{Reach}^{\omega}(q_f)$ for some $q_f \in F$?

It is known that OCA-Reachability and OCA-Büchi are NLOGSPACE-complete [9,29], and that SOCA-Reachability and SOCA-Büchi are NP-complete [21,22] (cf. also Table 1).

2.2 Freeze LTL and Its Flat Fragment

We now define freeze LTL. To do so, we fix a countably infinite supply of registers $\mathcal{R}$.

▶ **Definition 2** (Freeze LTL). The logic *freeze LTL*, denoted by $\mathrm{LTL}^{\downarrow}$, is given by the grammar

$$\varphi \quad ::= \quad p \;\mid\; \bowtie r \;\mid\; \neg\varphi \;\mid\; \varphi \vee \varphi \;\mid\; \varphi \wedge \varphi \;\mid\; \mathsf{X}\varphi \;\mid\; \varphi\,\mathsf{U}\,\varphi \;\mid\; \varphi\,\mathsf{R}\,\varphi \;\mid\; \downarrow_r \varphi$$

where $p \in \mathbb{P}$, $r \in \mathcal{R}$, and $\bowtie \in \{<, =, >\}$.

Note that, apart from the until operator U, we also include the dual release operator R. This is convenient in proofs, as it allows us to assume formulas to be in negation normal form. We also use common abbreviations such as $\varphi \to \psi$ for $\neg\varphi \vee \psi$, $\mathsf{F}\varphi$ for $\mathsf{true}\,\mathsf{U}\,\varphi$ (with $\mathsf{true} = p \vee \neg p$ for some proposition p), and $\mathsf{G}\varphi$ for $\neg\mathsf{F}\neg\varphi$.

We call $\downarrow_r$ the *freeze quantifier*. It stores the current counter value in register r. Atomic formulas of the form $\bowtie r$, called *register tests*, perform a comparison of the current counter value with the contents of r, provided that they are in the scope of a freeze quantifier. Actually, inequalities of the form $<r$ and $>r$ were not present in the original definition of $\mathrm{LTL}^{\downarrow}$, but our techniques will take care of them without extra cost or additional technical complication. Classical LTL is obtained as the fragment of $\mathrm{LTL}^{\downarrow}$ that uses neither the freeze quantifier nor register tests.

A formula $\varphi \in \mathrm{LTL}^{\downarrow}$ is interpreted over an infinite sequence $w = (P_0, v_0)(P_1, v_1)\ldots \in (2^{\mathbb{P}} \times \mathbb{N})^{\omega}$ with respect to a position $i \in \mathbb{N}$ and a register assignment $\nu : \mathcal{R} \to \mathbb{N}$. The satisfaction relation $w, i \models_{\nu} \varphi$ is inductively defined as follows:

- $w, i \models_{\nu} p$ if $p \in P_i$,
- $w, i \models_{\nu} \bowtie r$ if $v_i \bowtie \nu(r)$,
- $w, i \models_{\nu} \mathsf{X}\varphi$ if $w, i+1 \models_{\nu} \varphi$,
- $w, i \models_{\nu} \varphi_1\,\mathsf{U}\,\varphi_2$ if there is $j \geq i$ such that $w, j \models_{\nu} \varphi_2$ and $w, k \models_{\nu} \varphi_1$ for all $k \in \{i, \ldots, j-1\}$,
- $w, i \models_{\nu} \varphi_1\,\mathsf{R}\,\varphi_2$ if one of the following holds: (i) $w, k \models_{\nu} \varphi_2$ for all $k \geq i$, or (ii) there is $j \geq i$ such that $w, j \models_{\nu} \varphi_1$ and $w, k \models_{\nu} \varphi_2$ for all $k \in \{i, \ldots, j\}$,
- $w, i \models_{\nu} \downarrow_r \varphi$ if $w, i \models_{\nu[r \mapsto v_i]} \varphi$, where the register assignment $\nu[r \mapsto v_i]$ maps r to v_i and coincides with ν on all other registers.

Negation, disjunction, and conjunction are interpreted as usual.

Let φ be a *sentence*, i.e., every subformula of φ of the form $\bowtie r$ is in the scope of a freeze quantifier $\downarrow_r$. Then, the initial register assignment is irrelevant, and we simply write $w \models \varphi$ if $w, 0 \models_{\nu} \varphi$ (with ν arbitrary). Let $\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta, \mu)$ be a SOCAP and let $\rho = (q_0, v_0) \longrightarrow_{\gamma} (q_1, v_1) \longrightarrow_{\gamma} \cdots$ be an infinite run of $\mathcal{A}$. We say that ρ satisfies φ, written $\rho \models \varphi$, if $(\mu(q_0), v_0)(\mu(q_1), v_1)\ldots \models \varphi$. Moreover, we write $\mathcal{A} \models_{\exists} \varphi$ if there *exist* γ and an infinite initialized γ-run ρ of $\mathcal{A}$ such that $\rho \models \varphi$.

Model checking for a class $\mathcal{C}$ of SOCAPs and a logic $\mathcal{L} \subseteq \mathrm{LTL}^{\downarrow}$ is defined as follows:

Table 1 Old and new results

	Reachability/ Büchi	MC(LTL)	MC(flatLTL$^{\downarrow}$)
OCA	NLOGSPACE-complete [29] / [9]	PSPACE-complete (e.g., [21])	NEXPTIME-complete Theorem 21
SOCA	NP-complete [22] / [21]	PSPACE-complete [18]	NEXPTIME-complete Theorem 21

$\mathcal{C}$-MC($\mathcal{L}$)

Input: $\mathcal{A} \in \mathcal{C}$ and a sentence $\varphi \in \mathcal{L}$

Question: Do we have $\mathcal{A} \models_{\exists} \varphi$?

Note that, following [12, 26], we study the existential version of the model-checking problem ("Is there some run satisfying the formula?"). The reason is that the flat fragment that we consider next is not closed under negation, and the negation of many useful freeze LTL formulas are actually *flat* (cf. Example 4 below).

Unfortunately, OCA-MC(LTL$^{\downarrow}$) is undecidable [12], even if formulas use only one register. This motivates the study of the flat fragment of LTL$^{\downarrow}$, restricting the usage of the freeze quantifier [12]. Essentially, it is no longer possible to overwrite a register unboundedly often.

▶ **Definition 3** (Flat Freeze LTL). The fragment flatLTL$^{\downarrow}$ of LTL$^{\downarrow}$ contains a formula φ if, for every occurrence of a subformula $\psi = \varphi_1 \,\mathsf{U}\, \varphi_2$ (respectively, $\psi = \varphi_2 \,\mathsf{R}\, \varphi_1$) in φ, the following hold: (i) If the occurrence of ψ is in the scope of an even number of negations, then the freeze quantifier does not occur in φ_1, and (ii) if it is in the scope of an odd number of negations, then the freeze quantifier does not occur in φ_2.

▶ **Example 4.** An example flatLTL$^{\downarrow}$ formula is $\mathsf{F}{\downarrow_r}\mathsf{G}\big((<r) \vee (=r)\big)$, saying that a run takes only finitely many different counter values. The formula $\varphi = \mathsf{G}{\downarrow_r}\big(\mathit{req} \rightarrow \mathsf{F}(\mathit{serve} \wedge (=r))\big)$ (from [26]) says that every request is eventually served with the same ticket. Note that φ is *not* in flatLTL$^{\downarrow}$, but its negation $\neg\neg\big(\mathsf{true}\,\mathsf{U}\,\neg{\downarrow_r}\big(\mathit{req} \rightarrow \mathsf{F}(\mathit{serve} \wedge (=r))\big)\big)$ is: The until lies in the scope of an even number of negations, and the freeze quantifier is in the second argument of the until.

In [13], it has been shown that OCA-MC(flatLTL$^{\downarrow}$) can be reduced to OCAP-Büchi, but decidability of the latter was left open (positive results were only obtained for a restricted fragment of flatLTL$^{\downarrow}$). Recently, Lechner et al. showed that OCAP-Büchi is decidable [26]. In fact, they establish that SOCA-MC(flatLTL$^{\downarrow}$) is in 2NEXPTIME.

Our main result states that the problems OCA-MC(flatLTL$^{\downarrow}$) and SOCA-MC(flatLTL$^{\downarrow}$) are both NEXPTIME-complete. A comparison with known results can be found in Table 1. The proof outline is depicted in Figure 1. Essentially, we also rely on a reduction of SOCA-MC(flatLTL$^{\downarrow}$) to OCAP-Reachability, and the main challenge is to establish the precise complexity of the latter. In Section 3, we reduce OCAP-Reachability to non-emptiness of alternating two-way automata over infinite words, which is then exploited to prove an NP upper bound. The reductions from SOCA-MC(flatLTL$^{\downarrow}$) to OCAP-Reachability themselves are given in Section 4, which also contains our main results.

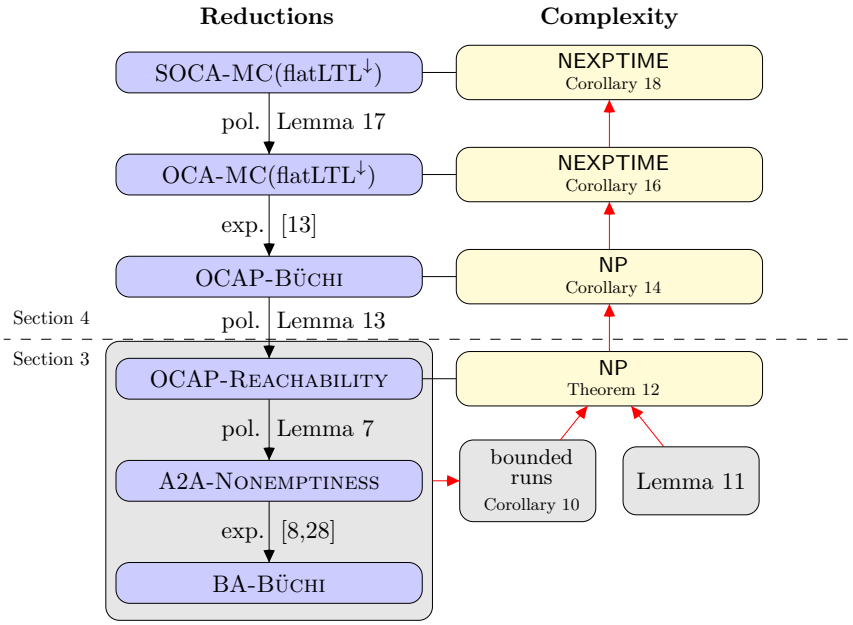

Figure 1 Proof structure for upper bounds

3 OCAP-Reachability is NP-complete

In this section, we reduce OCAP-Reachability to non-emptiness of alternating two-way automata (A2As) over infinite words. While this already implies that OCAP-Reachability is in PSPACE, we then go a step further and show that the problem is in NP.

3.1 From OCAPs to Alternating Two-Way Automata

We fix an OCAP $\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta)$. The main idea is to encode a parameter instantiation $\gamma : \mathcal{X} \to \mathbb{N}$ as a word over the alphabet $\Sigma = \mathcal{X} \cup \{\square\}$, where $\square$ is a fresh symbol. A word $w = a_0 a_1 a_2 \cdots \in \Sigma^\omega$, with $a_i \in \Sigma$, is called a *parameter word* (over $\mathcal{X}$) if $a_0 = \square$ and, for all $x \in \mathcal{X}$, there is exactly one position $i \in \mathbb{N}$ such that $a_i = x$. In other words, w starts with $\square$ and every parameter occurs exactly once. Then, w determines a parameter instantiation $\gamma_w : \mathcal{X} \to \mathbb{N}$ as follows: If $x = a_i$, then $\gamma_w(x) = |a_1 \cdots a_{i-1}|_\square$ where $|a_1 \cdots a_{i-1}|_\square$ denotes the number of occurrences of $\square$ in $a_1 \cdots a_{i-1}$ (note that we start at the second position of w). For example, given $\mathcal{X} = \{x_1, x_2, x_3\}$, both $w = \square x_2 \square\square x_1 x_3 \square^\omega$ and $w' = \square x_2 \square\square x_3 x_1 \square^\omega$ are parameter words with $\gamma_w = \gamma_{w'} = \{x_1 \mapsto 2, x_2 \mapsto 0, x_3 \mapsto 2\}$. Note that, for every parameter instantiation γ, there is at least one parameter word w such that $\gamma_w = \gamma$. Let $\mathcal{W}_\mathcal{X}$ denote the set of all parameter words over $\mathcal{X}$.

From $\mathcal{A}$ and a state $q_f \in Q$, we will build an A2A that accepts the set of parameter words w such that q_f is γ_w-reachable. Like a Turing machine, an A2A can read a letter, change its state, and move its head to the left or to the right (or stay at the current position). In addition, it can spawn several independent copies, for example one that goes to the left and one that goes to the right. However, unlike a Turing machine, an A2A is not allowed to modify the input word so that its expressive power does not go beyond finite (Büchi) automata. The simulation proceeds as follows. When the OCAP increments its counter, the A2A moves to the right to the next occurrence of $\square$. To simulate a decrement, it moves

to the left until it encounters the previous $\square$. To mimic the zero test, it verifies that it is currently on the first position of the word. Moreover, it will make use of the letters in $\mathcal{X}$ to simulate parameter tests. At the beginning of an execution, the A2A spawns independent copies that check whether the input word is a valid parameter word.

Let us define A2As and formalize the simulation of an OCAP. Given a finite set Y, we denote by $\mathbb{B}^+(Y)$ the set of positive Boolean formulas over Y, including `true` and `false`. A subset $Y' \subseteq Y$ *satisfies* $\beta \in \mathbb{B}^+(Y)$, written $Y' \models \beta$, if β is evaluated to true when assigning `true` to every variable in Y', and `false` to every variable in $Y \backslash Y'$. In particular, we have $\emptyset \models$ `true`. For $\beta \in \mathbb{B}^+(Y)$, let $|\beta|$ denote the size of β, defined inductively by $|\beta_1 \vee \beta_2| = |\beta_1 \wedge \beta_2| = |\beta_1| + |\beta_2| + 1$ and $|\beta| = 1$ for atomic formulas β.

▶ **Definition 5.** An *alternating two-way automaton* (A2A) is a tuple $\mathcal{T} = (S, \Sigma, s_{\mathsf{in}}, \delta, S_f)$, where S is a finite set of states, Σ is a finite alphabet, $s_{\mathsf{in}} \in S$ is the initial state, $S_f \subseteq S$ is the set of accepting states, and $\delta \subseteq S \times (\Sigma \cup \{\mathsf{first?}\}) \times \mathbb{B}^+(S \times \{+1, 0, -1\})$ is the finite transition relation[1]. A transition $(s, \mathit{test}, \beta) \in \delta$ will also be written $s \xrightarrow{\mathit{test}} \beta$. The size of $\mathcal{T}$ is defined as $|\mathcal{T}| = |S| + |\Sigma| + \sum_{(s,\mathit{test},\beta)\in\delta} |\beta|$.

While, in an OCAP, $+1$ and -1 are interpreted as *increment* and *decrement* the counter, respectively, their interpretation in an A2A is *go to the right* and *go to the left* in the input word. Moreover, 0 means *stay*. Actually, when $\delta \subseteq S \times \Sigma \times (S \times \{+1\})$, then we deal with a classical *Büchi automaton*.

A *run* of $\mathcal{T}$ on an infinite word $w = a_0 a_1 a_2 \cdots \in \Sigma^\omega$ is a rooted tree (possibly infinite, but finitely branching) whose vertices are labeled with elements in $S \times \mathbb{N}$. A node with label (s, n) represents a proof obligation that has to be fulfilled starting from state s and position n in the input word. The root of a run is labeled by $(s_{\mathsf{in}}, 0)$. Moreover, we require that, for every vertex labeled by (s, n) with $k \in \mathbb{N}$ children labeled by $(s_1, n_1), \ldots, (s_k, n_k)$, there is a transition $(s, \mathit{test}, \beta) \in \delta$ such that (i) the set $\{(s_1, n_1 - n), \ldots, (s_k, n_k - n)\} \subseteq S \times \{+1, 0, -1\}$ satisfies β, (ii) $\mathit{test} = \mathsf{first?}$ implies $n = 0$, and (iii) $\mathit{test} \in \Sigma$ implies $a_n = \mathit{test}$. Note that, similarly to an OCAP, a transition with move -1 is blocked if $n = 0$, i.e., if $\mathcal{T}$ is at the first position of the input word. A run is *accepting* if every infinite branch visits some accepting state from S_f infinitely often. The language of $\mathcal{T}$ is defined as $L(\mathcal{T}) = \{w \in \Sigma^\omega \mid \text{there exists an accepting run of } \mathcal{T} \text{ on } w\}$. The *non-emptiness problem* for A2As is to decide, given an A2A $\mathcal{T}$, whether $L(\mathcal{T}) \neq \emptyset$.

▶ **Theorem 6** ([28]). *The non-emptiness problem for A2As is in* PSPACE.

It is worth noting that [28] also uses two-wayness to simulate one-counter automata, but in a game-based setting (the latter is reflected by alternation).

Let us show how to build an A2A from an OCAP and a target state.

▶ **Lemma 7.** *Let $\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta)$ be an OCAP and $q_f \in Q$. There is an A2A $\mathcal{T} = (S, \Sigma, s_{\mathsf{in}}, \delta, S_f)$, with $\Sigma = \mathcal{X} \cup \{\square\}$, such that $L(\mathcal{T}) = \{w \in \mathcal{W}_{\mathcal{X}} \mid q_f$ is γ_w-reachable$\}$. Moreover, $|\mathcal{T}| = \mathcal{O}(|\mathcal{A}|^2)$.*

Proof. The states of $\mathcal{T}$ include Q (to simulate $\mathcal{A}$), a new initial state s_{in}, and some extra states, which will be introduced below.

Starting in s_{in} and at the first letter of the input word, the A2A $\mathcal{T}$ spawns several copies: $s_{\mathsf{in}} \xrightarrow{\square} (q_{\mathsf{in}}, 0) \wedge \bigwedge_{x\in\mathcal{X}} (\mathsf{search}(x), +1)$. The copy starting in q_{in} will henceforth simulate $\mathcal{A}$.

[1] One often considers a transition *function* $\delta : S \times \Sigma \times \{\mathsf{first?}, \neg\mathsf{first?}\} \to \mathbb{B}^+(S \times \{+1, 0, -1\})$, but a relation is more convenient for us.

Moreover, from the new state $\mathsf{search}(x)$, we will check that x occurs exactly once in the input word. This is accomplished using the transitions $\mathsf{search}(x) \xrightarrow{x} (\checkmark_x, +1)$, as well as $\mathsf{search}(x) \xrightarrow{y} (\mathsf{search}(x), +1)$ and $\checkmark_x \xrightarrow{y} (\checkmark_x, +1)$ for all $y \in \Sigma \setminus \{x\}$. Thus, $\checkmark_x$ signifies that x has been seen and must not be encountered again. Since the whole word has to be scanned, $\checkmark_x$ is visited infinitely often so that we set $\checkmark_x \in S_f$.

It remains to specify how $\mathcal{T}$ simulates $\mathcal{A}$. The underlying idea is very simple. A configuration $(q, v) \in \mathcal{C}_\mathcal{A}$ of $\mathcal{A}$ corresponds to the configuration/proof obligation (q, i) of $\mathcal{T}$ where position i is the $(v+1)$-th occurrence of $\square$ in the input parameter word. The simulation then proceeds as follows.

Increment/decrement: To mimic a transition $(q, +1, q') \in \Delta$, the A2A $\mathcal{T}$ simply goes to the next occurrence of $\square$ on the right hand side and enters q'. This is accomplished by several A2A-transitions: $q \xrightarrow{\square} (\mathsf{right}(q'), +1)$, $\mathsf{right}(q') \xrightarrow{x} (\mathsf{right}(q'), +1)$ for every $x \in \mathcal{X}$, and $\mathsf{right}(q') \xrightarrow{\square} (q', 0)$. Decrements are handled similarly, using states of the form $\mathsf{left}(q')$. Finally, $(q, 0, q') \in \Delta$ is translated to $q \xrightarrow{\square} (q', 0)$.

Equality test: A zero test $(q, \mathsf{zero}?, q') \in \Delta$ corresponds to $q \xrightarrow{\mathsf{first}?} (q', 0)$. To simulate a transition $(q, {=}x, q') \in \Delta$, the A2A spawns two copies. One goes from q to q' and stays at the current position. The other goes to the right and accepts if it sees x before hitting another $\square$-symbol. Thus, we introduce a new state $\mathsf{present}(x)$ and transitions $q \xrightarrow{\square} (q', 0) \wedge (\mathsf{present}(x), +1)$, $\mathsf{present}(x) \xrightarrow{x} \mathtt{true}$, and $\mathsf{present}(x) \xrightarrow{y} (\mathsf{present}(x), +1)$ for all $y \in \mathcal{X} \setminus \{x\}$. Note that there is *no* transition that allows $\mathcal{T}$ to read $\square$ in state $\mathsf{present}(x)$.

Inequality tests: To simulate a test of the form $>x$, we proceed similarly to the previous case. The A2A generates a branch in charge of verifying that the parameter x lies strictly to the left of the current position. Note that transitions with label $<x$ are slightly more subtle, as x has to be retrieved strictly *beyond* the next delimiter $\square$ to the right of the current position. More precisely, $(q, <x, q') \in \Delta$ translates to $q \xrightarrow{\square} (q', 0) \wedge (\mathsf{search}^+(x), +1)$. We stay in $\mathsf{search}^+(x)$ until we encounter the next occurrence of $\square$. We then go into $\mathsf{search}(x)$ (introduced at the beginning of the proof), which will be looking for an occurrence of x. Formally, we introduce $\mathsf{search}^+(x) \xrightarrow{y} (\mathsf{search}^+(x), +1)$ for all $y \in \mathcal{X} \setminus \{x\}$, and $\mathsf{search}^+(x) \xrightarrow{\square} (\mathsf{search}(x), +1)$.

In state q_f, we accept: $q_f \xrightarrow{\square} \mathtt{true}$. The only infinite branches in an accepting run are those that eventually stay in a state of the form $\checkmark_x$. Thus, we set $S_f = \{\checkmark_x \mid x \in \mathcal{X}\}$. It is not hard to see that $L(\mathcal{T}) = \{w \in \mathcal{W}_\mathcal{X} \mid q_f \text{ is } \gamma_w\text{-reachable}\}$. Note that $\mathcal{T}$ has a linear number of states and a quadratic number of transitions (some transitions of $\mathcal{A}$ are simulated by $\mathcal{O}(|\mathcal{X}|)$-many transitions of $\mathcal{T}$). ◀

A similar reduction takes care of Büchi reachability. Together with Theorem 6, we thus obtain the following:

▶ **Corollary 8.** OCAP-Reachability *and* OCAP-Büchi *are both in* PSPACE.

Note that we are using alternation only to a limited extent. We could also reduce reachability in OCAPs to non-emptiness of the intersection of several two-way automata. However, this would require a more complicated word encoding of parameter instantiations, which then have to include letters of the form $<x$ and $>x$.

3.2 OCAP-Reachability is in NP

We will now show that we can improve the upper bounds given in Corollary 8 to NP, which is then optimal: NP-hardness of both problems, even in the presence of a single parameter, can be proved using a straightforward reduction from the non-emptiness problem for nondeterministic two-way automata over finite words over a *unary* alphabet, which is NP-complete [17].

In a first step, we exploit our reduction from OCAPs to A2As to establish a bound on the parameter values. To solve the reachability problems, it will then be sufficient to consider parameter instantiations up to that bound. For the rest of this section, unless stated otherwise, we fix an OCAP $\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta)$ and a state $q_f \in Q$.

▶ **Lemma 9.** *There is $d \in 2^{\mathcal{O}(|\mathcal{A}|^4)}$ such that the following holds: If $\mathcal{A} \models \mathit{Reach}(q_f)$, then there is a parameter instantiation $\gamma : \mathcal{X} \to \mathbb{N}$ such that $\gamma(x) \leq d$ for all $x \in \mathcal{X}$ and q_f is γ-reachable.*

Proof. According to Lemma 7, there is an A2A $\mathcal{T} = (S, \mathcal{X} \cup \{\Box\}, s_{\mathsf{in}}, \delta, S_f)$ such that $L(\mathcal{T}) = \{w \in \mathcal{W}_{\mathcal{X}} \mid q_f \text{ is } \gamma_w\text{-reachable}\}$ and $|\mathcal{T}| = \mathcal{O}(|\mathcal{A}|^2)$. By [30], there is a (nondeterministic) Büchi automaton $\mathcal{B}$ such that $L(\mathcal{B}) = L(\mathcal{T})$ and $|\mathcal{B}| \in 2^{\mathcal{O}(|\mathcal{T}|^2)}$ (cf. also [8]).

Let $d = |\mathcal{B}|$. Suppose $\mathcal{A} \models \mathit{Reach}(q_f)$. This implies that $L(\mathcal{B}) \neq \emptyset$. But then, there must be a word $u \in \Sigma^*$ such that $|u| \leq |\mathcal{B}|$ ($|u|$ denoting the length of u) and $w = u\Box^\omega \in L(\mathcal{B})$. We have that q_f is γ_w-reachable and $\gamma_w(x) \leq |\mathcal{B}| = d$ for all $x \in \mathcal{X}$. ◀

As a corollary, we obtain that it is sufficient to consider bounded runs only. For a parameter instantiation $\gamma : \mathcal{X} \to \mathbb{N}$ and a bound $d \in \mathbb{N}$, we say that a *γ-run* $\rho = (q_0, v_0) \longrightarrow_\gamma (q_1, v_1) \longrightarrow_\gamma (q_2, v_2) \longrightarrow_\gamma \cdots$ is *d-bounded* if all its counter values $v_0, v_1, \ldots$ are at most d.

▶ **Corollary 10.** *There is $d \in 2^{\mathcal{O}(|\mathcal{A}|^4)}$ such that the following holds: If $\mathcal{A} \models \mathit{Reach}(q_f)$, then there is a parameter instantiation $\gamma : \mathcal{X} \to \mathbb{N}$ such that $\gamma(x) \leq d$ for all $x \in \mathcal{X}$ and q_f is reachable within a d-bounded γ-run.*

Proof. Consider $d \in 2^{\mathcal{O}(|\mathcal{A}|^4)}$ due to Lemma 9. A standard argument in OCAs with n states is that, for reachability, it is actually sufficient to consider runs up to some counter value in $\mathcal{O}(n^3)$ [25]. We can apply the same argument here to deduce, together with Lemma 9, that parameter and counter values can be bounded by $d + \mathcal{O}(|Q|^3)$. ◀

The algorithm that solves OCAP-Reachability in NP will guess a parameter instantiation γ and a maximal counter value in binary representation (thus, of polynomial size). It then remains to determine, in polynomial time, whether the target state is reachable under this guess. To this end, we will exploit the following lemma due to Galil [17].

Let $\mathcal{A} = (Q, q_{\mathsf{in}}, \Delta)$ be an OCA, $q, q' \in Q$, and $v, v' \in \mathbb{N}$. A run $(q_0, v_0) \longrightarrow (q_1, v_1) \longrightarrow \cdots \longrightarrow (q_{n-1}, v_{n-1}) \longrightarrow (q_n, v_n)$ of $\mathcal{A}$, $n \in \mathbb{N}$, is called a *(v, v')-run* if $\min\{v, v'\} < v_i < \max\{v, v'\}$ for all $i \in \{1, \ldots, n-1\}$. In other words, all intermediate configurations have counter values strictly between v and v'.

▶ **Lemma 11** ([17])**.** *The following problems can be solved in polynomial time:*

Input:	An OCA $\mathcal{A} = (Q, q_{\mathsf{in}}, \Delta)$, $q, q' \in Q$, and $v, v' \in \mathbb{N}$ given in binary.
Question 1:	Is there a (v, v')-run from (q, v) to (q', v') ?
Question 2:	Is there a (v, v')-run from (q, v) to (q', v) ?

Actually, [17] uses the polynomial-time algorithms stated in Lemma 11 to show that non-emptiness of two-way automata on finite words over a unary alphabet is in NP (cf. the

proof of the corresponding lemma on page 84 of [17], where the endmarkers of a given word correspond to v and v').

We can now deduce the following result, which is an important step towards model checking, but also of independent interest.

▶ **Theorem 12.** OCAP-Reachability *is* NP*-complete.*

Sketch of proof. Let $\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta)$ be an OCAP with $\mathcal{X} = \{x_1, \ldots, x_n\}$, and let $q_f \in Q$ be a state of $\mathcal{A}$. Without loss of generality, we consider reachability of the *configuration* $(q_f, 0)$ (since one can add a new target state and a looping decrement transition).

Our nondeterministic polynomial-time algorithm proceeds as follows. First, it guesses $d \in 2^{\mathcal{O}(|\mathcal{A}|^4)}$ due to Corollary 10 (note that the binary representation of d is of polynomial size with respect to $|\mathcal{A}|$), as well as some parameter instantiation γ satisfying $\gamma(x_i) \leq d$ for all $1 \leq i \leq n$ where each $d_i := \gamma(x_i)$ is represented in binary, too. We may assume $n \geq 1$ and that $d_0 < d_1 < d_2 < \cdots < d_n < d_{n+1}$, where $d_0 = 0$ and $d_{n+1} = d$ (otherwise, we can rename the parameters accordingly). Second, the algorithm checks that there exists a d-bounded γ-run of the form $(q_0, v_0) \longrightarrow_\gamma^* (q'_0, v_0) \longrightarrow_\gamma^* (q_1, v_1) \longrightarrow_\gamma^* (q'_1, v_1) \longrightarrow_\gamma^* \cdots \longrightarrow_\gamma^* (q_k, v_k) \longrightarrow_\gamma^* (q'_k, v_k)$ such that (letting $D = \{d_0, d_1, \ldots, d_n, d_{n+1}\}$)

(1) $(q_0, v_0) = (q_{\mathsf{in}}, 0)$ and $(q'_k, v_k) = (q_f, 0)$,
(2) $v_j \in D$ for all $j \in \{0, \ldots, k\}$,
(3) between (q_j, v_j) and (q'_j, v_j), the counter values always equal v_j, for all $j \in \{0, \ldots, k\}$,
(4) (strictly) between (q'_j, v_j) and (q_{j+1}, v_{j+1}), the counter values are always different from the values in D, for all $j \in \{0, \ldots, k-1\}$.

Note that we can assume $k \leq |Q| \cdot (|\mathcal{X}| + 2)$, since in every longer run, the exact same configuration is encountered at least twice.

Hence, to check whether there exists a finite initialized d-bounded γ-run $(q_{\mathsf{in}}, 0) \longrightarrow_\gamma^* (q_f, 0)$, it is sufficient to guess $2k$ configurations, where $k \leq |Q| \cdot (|\mathcal{X}| + 2)$, and to verify that these configurations contribute to constructing a run of the form described above. This can be checked in polynomial time: The case (3) is obvious, and (4) is due to Lemma 11. ◀

4 From OCAP-Reachability to SOCA-Model-Checking

This section is dedicated to model checking OCAs against flatLTL$^{\downarrow}$. In Section 4.1, we establish a couple of reductions from SOCA-MC(flatLTL$^{\downarrow}$) down to OCAP-Reachability, which allow us to conclude that the former problem is in NEXPTIME. Then, in Section 4.2, we provide a matching lower bound.

4.1 Upper Bounds

First, we use the fact that OCAP-Reachability is in NP (Theorem 12) to show that OCAP-Büchi is in NP, too:

▶ **Lemma 13.** OCAP-Büchi *is polynomial-time reducible to* OCAP-Reachability.

Proof. Let $\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta)$ be an OCAP with $\mathcal{X} = \{x_1, \ldots, x_n\}$ (without loss of generality, we suppose $n \geq 1$), and let $F \subseteq Q$. It is sufficient to give an algorithm that determines whether a particular state $q_f \in F$ can be repeated infinitely often. Thus, we construct an OCAP $\mathcal{A}'$, with a distinguished state $\hat{s}$, such that $\mathcal{A} \models \mathit{Reach}^{\omega}(q_f)$ iff $\mathcal{A}' \models \mathit{Reach}(\hat{s})$.

To this end, we first define an OCA $\mathcal{M}$, which is obtained from $\mathcal{A}$ by (i) removing all transitions whose labels are of the form $\mathsf{zero}?$, $=x$, or $<x$, for some $x \in \mathcal{X}$, and (ii) replacing

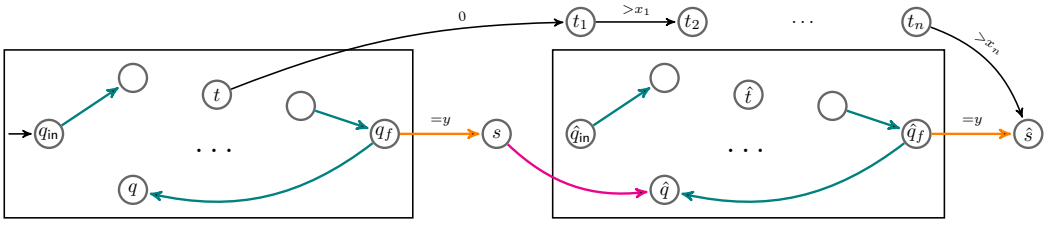

Figure 2 The OCAP $\mathcal{A}'$ constructed from $\mathcal{A}$ in the proof of Lemma 13.

all transition labels $>x$ (for some $x \in \mathcal{X}$) by 0. Then, we compute the set $T \subseteq Q$ of states t of $\mathcal{M}$ from which there is an infinite run starting in $(t, 0)$ and visiting q_f infinitely often. This can be done in NLOGSPACE [9].

Now, we obtain the desired OCAP $\mathcal{A}' = (Q', \mathcal{X}', q_{\text{in}}, \Delta')$ as follows. The set of states is $Q' = Q \uplus \{\hat{q} \mid q \in Q\} \uplus \{s, \hat{s}, t_1, \ldots, t_n\}$. That is, we introduce a copy $\hat{q}$ for every $q \in Q$ as well as fresh states $s, \hat{s}, t_1, \ldots t_n$. Recall that $\hat{s}$ will be the target state of the reachability problem we reduce to. Moreover, $\mathcal{X}' = \mathcal{X} \uplus \{y\}$ where y is a fresh parameter. It remains to define the transition relation Δ', which includes Δ as well as some new transitions (cf. Figure 2). Essentially, there are two cases to consider:

(1) First, q_f may be visited infinitely often in $\mathcal{A}$, and infinitely often along with the same counter value. To take this case into account, we use the new parameter y and a new transition $(q_f, =y, s)$, which allows $\mathcal{A}'$ to "store" the current counter value in y. From s, we then enter and simulate the copy of $\mathcal{A}$, i.e., we introduce transitions $(s, \mathsf{op}, \hat{q})$ for all $(q_f, \mathsf{op}, q) \in \Delta$, as well as $(\hat{q}_1, \mathsf{op}, \hat{q}_2)$ for all $(q_1, \mathsf{op}, q_2) \in \Delta$. If, in the copy, $\hat{q}_f$ is visited along with the value stored in y, we may thus enter $\hat{s}$.

(2) Suppose, on the other hand, that a γ-run ρ of $\mathcal{A}$ visits q_f infinitely often, but not infinitely often with the same counter value. If ρ visits some counter value $v \leq \max\{\gamma(x) \mid x \in \mathcal{X}\}$ infinitely often, then it necessarily contains a subrun of the form $(q, v) \longrightarrow_\gamma^* (q_f, v') \longrightarrow_\gamma^* (q, v)$ for some $q \in Q$ and $v' \in \mathbb{N}$. But then, there is an infinite γ-run that visits (q_f, v') infinitely often so that case (1) above applies. Otherwise, ρ will eventually stay strictly above $\max\{\gamma(x) \mid x \in \mathcal{X}\}$. Thus, we add the following transitions to Δ', which will allow $\mathcal{A}'$ to move from $t \in T$ to $\hat{s}$ provided the counter value is greater than the maximal parameter value: $(t, 0, t_1)$ for all $t \in T$, as well as $(t_1, >x_1, t_2), (t_2, >x_2, t_3), \ldots, (t_n, >x_n, \hat{s})$. ◀

From Theorem 12 and Lemma 13, we obtain the exact complexity of OCAP-Büchi (the lower bound is by a straightforward reduction from OCAP-Reachability).

▶ **Corollary 14.** OCAP-Büchi *is* NP*-complete.*

The link between OCA-MC(flatLTL$^{\downarrow}$) and OCAP-Büchi is due to [13]:

▶ **Lemma 15** ([13])**.** *Let $\mathcal{A}$ be an OCA and $\varphi \in$ flatLTL$^{\downarrow}$ be a sentence. One can compute, in exponential time, an OCAP $\mathcal{A}' = (Q, \mathcal{X}, q_{\text{in}}, \Delta)$ (of exponential size) and a set $F \subseteq Q$ such that $\mathcal{A} \models_\exists \varphi$ iff $\mathcal{A}' \models \mathit{Reach}^\omega(q_f)$ for some $q_f \in F$.*

Actually, the construction from [13] can be easily extended to handle register tests of the form $<r$ and $>r$. However, the restriction to flatLTL$^{\downarrow}$ is crucial. It allows one to assume that a register is written at most once so that, in the OCAP, it can be represented as a parameter. In particular, $\mathcal{X}$ is the set of registers that occur in φ. From Corollary 14 and Lemma 15, we now deduce our first model-checking result:

▶ **Corollary 16.** OCA-MC(flatLTL$^{\downarrow}$) *is in* NEXPTIME.

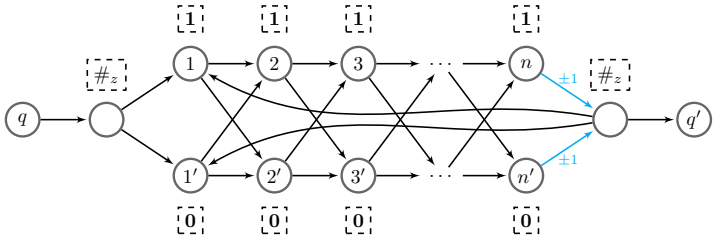

Figure 3 The OCA-gadget for simulating a SOCA-transition (q, z, q') with binary update z. The new propositions are depicted in dashed boxes. States $1, \ldots, n, 1', \ldots, n'$, where $n = \mathit{bits}(z)$, represent the bits needed to encode z in binary. At the transitions originating from n and n', the counter is updated by $+1$ or -1, depending on whether z is positive or negative, respectively.

$$\begin{array}{ccccccccccccc} \mu(q) & \#_6 & \mathbf{100} & \#_6 & \mathbf{010} & \#_6 & \mathbf{110} & \#_6 & \mathbf{001} & \#_6 & \mathbf{101} & \#_6 & \mathbf{011} & \#_6 & \mu(q') \\ \uparrow & & & & \uparrow & & & & & & \uparrow & & \uparrow & & \\ \mathit{first}(\#_6) & & & & \mathit{suff}^{=}_{\#_6} & & & & & & \mathit{last}^{-}(\#_6) & & \mathit{last}(\#_6) & & \end{array}$$

Figure 4 A counting sequence

However, it turns out that SOCA model checking is no harder than OCA model checking. The reason is that succinct updates can be encoded in small LTL formulas.

▶ **Lemma 17.** SOCA-MC(flatLTL$^{\downarrow}$) *is polynomial-time reducible to* OCA-MC(flatLTL$^{\downarrow}$).

Proof. Let $\mathcal{A} = (Q, q_{\mathsf{in}}, \Delta, \mu)$ be a SOCA and $\varphi \in \text{flatLTL}^{\downarrow}$ be a sentence. Without loss of generality, we assume that φ is in negation normal form, i.e., negation is applied only to atomic propositions. This is thanks to the logical equivalences $\neg(\varphi_1 \,\mathsf{U}\, \varphi_2) \equiv \neg\varphi_1 \,\mathsf{R}\, \neg\varphi_2$ and $\neg(\varphi_1 \,\mathsf{R}\, \varphi_2) \equiv \neg\varphi_1 \,\mathsf{U}\, \neg\varphi_2$. We construct an OCA $\mathcal{A}' = (Q', q'_{\mathsf{in}}, \Delta', \mu')$ and a sentence $\varphi' \in \text{flatLTL}^{\downarrow}$ of polynomial size such that $\mathcal{A} \models_{\exists} \varphi$ iff $\mathcal{A}' \models_{\exists} \varphi'$.

Let $Z = \{z \mid (q, z, q') \in \Delta \cap (Q \times \mathbb{Z} \times Q) \text{ with } |z| \geq 2\}$ and let $\Lambda = \{\#_z \mid z \in Z\} \cup \{\mathbf{0}, \mathbf{1}\}$ be a set of fresh propositions. To obtain $\mathcal{A}'$ from $\mathcal{A}$, we replace every transition $(q, z, q') \in \Delta \cap (Q \times Z \times Q)$ by the gadget depicted in Figure 3. The gadget implements a binary counter generating sequences (of sets of propositions) of the form depicted in Figure 4, for $z = 6$. We assume that the least significant bit is on the left. To make sure that the binary counter works as requested, we define an LTL formula *Counter*.

Towards its definition, let us first introduce some notation and abbreviations. For $z \in Z$, let $\mathit{bits}(z)$ denote the number of bits needed to represent the binary encoding of $|z|$. For example, $\mathit{bits}(6) = 3$. For $i \in \{1, \ldots, \mathit{bits}(z)\}$, we let $\mathit{bit}_i(z)$ denote the i-th bit in that encoding. For example, $(\mathit{bit}_1(6), \mathit{bit}_2(6), \mathit{bit}_3(6)) = (\mathbf{0}, \mathbf{1}, \mathbf{1})$. In the following, we write Λ for $\bigvee_{\gamma \in \Lambda} \gamma$ and $\neg\Lambda$ for $\bigwedge_{\gamma \in \Lambda} \neg\gamma$. For an illustration of the following LTL formulas, we refer to Figure 4. Formula $\mathit{first}(\#_z) = \neg\Lambda \wedge \mathsf{X}\#_z$ holds right before a first delimiter symbol. Similarly, $\mathit{last}(\#_z) = \#_z \wedge \mathsf{X}\neg\Lambda$ identifies all last delimiters and $\mathit{last}^{-}(\#_z) = \#_z \wedge \mathsf{X}\big((\mathbf{0} \vee \mathbf{1}) \,\mathsf{U}\, \mathit{last}(\#_z)\big)$ all second-last delimiters. Finally, we write $\curvearrowright_z \psi$ for $\mathsf{X}^{\mathit{bits}(z)+1}\psi$.

Now, $\mathit{Counter} = \mathit{Init} \wedge \mathit{Fin} \wedge \mathit{Inc} \wedge \mathit{Exit}$ is the conjunction of the following formulas:

- *Init* says that the first binary number is always one:

$$\mathit{Init} = \textstyle\bigwedge_{z \in Z} \mathsf{G}\big(\mathit{first}(\#_z) \;\rightarrow\; \mathsf{X}^2(\mathbf{1} \;\wedge\; \mathsf{X}(\mathbf{0} \,\mathsf{U}\, \#_z))\big)$$

- *Fin* says that the last value represents $|z|$:

$$\mathit{Fin} = \textstyle\bigwedge_{z \in Z} \mathsf{G}\big(\mathit{last}^{-}(\#_z) \;\rightarrow\; \bigwedge_{i=1}^{\mathit{bits}(z)} \mathsf{X}^i\, \mathit{bit}_i(z)\big)$$

- *Inc* implements the increments:

$$Inc = \bigwedge_{z\in Z} \mathsf{G}\left(\begin{array}{rl} & \big(\#_z \wedge \neg last^-(\#_z) \wedge \neg last(\#_z)\big) \\ \rightarrow & \mathsf{X}\big((\mathbf{1}\wedge \curvearrowright_z \mathbf{0})\,\mathsf{U}\,(\mathbf{0}\wedge\curvearrowright_z\mathbf{1}\wedge \mathsf{X}\, suff_z^{=})\big)\end{array}\right)$$

  where $suff_z^{=} = \big((\mathbf{0}\wedge\curvearrowright_z\mathbf{0})\vee(\mathbf{1}\wedge\curvearrowright_z\mathbf{1})\big)\,\mathsf{U}\,\#_z$ verifies that the suffixes (w.r.t. the current position) of the current and the following binary number coincide.
- *Exit* states that we do not loop forever in the gadget from Figure 3:

$$Exit = \bigwedge_{z\in Z} \mathsf{G}\big(first(\#_z) \;\rightarrow\; \mathsf{F}\; last(\#_z)\big)$$

Note that the gadget makes sure that, in between two delimiters $\#_z$, there are exactly $bits(z)$-many bits.

Now, we set $\varphi' = [\varphi] \wedge Counter$. Here, $[\varphi]$ simulates φ on all positions that do not contain propositions from Λ. It is inductively defined as follows:

$$[p] = p \qquad [\neg p] = \neg p \qquad [\bowtie r] = \bowtie r \qquad [\downarrow_r \psi] = \downarrow_r [\psi]$$

$$[\psi_1 \vee \psi_2] = [\psi_1] \vee [\psi_2] \qquad [\psi_1 \wedge \psi_2] = [\psi_1] \wedge [\psi_2] \qquad [\mathsf{X}\varphi] = \mathsf{X}\big(\Lambda\,\mathsf{U}\,(\neg\Lambda \wedge [\varphi])\big)$$

$$[\psi_1\,\mathsf{U}\,\psi_2] = (\neg\Lambda \rightarrow [\psi_1])\,\mathsf{U}\,(\neg\Lambda\wedge[\psi_2]) \qquad [\psi_1\,\mathsf{R}\,\psi_2] = (\neg\Lambda\wedge[\psi_1])\,\mathsf{R}\,(\neg\Lambda\rightarrow[\psi_2])$$

Note that, since φ was required to be in negation normal form, φ' is indeed in flatLTL$^{\downarrow}$. ◀

Corollary 16 and Lemma 17 imply the NEXPTIME upper bound for SOCA model checking:

▶ **Corollary 18.** SOCA-MC(flatLTL$^{\downarrow}$) *is in* NEXPTIME.

4.2 Lower Bounds

This subsection presents a matching lower bound for model checking flatLTL$^{\downarrow}$. We first state NEXPTIME-hardness of OCAP-MC(LTL), which we reduce, in a second step, to OCA-MC(flatLTL$^{\downarrow}$).

▶ **Theorem 19.** OCAP-MC(LTL) *and* SOCAP-MC(LTL) *are* NEXPTIME*-complete.*

Proof. NEXPTIME-hardness of SOCAP-MC(LTL) is due to [18], where Göller et al. show NEXPTIME-hardness of LTL model checking OCAs with parametric and succinct *updates* [18]. Their proof can be adapted in a straightforward way: In Figure 3 of [18], we replace the parametric update $+x$ by an equality test $=x$ and add a self-loop to the start state with label $+1$. We remark that this lower bound already holds for SOCAPs that use only one parameter x and all parameterized tests are of the form $=x$.

Moreover, SOCAP-MC(LTL) can be reduced to OCAP-MC(LTL) in polynomial time, using the construction from Lemma 17.

Membership of OCAP-MC(LTL) in NEXPTIME is by a standard argument. The given LTL formula can be translated into a Büchi automaton of exponential size. Then, we check non-emptiness of its product with the given OCAP using Corollary 14. ◀

▶ **Lemma 20.** OCA-MC(flatLTL$^{\downarrow}$) *and* SOCA-MC(flatLTL$^{\downarrow}$) *are* NEXPTIME*-hard.*

Proof. We present a reduction from OCAP-MC(LTL), which is NEXPTIME-complete by Theorem 19. Let $\mathcal{A} = (Q, \mathcal{X}, q_{\mathsf{in}}, \Delta, \mu)$ be an OCAP and let φ be an LTL formula. We define an OCA $\mathcal{A}' = (Q', q'_{\mathsf{in}}, \Delta', \mu')$ and a sentence $\varphi' \in$ flatLTL$^{\downarrow}$ such that $\mathcal{A} \models_{\exists} \varphi$ iff $\mathcal{A}' \models_{\exists} \varphi'$.

Without loss of generality, by [18] (cf. proof of Theorem 19), we may assume that $\mathcal{A}$ uses only one parameter x and that all parameterized tests are of the form $=x$.

The idea is as follows. A new initial state q'_{in}, in which only a fresh proposition *freeze* holds, will allow $\mathcal{A}'$ to go to an arbitrary counter value, say, v. The flatLTL$^{\downarrow}$ formula of the form $\mathsf{F}\big(\mathit{freeze} \wedge \downarrow_r \psi\big)$ stores v, which will henceforth be interpreted as parameter x. Hereby, formula ψ makes sure that, whenever $\mathcal{A}$ performs an equality check ${=}x$, the current counter value coincides with v. To do so, we create two copies of the original state space: $Q \times \{0,1\}$. A transition $(q, {=}x, q')$ of $\mathcal{A}$ is then simulated, in $\mathcal{A}'$, by a 0-labeled transition to $(q', 1)$. States of the latter form are equipped with a fresh proposition $p_{=x}$ indicating that the current counter value has to coincide with the contents of r. Thus, we can set $\psi = \mathsf{G}(p_{=x} \to {=}r)$.

Formally, $\mathcal{A}'$ is given as follows. We let $Q' = (Q \times \{0,1\}) \uplus \{q'_{\mathsf{in}}\}$ and

$$\begin{aligned}\Delta' = \quad & \{(q'_{\mathsf{in}}, \mathsf{op}, q'_{\mathsf{in}}) \mid \mathsf{op} \in \{+1,-1\}\} \;\cup\; \{(q'_{\mathsf{in}}, \mathsf{zero?}, (q_{\mathsf{in}}, 0))\} \\ \cup \quad & \{((q,b), 0, (q,1)) \mid (q, {=}x, q') \in \Delta \text{ and } b \in \{0,1\}\} \\ \cup \quad & \{((q,b), \mathsf{op}, (q,0)) \mid (q, \mathsf{op}, q') \in \Delta \setminus (Q \times \{{=}x\} \times Q) \text{ and } b \in \{0,1\}\}\,.\end{aligned}$$

Moreover, we set $\mu'(q'_{\mathsf{in}}) = \{\mathit{freeze}\}$ as well as $\mu'((q,0)) = \mu(q)$ and $\mu'((q,1)) = \mu(q) \cup \{p_{=x}\}$ for all $q \in Q$. Finally, $\varphi' = \mathsf{F}\big(\mathit{freeze} \wedge \downarrow_r \mathsf{G}(p_{=x} \to {=}r)\big) \wedge \big(\mathit{freeze}\,\mathsf{U}\,(\neg \mathit{freeze} \wedge \varphi)\big)$. Note that the subformula $\mathit{freeze}\,\mathsf{U}\,(\neg \mathit{freeze} \wedge \varphi)$ makes sure that φ is satisfied as soon as we enter the original initial state q_{in} (actually, $(q_{\mathsf{in}}, 0)$). ◀

We conclude stating the exact complexity of model checking (S)OCAs against flatLTL$^{\downarrow}$:

▶ **Theorem 21.** OCA-MC(flatLTL$^{\downarrow}$) *and* SOCA-MC(flatLTL$^{\downarrow}$) *are* NEXPTIME*-complete.*

5 Conclusion

In this paper, we established the precise complexity of model checking OCAs and SOCAs against flat freeze LTL. To do so, we established a tight link between OCAPs and alternating two-way automata over words. Exploiting alternation further, it is likely that we can obtain positive results for one-counter games with parameterized tests. Another interesting issue for future work concerns model checking OCAPs, which, in this paper, is defined as the following question: Are there a parameter instantiation γ and a γ-run that satisfies the given formula? In fact, it also makes perfect sense to ask whether there is such a run for *all* parameter instantiations, in particular when requiring that all system runs satisfy a given property. We believe that our techniques can be used also in that case.

Although OCAPs have mainly been used for deciding a model-checking problem for OCAs, they constitute a versatile tool as well as a natural stand-alone model. In fact, parameterized extensions of a variety of system models have been explored recently, among them OCAs with parameterized *updates* [19, 22], and parametric timed automata [2, 5] where clocks can be compared with parameters (similarly to parameterized tests in OCAPs). Our results may have an impact on those models, too, thus beyond their application to model checking flat freeze LTL of OCAs: The (existential) non-emptiness problem for parametric timed automata with a single parametric clock (1PTAs) was first studied and solved by Alur, Henzinger, and Vardi, who provided a nonelementary decision procedure [2]. Recently, Bundala and Ouaknine improved this significantly to 2NEXPTIME [5]. They provide a polynomial-time reduction to the reachability problem for OCAPs where, in addition to parameters, the counter values can be compared with constants $c \in \mathbb{N}$ encoded in binary. Altogether, they obtain a 2NEXPTIME algorithm for the non-emptiness problem for 1PTAs. Therefore, it will be worthwhile to study whether our NP upper bound for reachability in OCAPs applies to extended OCAPs, too (however, with *binary updates*, the problem is already PSPACE-hard [14]). This would yield an optimal NEXPTIME algorithm for non-emptiness in 1PTAs.

References

1 S. Akshay, B. Bollig, P. Gastin, M. Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. *Fundamenta Informaticae*, 130(4):377–407, 2014.

2 R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proceedings of STOC'93*, pages 592–601. ACM, 1993.

3 M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011.

4 P. Bouyer, N. Markey, J. Ouaknine, and J. Worrell. On expressiveness and complexity in real-time model checking. In *Proceedings of ICALP'08, Part II*, volume 5126 of *LNCS*, pages 124–135. Springer, 2008.

5 D. Bundala and J. Ouaknine. On parametric timed automata and one-counter machines. *Inf. Comput.*, 253:272–303, 2017.

6 T. Cachat. Two-way tree automata solving pushdown games. In *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *LNCS*, pages 303–317. Springer, 2002.

7 H. Comon and V. Cortier. Flatness is not a weakness. In *Proceedings of CSL'00*, volume 1862 of *LNCS*, pages 262–276. Springer, 2000.

8 C. Dax and F. Klaedtke. Alternation elimination by complementation. In *Proceedings of LPAR'08*, volume 5330 of *LNCS*, pages 214–229. Springer, 2008.

9 S. Demri and R. Gascon. The effects of bounding syntactic resources on presburger LTL. *J. Log. Comput.*, 19(6):1541–1575, 2009.

10 S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

11 S. Demri, R. Lazic, and D. Nowak. On the freeze quantifier in constraint LTL: decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.

12 S. Demri, R. Lazic, and A. Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theor. Comput. Sci.*, 411(22-24):2298–2316, 2010.

13 S. Demri and A Sangnier. When model-checking freeze LTL over counter machines becomes decidable. In *Proceedings of FoSSaCS'10*, volume 6014 of *LNCS*, pages 176–190. Springer, 2010.

14 J. Fearnley and M. Jurdzinski. Reachability in two-clock timed automata is pspace-complete. In *Proceedings of ICALP'13, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2013.

15 S. Feng, M. Lohrey, and K. Quaas. Path checking for MTL and TPTL over data words. In *Proceedings of DLT'15*, volume 9168 of *LNCS*, pages 326–339. Springer, 2015.

16 T. French. Quantified propositional temporal logic with repeating states. In *Proceedings of TIME-ICTL'03*, pages 155–165. IEEE Computer Society, 2003.

17 Z. Galil. Hierarchies of complete problems. *Acta Inf.*, 6:77–88, 1976.

18 S. Göller, C. Haase, J. Ouaknine, and J. Worrell. Model checking succinct and parametric one-counter automata. In *Proceedings of ICALP'10, Part II*, volume 6199 of *LNCS*, pages 575–586. Springer, 2010.

19 S. Göller, C. Haase, J. Ouaknine, and J. Worrell. Branching-time model checking of parametric one-counter automata. In *Proceedings of FOSSACS'12*, volume 7213 of *LNCS*, pages 406–420. Springer, 2012.

20 S. Göller and M. Lohrey. Branching-time model checking of one-counter processes and timed automata. *SIAM J. Comput.*, 42(3):884–923, 2013.

21 C. Haase. *On the complexity of model checking counter automata*. PhD thesis, University of Oxford, 2012.

22 C. Haase, S. Kreutzer, J. Ouaknine, and J. Worrell. Reachability in succinct and parametric one-counter automata. In *Proceedings of CONCUR'09*, volume 5710 of *LNCS*, pages 369–383. Springer, 2009.

23 P. Hofman, S. Lasota, R. Mayr, and P. Totzke. Simulation problems over one-counter nets. *Logical Methods in Computer Science*, 12(1), 2016.

24 O. Kupferman and M. Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *Proceedings of CAV'00*, volume 1855 of *LNCS*, pages 36–52. Springer, 2000.

25 P. Lafourcade, D. Lugiez, and R. Treinen. Intruder deduction for AC-like equational theories with homomorphisms. In *Proceedings of RTA'05*, volume 3467 of *LNCS*, pages 308–322. Springer, 2005.

26 A. Lechner, R. Mayr, J. Ouaknine, A. Pouly, and J. Worrell. Model checking flat freeze LTL on one-counter automata. In *Proceedings of CONCUR'16*, volume 59 of *LIPIcs*, pages 29:1–29:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

27 A. Lisitsa and I. Potapov. Temporal logic with predicate lambda-abstraction. In *Proceedings of TIME'05*, pages 147–155. IEEE Computer Society, 2005.

28 O. Serre. Parity games played on transition graphs of one-counter processes. In *Proceedings of FoSSaCS'06*, volume 3921 of *LNCS*, pages 337–351. Springer, 2006.

29 L. G. Valiant and M. S. Paterson. Deterministic one-counter automata. *Journal of Computer and System Sciences*, 10(3):340–350, 1975.

30 M. Y. Vardi. Reasoning about the past with two-way automata. In *Proceedings of ICALP'98*, volume 1443 of *LNCS*, pages 628–641. Springer, 1998.