Model Checking Recursive Programs with Numeric Data Types

Matthew Hague and Anthony Widjaja Lin

Oxford University Computing Laboratory

Abstract. Pushdown systems (PDS) naturally model sequential recursive programs. Numeric data types also often arise in real-world programs. We study the extension of PDS with unbounded counters, which naturally model numeric data types. Although this extension is Turing-powerful, reachability is known to be decidable when the number of reversals between incrementing and decrementing modes is bounded. In this paper, we (1) pinpoint the decidability/complexity of reachability and linear/branching time model checking over PDS with reversal-bounded counters (PCo), and (2) experimentally demonstrate the effectiveness of our approach in analysing software. We show reachability over PCo is NP-complete, while LTL is coNEXP-complete (coNP-complete for fixed formulas). In contrast, we prove that EF-logic over PCo is undecidable. Our NP upper bounds are by a direct poly-time reduction to satisfaction over existential Presburger formulas, allowing us to tap into highly optimized solvers like Z3. Although reversal-bounded analysis is incomplete for PDS with unbounded counters in general, our experiments suggest that some intricate bugs (e.g. from Linux device drivers) can be discovered with a small number of reversals. We also pinpoint the decidability/complexity of various extensions of PCo, e.g., with discrete clocks.

1 Introduction

Pushdown systems (PDS) are natural abstractions of sequential programs with unbounded recursions whose verification problems have been extensively studied (cf. [4, 12, 35]). In addition to recursions, numerical data types commonly arise in real-world programs. A standard approach to these potentially infinite domains is to map them into abstract domains that are more amenable to analysis (e.g. finite ones like $\{\text{Pos}, \text{Neg}, \text{Zero}\}$, or infinite ones expressed by intervals, difference bound matrices, polyhedra, etc.). For other types of program analysis, it is also common to simply ignore numerical data types. For a comprehensive treatment of these techniques, and others, the reader is referred to the survey [11].

In this paper, we study a different approach. Motivated by the success of pushdown systems (or similar models like boolean programs) in software model checking (cf. [2, 3, 29]), we aim to investigate extensions of pushdown systems with numerical data types and preserve nice properties, such as decidability and good complexity. A clean and simple approach is to enrich pushdown systems with unbounded counters, which can be incremented/decremented and tested for zero. Unfortunately, this model is Turing-powerful, even without the stack.

One way to retain decidability of reachability is to impose an upper bound r on the number of reversals between incrementing and decrementing modes for each counter (cf. [9, 19]). In fact, decidability holds even if discrete clocks are added to the model [9]. On the other hand, the complexity of reachability over these models is still open; a simple analysis of [9, 19] yields at least double exponential-time complexity for their algorithms. Recently, the authors of [13] observed that replacing the use of Parikh's Theorem [24] in [19] by a recently improved version of [34] immediately yields an NP procedure for this reachability problem (without clocks) for *fixed* numbers of reversals and counters, yielding only an NEXP procedure in general. Furthermore, the decidability of linear/branching time model checking over these models is also unknown.

There are at least two potential applications of reversal-bounded model checking (cf. [9, 20]). First, it could be used as a sound but *incomplete* verification technique for the case of unbounded reversals. Despite this incompleteness, results in bounded model checking suggest that "shallow" bugs are common in practice (cf. [11]). Clearly, reversal-bounded model checking is an infinite-state generalization of bounded model checking over counter systems. A second application is the use of reversal-bounded counters for tracking the number of times certain actions have been executed to reach the current configuration. For example, we can check the existence of a computation path in a recursive program where the number of invocations for the functions f_1, f_2, f_3, and f_4 are the same. Similar counting properties (and their model checking problems) have been studied in many other contexts (cf. [22] and references therein).

Contributions. We begin by studying pushdown systems enriched with reversal-bounded counters, which can be compared against and incremented by constants given in binary, but without clocks (PCo). This model is more general than the model studied in [13, 19], which allows only counters to be compared against 0 and incremented by $\{-1, 0, 1\}$, though at the cost of an exponential blow-up (cf. [20]) our model can be translated into their model. Our main contributions are (1) to pinpoint the decidability/complexity of reachability and linear/branching time model checking over PCo, and (2) experimentally demonstrate the effectiveness of our approach in the analysis of software.

We show that reachability over PCo is NP-complete, while LTL model checking is coNEXP-complete (coNP-complete for fixed formulas). In contrast, we prove that model checking EF-logic over PCo is undecidable. All of our lower bounds hold already for PDS with one 1-reversal counter wherein numeric constants, which can be either be compared against or used to increment/decrement counters, are restricted to 0 or 1. Our NP upper bounds are established by a direct poly-time reduction to satisfaction over existential Presburger formulas, allowing us to tap into highly optimized solvers like Z3 [10]. This reduction also permits additional constraints on the number of actions executed and the values of the counters at the end of the run (also specified in existential Presburger arithmetic) without further computation overhead in the reduction. We have implemented our algorithm and use it to analyse several examples, including some derived from memory management issues in Linux device drivers. Although reversal-

bounded analysis is only complete up to the bound on the number of reversals, our experiments suggest that many subtle bugs manifest themselves even within a small number of reversals, which our tool can detect reasonably fast. Without increasing complexity, our algorithm can also check whether a given PDS $\mathcal{P}$ with unbounded counters is r-reversal bounded, for a given input $r \in \mathbb{N}$; note that this is not the same as deciding whether $\mathcal{P}$ is reversal-bounded, which is undecidable [14]. In the case when $\mathcal{P}$ is r-reversal bounded, our technique gives a complete coverage of the infinite state space, which suggests the usefulness of our technique in proving correctness as well as finding bugs.

We then study the extension of PCo with discrete clocks (PCC). We show that LTL model checking over PCC is still coNEXP-complete, but hardness holds even for a fixed formula. Similarly, we show that reachability over PCC is NEXP-complete. We also show that, without reversal-bounded counters, model checking EF-logic over PCC is EXPSPACE-complete even for a fixed formula.

Related work. The complexities of most standard model checking problems over pushdown systems are well-understood. In relation to our results, we mention that LTL model checking over PDS is EXP-complete and is P-complete for fixed formulas [4] (the latter is also the complexity for reachability), whereas model checking EF-logic is PSPACE-complete [4, 35]. Therefore, adding reversal-bounded counters yields computationally harder problems in both cases.

Over reversal-bounded counter systems (without stack), reachability is NP-complete but becomes NEXP-complete when the number of reversals is given in binary [18]. On the other hand, when the numbers of reversals and counters are fixed, the problem is solvable in P [16]. The techniques developed by [16, 18], which reason about the maximal counter values, are very different to our techniques, which exploit the connection to Parikh images of pushdown automata (first explicated in Ibarra's original paper [19] though not in a way that gives optimal complexity or practical algorithm). For LTL model checking, the problem is solvable in EXP even in the presence of discrete clocks [31], whereas EF-logic model checking is still decidable but becomes undecidable for CTL [31].

For discrete-timed systems, reachability is known to be PSPACE-complete [1], where hardness holds already for three clocks [8]. Using region graph constructions [1], LTL model checking and EF-logic can also be easily shown to be PSPACE-complete. Note that we do not consider timed logics (cf. [5]). The complexities of pushdown systems with clocks have also been studied, e.g., [7].

Organization. §2 contains preliminaries. In §3, we define the basic model PCC that we study. §4 and §5, contain upper and lower bounds for model checking PCo. In §6, we extend our results to PCC. Experimental results are given in §7. Other extensions and future work are given in §8. Due to the space limit, some proofs are in the full version from the project page http://www.cs.ox.ac.uk/recount.

2 Preliminaries

Transition systems. An *action alphabet* ACT is a finite nonempty set of *actions*. A *transition system* over ACT is a tuple $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \mathsf{ACT}} \rangle$, where S

is a set of *configurations* and each $\rightarrow_a \subseteq S \times S$ is an a-labeled *transition relation* containing the set of all a-labeled *transitions*. We use $\rightarrow$ to denote the union of all $\rightarrow_a$. A *(computation) path* in $\mathfrak{S}$ is a finite or infinite sequence $\pi = \alpha_0 \rightarrow_{a_1} \alpha_1 \rightarrow_{a_2} \ldots$ such that $\alpha_i \in S$, $\alpha_i \rightarrow_{a_{i+1}} \alpha_{i+1}$ and $a_i \in \mathsf{ACT}$ for each i. If π is finite, let $\mathbf{last}(\pi)$ denote the last configuration in π. In this case, $a_1 a_2 \ldots$ is said to be a *(finite) trace in* $\mathfrak{S}$ *from* α_0 *to* $\boldsymbol{last}(\pi)$. **Automata** We assume familiarity with automata theory. In particular, nondeterministic Büchi automata (NBWA), cf. [32], and pushdown automata (PDA), cf. [27]. For an NBWA $\mathcal{A}$, we denote by $\mathcal{L}(\mathcal{A})$ the language recognized by $\mathcal{A}$. Similarly, given a PDA we also write $\mathcal{L}(\mathcal{P})$ for the language $\mathcal{P}$ recognizes.

Parikh images Given an alphabet $\Sigma = \{a_1, \ldots, a_k\}$ and a word $w \in \Sigma^*$, we write $\mathbb{P}(w)$ to denote a tuple with $|\Sigma|$ entries where the ith entry counts the number of occurrences of a_i in w. Given a language $\mathcal{L} \subseteq \Sigma^*$, we write $\mathbb{P}(\mathcal{L})$ to denote the set $\{\mathbb{P}(w) : w \in \mathcal{L}\}$. We say that $\mathbb{P}(\mathcal{L})$ is the *Parikh image* of $\mathcal{L}$.

Logic The syntax of LTL (cf. [17, 31, 32]) over ACT is given by: $\varphi, \varphi' := a$ ($a \in \mathsf{ACT}$) $\mid \neg\varphi \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi' \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi'$. Given an ω-word $w \in \mathsf{ACT}^\omega$ and an LTL formula φ over ACT, we define the satisfaction relation $w \models \varphi$ in the standard way. Write $[\![\varphi]\!]$ for all $w \in \mathsf{ACT}^\omega$ such that $w \models \varphi$. EF-logic (over ACT) is a fragment of CTL (cf. [17, 31]) with the syntax $\varphi, \psi := \top \mid \neg\varphi \mid \varphi \vee \psi \mid \langle a \rangle \varphi$ ($a \in \mathsf{ACT}$) $\mid \mathsf{EF}\varphi$. Given an EF formula φ, a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \mathsf{ACT}} \rangle$ and $s \in S$, we may define $\mathfrak{S}, s \models \varphi$ in the standard way.

Presburger formulas Presburger formulas are first-order formulas over natural numbers with addition. Here, we use extended existential Presburger formulas $\exists x_1, \ldots, x_k . \varphi$ where φ is a boolean combination of expressions $\sum_{i=1}^{k} a_i x_i \sim b$ for constants $a_1, \ldots, a_k, b \in \mathbb{Z}$ and $\sim \in \{\leq, \geq, <, >, =\}$ with constants represented in binary. It is known that satisfiability of existential Presburger formulas is NP-complete even with these extensions (cf. [26]).

3 Pushdown systems with counters and clocks

The model. An *atomic clock constraint* on clock variables $Y = \{y_1, \ldots, y_m\}$ is simply an expression of the form $y_i \sim c$ or $y_i - y_j \sim c$, where $\sim \in \{<, >, =\}$, $1 \leq i, j \leq m$ and $c \in \mathbb{Z}$. An *atomic counter constraint* on counter variables $X = \{x_1, \ldots, x_n\}$ is simply an expression of the form $x_i \sim c$, where $c \in \mathbb{Z}$. A *clock-counter (CC) constraint* θ on (X, Y) is simply a boolean combination of atomic counter constraints on X and atomic clock constraints on Y. Given a valuation $\nu : X \cup Y \rightarrow \mathbb{N}$ to the counter/clock variables, we can determine whether $\theta[\nu]$ is true or false by replacing a variable z by $\nu(z)$ and evaluting the resulting arithmetic expressions in the obvious way. Let $Const_{X,Y}$ denote the set of all CC constraints on (X, Y). Intuitively, these formulas will act as "enabling conditions" (or "guards") to determine whether certain transitions can be fired.

A *pushdown system with n counters and m discrete clocks* is a tuple $\mathcal{P} = (Q, \mathsf{ACT}, \Gamma, \delta, X, Y)$ where (1) Q is a finite set of states, (2) ACT is a set of action labels, (3) Γ is a stack alphabet, (4) $X = \{x_1, \ldots, x_n\}$ is a set of counter variables, (5) $Y = \{y_1, \ldots, y_m\}$ is a set of clock variables, and (6) δ is a transition

relation of $\mathcal{P}$, which is defined to be a finite subset of $(Q\times\Gamma^*\times Const_{X,Y})\times\mathsf{ACT}\times(Q\times\Gamma^*\times 2^Y\times\mathbb{Z}^n)$. A *configuration* of $\mathcal{P}$ is a tuple $(q,u,\mathbf{u},\mathbf{v})\in Q\times\Gamma^*\times\mathbb{N}^n\times\mathbb{N}^m$. Given $a\in\mathsf{ACT}$ and two configurations $\alpha_1=(q,u,\mathbf{u},\mathbf{v})$ and $\alpha_2=(q',u',\mathbf{u}',\mathbf{v}')$ of $\mathcal{P}$, $\alpha_1\rightarrow_{a,\mathcal{P}}\alpha_2$ iff there exists $\langle(q,w,\theta),a,(q',w',Y',\mathbf{w})\rangle\in\delta$ such that

- θ holds under the valuation $(\mathbf{u},\mathbf{v})$,
- for some $v\in\Gamma^*$, $u=wv$ and $u'=w'v$,
- if $Y'=\emptyset$, then each clock progresses by one time unit, i.e., $\mathbf{v}'=\mathbf{v}+\mathbf{1}$; if $Y'\neq\emptyset$, then the clocks in Y' are reset, while other clocks do not change, i.e., for each $y_i\in Y$, set $v_i':=0$ and, for each $y_i\notin Y$, set $v_i':=v_i$.
- $\mathbf{u}':=\mathbf{u}+\mathbf{w}$ (note, all elements of $\mathbf{u}'$ must be non-negative).

The transition system generated by $\mathcal{P}$ is $\mathfrak{S}_{\mathcal{P}}=\langle S,\{\rightarrow_a\}_{a\in\mathsf{ACT}}\rangle$, where S denotes the set of configurations of $\mathcal{P}$ and $\rightarrow_a$ is defined to be $\rightarrow_{a,\mathcal{P}}$.

Notice that in this model, which is similar to the model given in [9], clocks are updated via transitions. This is slightly different from the usual definition of discrete timed systems (cf. [1]) in which (1) clocks may progress within any particular state of the system without taking any transitions as long as some *invariants* are satisfied, and (2) transitions are *instantaneous*. However, by introducing self-looping transitions with no reset and adding an extra "dummy" clock which always resets for old transitions, we can easily construct a weakly bisimilar system in our model. See [9] for more details.

Let us now define the r-reversal-bounded variant of this model for each $r\in\mathbb{N}$. Syntactically, a *pushdown system with n r-reversal bounded counters and m discrete clocks* is simply a pair $(r,\mathcal{P})$ of number r and a pushdown system $\mathcal{P}$ with n counters and m clocks. Together with a given initial configuration α of $\mathcal{P}$, the system $(r,\mathcal{P})$ generates a transition system $\mathfrak{S}^{\alpha}_{(r,\mathcal{P})}=\langle S,\{\rightarrow_a\}_{a\in\mathsf{ACT}}\rangle$ defined as follows. Let $\mathfrak{S}_{\mathcal{P}}=\langle S',\{\rightarrow'_a\}_{a\in\mathsf{ACT}}\rangle$ be the transition system generated by $\mathcal{P}$. A path π in $\mathfrak{S}_{\mathcal{P}}$ from α is said to be *r-reversal-bounded* if each counter of $\mathcal{P}$ changes from a non-incrementing mode to non-decrementing mode (or vice versa) at most r times in the path π. For example, if the values of a counter x in a path π from α are $1,1,1,2,3,4,4,\overline{4,3},2,\overline{2,3}$, then the number of reversals of x is 2 (reversals occur in between the overlined positions). This sequence has three *phases* (i.e. subpaths interleaved by consecutive reversals or end points): non-decrementing, non-incrementing, and finally non-decrementing. This intuition suffices for understanding the main ideas in this paper, though we provide the detailed definitions in the full version. The set S is then defined to be the set of all finite r-reversal-bounded paths from α. Given two such paths π and π' such that $\pi'=\pi,\alpha'$, we define $\pi\rightarrow_a\pi'$ iff $\mathbf{last}(\pi)\rightarrow'_a\alpha'$. Notice that $\mathfrak{S}^{\alpha}_{(r,\mathcal{P})}$ is a tree.

We write (r,n,m)-PCC to denote the set of all pushdown systems with n r-reversal-bounded counters and m discrete clocks. We write PCC to denote the union of all (r,n,m)-PCC for all $r,n,m\in\mathbb{N}$. Similarly, we use (r,n)-PCo to denote $(r,n,0)$-PCC and (r,m)-PCl to denote $(r,0,m)$-PCC. We use PCo and PCl as well in the same way.

Unless stated otherwise, we make the following conventions: (1) the number r of reversals is given in unary, and (2) numeric constants in CC constraints

and counter increments are given in binary. In the sequel, we will deal with (control-state) reachability, model checking LTL and EF over PCC (and their variants) defined as follows:

- *Reachability*: given a PCC $\mathcal{P}$ and two configurations α, α' of $\mathcal{P}$ (in binary representation), decide whether α' is reachable in $\mathfrak{S}^{\alpha}_{\mathcal{P}}$.
- *Control-state reachability*: given a PCC $\mathcal{P}$ and two states q, q' of $\mathcal{P}$, decide whether there exist stack contents $u, u' \in \Gamma$ and counter values $\mathbf{c}, \mathbf{c}' \in \mathbb{N}^k$ such that $(q', u', \mathbf{c}')$ is reachable in $\mathfrak{S}^{(q,u,\mathbf{c})}_{\mathcal{P}}$.
- *LTL model checking*: given a PCC $\mathcal{P}$, a configuration α of $\mathcal{P}$ (in binary), and an LTL formula φ, decide whether $\mathfrak{S}^{\alpha}_{\mathcal{P}}, \alpha \models \varphi$.
- *EF model checking*: given a PCC $\mathcal{P}$, a configuration α of $\mathcal{P}$ (in binary), and an EF formula φ, decide whether $\mathfrak{S}^{\alpha}_{\mathcal{P}}, \alpha \models \varphi$.

4 Upper bounds for model checking PCo

In this section, we show that reachability over PCo is in NP by providing a direct poly-time reduction to satisfactions of existential Presburger formulas. As applications of our technique, we will provide: (1) an NP upper bound for control-state reachability with additional constraints on counter values at the beginning/end of the run and how many times actions are executed at the end of the run, and (2) a coNEXP upper bound for LTL model checking over PCo (coNP for fixed formulas). Lower bounds are proved in the next section.

An NP procedure for reachability over PCo. We prove the following.

Theorem 1. *Reachability over PCo is* NP*-complete. In fact, it is poly-time reducible to checking satisfactions of existential Presburger formulas.*

Given an (r,k)-PCo $\mathcal{P} = (Q, \mathsf{ACT}, \Gamma, \delta, X)$, and two configurations $\alpha = (q, u, \mathbf{c})$ and $\alpha' = (q', u', \mathbf{c}')$ of $\mathcal{P}$, our algorithm decides if α' is reachable from α in $\mathfrak{S}_{\mathcal{P}}$. Let $\mathbf{c} = (c_1, \ldots, c_k)$ and $\mathbf{c}' = (c'_1, \ldots, c'_k)$. We assume that $u = u' = \bot$ since, by hardwiring u and u' into the finite control of $\mathcal{P}$ the standard way, the PCo $\mathcal{P}$ may initialize the stack content to u and make sure the final stack content is precisely u'. In addition, let $d_1 < \ldots < d_m$ denote all the numeric constants appearing in an atomic counter constraint as a part of CC constraints in $\mathcal{P}$. Without loss of generality, we assume that $d_1 = 0$ for notational convenience. Let $\mathtt{REG} = \{\varphi_1, \ldots, \varphi_m, \psi_1, \ldots, \psi_m\}$ be a set of formulas defined as follows. Note that these formulas partition $\mathbb{N}$ into $2m$ pairwise disjoint regions.

$$\varphi_i(x) \equiv x = d_i, \quad \psi_i(x) \equiv d_i < x < d_{i+1}\ (1 \leq i < m), \quad \psi_m(x) \equiv d_m < x\ .$$

A vector $\mathbf{v}$ in $\mathbf{Modes} = \mathtt{REG}^k \times [0,r]^k \times \{\uparrow, \downarrow\}^k$ is said to be a *mode vector*. Given a path $\pi = \alpha_0 \alpha_1 \ldots \alpha_h$ from α to α', we may associate a mode vector $\mathbf{v}_j$ to each configuration α_i in π that records for each counter: which region its value is in, how many reversals its used, and whether its phase is non-decrementing ($\uparrow$) or non-incrementing ($\downarrow$). Consider the sequence $\sigma = \{\mathbf{v}_j\}_{j=0}^{h}$ of mode vectors. A

crucial observation is that each mode vector $\mathbf{v} \in \mathbf{Modes}$ in this sequence occurs in a contiguous block, i.e., if $0 \leq j \leq j' \leq h$ are such that $\mathbf{v}_j$ (resp. $\mathbf{v}_{j'}$) is the first (resp. last) time $\mathbf{v}$ appearing in σ, then $\mathbf{v}_l = \mathbf{v}$ for all $l \in [j, j']$. Intuitively, once a change occurs in σ, we cannot revert to the previous vector. This is because any such change will incur an extra reversal for at least one counter. There are at most $N_{\max} := |\mathtt{REG}| \times (r+1) \times k = 2mk(r+1)$ distinct mode vectors in σ.

In outline, to avoid an exponential blow-up in our reduction, we will first construct a very rough "upperapproximation" of the PCo $\mathcal{P}$ as a PDA $\mathcal{P}'$. Intuitively, $\mathcal{P}'$ will simulate $\mathcal{P}$, while guessing and remembering *only* how many mode vector changes have occurred, but disregarding the counter information. In this case, there are runs in $\mathcal{P}'$ that are not valid in $\mathcal{P}$. Each time $\mathcal{P}'$ fires a transition t (derived from a transition t' of $\mathcal{P}$ by disregarding counters), it will also output information about the counter tests and in(de)crements associated with t', and how many changes in the mode counter have occurred thus far (recorded in the states of $\mathcal{P}'$). Since $\mathcal{P}'$ is of polynomial size, we apply on $\mathcal{P}'$ the linear-time algorithm of Verma *et al.* [34] to compute the Parikh images of CFGs (equivalently, PDA). Building on the output formula, we use further existential quantifications to guess the evolution of the mode vectors, on which we impose further constraints to eliminate invalid runs. We give the details below.

Building the PDA $\mathcal{P}$. Define $\mathcal{P}' = (Q', \mathsf{ACT}', \Gamma, \delta', (q,0), F')$ allowing transitions to execute a (finite) *sequence* of actions, instead of just one. These are for convenience and can be encoded in the states of $\mathcal{P}'$. Let $Q' = Q \times [0, N_{\max}-1]$ and define ACT' implicitly from δ'. In fact, ACT' is a (finite) subset of $\{(\mathtt{ctr}_i, u, j, l) : i \in [1,k], u \in \mathbb{Z}, j \in [0, N_{\max}-1], l \in \{0,1\}\} \cup (Const_{X,\emptyset} \times [0, N_{\max}-1])$. Here, $l \in \{0,1\}$ signifies whether this action changes the mode vector. We define δ' by initially setting $\delta' = \emptyset$ and adding rules as follows. If $\langle (q,w,\theta), a, (q',w',\mathbf{u})\rangle \in \delta$ where $\mathbf{u} = (u_1, \ldots, u_k)$ then for each $i \in [0, N_{\max}-1]$ we add

$$\langle ((q,i),w), (\theta,i)(\mathtt{ctr}_1, u_1, i, 0)(\mathtt{ctr}_2, u_2, i, 0) \ldots (\mathtt{ctr}_k, u_k, i, 0), ((q,i),w')\rangle$$

to δ'. If $i \in [0, N_{\max}-1)$, we also add the following rule:

$$\langle ((q,i),w), (\theta,i)(\mathtt{ctr}_1, u_1, i, 1)(\mathtt{ctr}_2, u_2, i, 1) \ldots (\mathtt{ctr}_k, u_k, i, 1), ((q,i+1),w')\rangle$$

In this way, $\mathcal{P}'$ makes "visible" the counter tests and the counter updates performed. Finally, the set F' of final states are defined to be $\{q'\} \times [0, N_{\max}-1]$ and the initial control state is $(q,0)$.

Constructing the formula. Fix an ordering on ACT', say $\alpha_1 < \ldots < \alpha_l$. For convenience, by f we denote a function mapping α_i to i for each $i \in [1,l]$. We apply the linear-time algorithm of [34] on $\mathcal{P}'$ above to obtain $\chi(\mathbf{z})$, where $\mathbf{z} = (z_1, \ldots, z_l)$, such that for each $\mathbf{n} \in \mathbb{N}^l$ we have $\mathbf{n} \in \mathbb{P}(\mathcal{L}(\mathcal{P}'))$ iff $\chi(\mathbf{n})$ holds. We impose constraint χ to eliminate vectors that do not correspond to traces of $\mathcal{P}'$. Currently, $\mathcal{P}'$ knows only the maximum number of allowed changes in mode vectors, but is not "aware" of other information about the counters. Therefore, the formula that we construct should assert the existence of a valid sequence of mode vectors that respect the counter tests and updates that $\mathcal{P}'$ outputs. We

construct $\texttt{HasRun}$ of the form

$$\exists \mathbf{z} \exists \mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}-1} \begin{pmatrix} \texttt{Init}(\mathbf{m}_0) \wedge \texttt{GoodSeq}(\mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}-1}) \wedge \chi(\mathbf{z}) \wedge \\ \texttt{Respect}(\mathbf{z}, \mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}-1}) \wedge \texttt{EndVal}(\mathbf{z}) \end{pmatrix}$$

where $\mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}-1}$ are variables for representing a valid sequence of mode vectors occurring in the run of $\mathcal{P}$.

Let us now elaborate $\texttt{HasRun}$. First, since a mode vector is a member of $\mathbf{Modes} = \texttt{REG}^k \times [0,r]^k \times \{\downarrow, \uparrow\}^k$, we set $\mathbf{m}_i$ to be a tuple of variables $\{reg^i_j, rev^i_j, arr^i_j : j \in [1,k]\}$, where:

- reg^i_j is a variable that will range over $[1, 2m]$ denoting which region the jth counter is in (a number of the form $2i+1$ refers to φ_i, while $2i$ refers to ψ_i).
- rev^i_j is a variable that will range over $[0, r]$ denoting the number of reversals that have been used thus far by the jth counter.
- arr^i_j is a variable that will range over $\{0, 1\}$ denoting the current arrow direction, e.g., 0/1 for $\uparrow/\downarrow$ (non-decrementing/non-incrementing mode).

Using $\texttt{Init}(\mathbf{m}_0)$, we assert that the initial mode vector needs to respect the given initial configuration $\alpha = (q, u, \mathbf{c})$, where $\mathbf{c} = (c_1, \ldots, c_k)$. More precisely,

$$\texttt{Init}(\mathbf{m}_0) \equiv \bigwedge_{j=1}^{k} \left(rev^0_j = 0 \wedge \bigwedge_{i=1}^{m} \begin{pmatrix} reg^0_j = 2i-1 \leftrightarrow \varphi_i(c_j) \bigwedge \\ reg^0_j = 2i \leftrightarrow \psi_i(c_j) \end{pmatrix} \right).$$

In fact, since $\mathbf{c}$ is given, we could replace some of these variables by constants. However, being able to define this in the formula allows us to prove something more general, as we will see later.

Recall that the target configuration is $\alpha' = (q', u', \mathbf{c}')$, where $\mathbf{c}' = (c'_1, \ldots, c'_k)$. We assert that the end counter values match $\mathbf{c}'$. This definition is given as follows. Note, multiplications by constants are allowed within Presburger arithmetic.

$$\texttt{EndVal}(\mathbf{z}) \equiv \bigwedge_{j=1}^{k} \left(\sum_{i=0}^{r} \sum_{(\texttt{ctr}_j, d, i, l) \in \mathsf{ACT}'} d \times z_{f(\texttt{ctr}_j, d, i, l)} \right) = c'_j.$$

We define $\texttt{GoodSeq}(\mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}-1})$ to express that $\mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}-1}$ is a valid sequence of mode counters. The formula is a conjunction of smaller formulas defined below. One conjunct says that each rev^i_j must be a number in $[0, r]$. Likewise, we add conjuncts expressing that each reg^i_j (resp. arr^i_j) ranges over $[1, 2m]$ (resp. $\{0, 1\}$). We also need to state that changes in the direction arrows incur an extra reversal (otherwise, no reversal is incurred):

$$\bigwedge_{j=1}^{k} \bigwedge_{i=0}^{N_{\max}-1} 0 \leq rev^i_j \leq r. \wedge \bigwedge_{j=1}^{k} \bigwedge_{i=0}^{N_{\max}-2} \begin{pmatrix} arr^i_j \neq arr^{i+1}_j \rightarrow rev^{i+1}_j = rev^i_j + 1 \bigwedge \\ arr^i_j = arr^{i+1}_j \rightarrow rev^{i+1}_j = rev^i_j \end{pmatrix}.$$

Finally, the sequence $\{reg^i_j\}_{i=0}^{N_{\max}-1}$ must obey the changes in $\{arr^i_j\}_{i=0}^{N_{\max}-1}$:

$$\bigwedge_{j=1}^{k} \bigwedge_{i=0}^{N_{\max}-2} \left(reg^i_j < reg^{i+1}_j \rightarrow arr^{i+1}_j = 0 \bigwedge reg^i_j > reg^{i+1}_j \rightarrow arr^{i+1}_j = 1 \right).$$

I.e., since regions denote increasing intervals, going to higher (resp. lower) regions mean the counter mode must be non-decrementing (resp. non-incrementing).

Lastly, we give $\texttt{Respect}(\mathbf{z}, \mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}-1})$. Again, this is a conjunction. First, when the jth counter is non-incrementing (resp. non-decrementing), we allow only non-negative (resp. non-positive) counter increments:

$$\bigwedge_{j=1}^{k} \bigwedge_{i=1}^{N_{\max}-1} \begin{pmatrix} arr_j^i = 0 \to \bigwedge_{(\texttt{ctr}_k,d,i,l)\in \mathsf{ACT}', d<0} z_{f(\texttt{ctr}_k,d,i,l)} = 0 \bigwedge \\ arr_j^i = 1 \to \bigwedge_{(\texttt{ctr}_k,d,i)\in \mathsf{ACT}', d>0} z_{f(\texttt{ctr}_k,d,i,l)} = 0 \end{pmatrix}.$$

Secondly, the value of the jth counter at the beginning and end of each mode must respect the guessed mode vector. Let us first introduce some notations. Observe that the value of the jth counter at the *end* of ith mode vector can be expressed by $c_j + \left(\sum_{i'=0}^{i-1} \sum_{(\texttt{ctr}_j,d,i',l)\in \mathsf{ACT}'} d \times z_{f(\texttt{ctr}_j,d,i',l)}\right) + \sum_{(\texttt{ctr}_j,d,i,0)} d \times z_{f(\texttt{ctr}_j,d,i,0)}$. Let us denote this term by $\mathit{EndCounter}_j^i$. Similarly, the value at the *beginning* of the ith mode is $c_j + \sum_{i'=0}^{i-1} \sum_{(\texttt{ctr}_j,d,i',l)\in \mathsf{ACT}'} d \times z_{f(\texttt{ctr}_j,d,i',l)}$. We denote this term by $\mathit{StartCounter}_j^i$. Hence, this second conjunct is

$$\bigwedge_{j=1}^{k} \bigwedge_{i=0}^{N_{\max}-1} \bigwedge_{l=1}^{m} \begin{pmatrix} reg_j^i = 2l-1 \to (\varphi_l(\mathit{EndCounter}_j^i) \wedge \varphi_l(\mathit{StartCounter}_j^i)) \bigwedge \\ reg_j^i = 2l \to (\psi_l(\mathit{EndCounter}_j^i) \wedge \psi_l(\mathit{StartCounter}_j^i)) \end{pmatrix}.$$

Finally, we need to express that no invalid counter tests are executed in a given mode. To test whether a CC constraint θ is satisfied by the values $\mathbf{b} = (b_1, \ldots, b_k)$ of the counters, it is necessary and sufficient to test whether θ is satisfied by *some* vector $\mathbf{b}' = (b'_1, \ldots, b'_k)$, where each b_i lies in the same region in REG as b'_i. Therefore, the desired property can be expressed as:

$$\bigwedge_{i=0}^{N_{\max}-1} \bigwedge_{(\theta,i)\in \mathsf{ACT}'} z_{f(\theta,i)} > 0 \to \theta(\mathit{StartCounter}_1^i, \ldots, \mathit{StartCounter}_k^i).$$

Of course, we might want to make the formula smaller by associating new variables for all terms $\mathit{StartCounter}_j^i$ and $\mathit{EndCounter}_j^i$. Since the translation from PDA to CFGs produces an output of cubic size, it is easy to check that the size of $\texttt{HasRun}$ is cubic in $\|\mathcal{P}\|$, r, and k.

Applications. We start with a straightforward application of the above proof: Control-state reachability over PCo with additional existential Presburger constraints on counter values at the beginning/end of the run and on how many times actions are executed at the end of the run can be checked in NP. In fact, it is poly-time reducible to checking satisfactions over existential Presburger formulas. To see this, observe that the counter values in α and α', which were treated as constants in $\texttt{HasRun}$, could be treated as *variables*. Hence, we can add the additional constraint within the inner bracket of $\texttt{HasRun}$ as a conjunct, and quantify the new variables for counter values. Secondly, we have the following:

Theorem 2. *LTL model checking over PCo is* coNEXP*-complete. For fixed formulas, it is* coNP*-complete.*

For this, we begin with the standard Vardi-Wolper automata-theoretic approach [32] reducing the *complement* of this problem to *recurrent reachability* over PCo: given a PCo $\mathcal{P} = (Q, \mathsf{ACT}, \Gamma, \delta, X)$ and a subset $F \subseteq Q$, decide if there is a run of $\mathcal{P}$ visiting configurations of the form $(q, u, \mathbf{c})$ where $q \in F$ infinitely often. This reduction obtains an NBWA of exponential size from the negation of the given LTL formula, and builds the product of this NBWA and $\mathcal{P}$. Thus, the reduction is exponential in the LTL formula but polynomial in the PCo. Therefore, it suffices to show that recurrent reachability is in NP. Observe that all infinite runs π of $\mathcal{P}$ stabilise in some mode, expressed by a mode vector $\mathbf{m}$. We split π into a subpath from the initial configuration α to a configuration $\alpha' = (q', u', \mathbf{c}')$ in π in mode $\mathbf{m}$, and the rest of the path from α'. In fact, if $\mathbf{c}' = (c'_1, \ldots, c'_k)$, then α' could be chosen such that if the value of the jth counter after α' in π has a maximum c (e.g. when jth counter value stabilises in a region $< 2m$, which is bounded), then $c'_j = c$. By allowing only rules which do not decrement counters whose values do not stabilise, we may treat the path from α' as a path of a PDS with no counters. Hence, we use a well-known lemma of PDS (w.l.o.g. assume that transitions of PDS/PCo only check the top stack character):

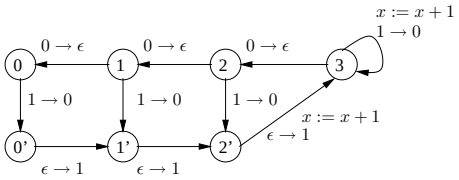
Lemma 3 ([4]). *Given a PDS $\mathcal{P} = (Q, \mathsf{ACT}, \Gamma, \delta)$, an initial configuration α, and a set $F \subseteq Q$, then there exists an infinite computation path of $\mathcal{P}$ from α visiting F infinitely often iff there exist configurations $(p, \alpha) \in Q \times \Gamma$, $(g, u), (p, \alpha v)$ such that $g \in F$ and (i) α can reach $(p, \alpha w)$ for some $w \in \Gamma^*$, and (ii) (p, α) can reach (g, u), from which $(p, \gamma v)$ is reachable.*

We construct a PCo $\mathcal{P}'$ that simulates $\mathcal{P}$ (for the initial subpath of the recurrent reachability witness). Eventually, it decides to stop simulating $\mathcal{P}$ at some configuration $(p, \alpha w, \mathbf{v})$. The state and the top-stack character are then made visible by $\mathcal{P}'$ with an action $\overline{(p, \alpha)}$. The PCo $\mathcal{P}'$ then empties the stack and continues the simulation of $\mathcal{P}$ from (p, α) except that (1) only rules which increment counters by non-negative values are allowed, but instead of actually modifying the counter values, actions of the form $+\mathtt{ctr}_j$ will be executed if the rule increments the jth counter by a positive value (2) counter tests θ are no longer performed, but instead we execute actions that signify θ. When a configuration of the form $(g, u, \mathbf{v})$ is reached with $g \in F$, it will make it visible by performing an action of the form $g!$. At any given time after this, it may output a character $\overline{(p', \alpha')}$, where p' is the current state and α' is the current top stack character, and go into the state `Finish`. There exists a computation path of $\mathcal{P}$ from α that visits F infinitely often iff there exists a path from α to the `Finish` in $\mathcal{P}'$ such that: (a) the number executions of some $\overline{(p, \alpha)}$ is 2, (b) some $g!$ action has been executed, (c) if the region of the end value of the jth counter (corresponding to the jth counter value of the initial witnessing subpath of $\mathcal{P}$) is bounded (i.e. $< 2m$), then no action $+\mathtt{ctr}_j$ must have been executed, and (d) no counter test actions violating the regions of the end counter values have been executed. Using the techniques from Theorem 1, we may express these constraints as a poly-size existential Presburger formula. In conclusion, we have reduced recurrent reachability over PCo to control-state reachability over PCo with constraints, which we already saw to be in NP.

5 Lower bounds for model checking PCo

In this section, we prove coNEXP and coNP lower bounds for model checking LTL and reachability over PCo. We will also show that model checking EF-logic is undecidable even for a fixed formula. All our lower bounds hold for $(1,1)$-PCo where counters can only be compared against 0 and incremented by $\{-1,0,1\}$.

Lower bound for LTL and reachability We start with coNP-hardness for a fixed LTL formula over $(1,1)$-PCo. We show that the complement of this model checking problem is NP-hard. It will be clear later that the same proof can be used to show that reachability is NP-hard over $(1,1)$-PCo. The idea is to reduce from the NP-complete `0-1 Knapsack` problem ([23]): given $a_1,\ldots,a_k,b \in \mathbb{N}$ in binary, decide whether $\sum_{i=1}^{k} a_i x_i = b$ for some $x_1,\ldots,x_k \in \{0,1\}$. The reduction constructs a $(1,1)$-PCo $\mathcal{P}$ which initializes the counter to 0 and repeats for each $i \in [1,k]$: guess x_i, add $a_i x_i$ to the counter. The PCo then checks whether the counter is b by substracting b and checking whether the result is 0. If the test is positive, execute a special action *success* and then loop silently. The LTL formula is $\mathsf{G}\neg$*success*. Note, this PCo makes one reversal. The problem with this reduction, however, is that the numbers $a_1,\ldots,a_k,b$ are given in binary, whereas each transition in $\mathcal{P}$ can add -1 or 1. Hence, we cannot naively hardwire the "intermediate" values of $a_1,\ldots,a_k,b$ during the computation in the finite control. Instead, we need to use the stack. We illustrate this technique by the example in the following figure on the right. This is a PCo with one counter x that increases x by the number represented by the topmost four bits on the stack (edge label $u \to v$ defines the stack operation). For example, if the PCo starts with the configuration $(3, 1101, 0)$, then it will end at the configuration $(0, 0, 13)$. Using this technique, we will only need at most $\sum_{i=1}^{k} \log(a_i) + \log(b)$ extra states and therefore avoiding an exponential blow-up.



For the coNEXP lower bound for non-fixed formulas, we reduce succinct `0-1 Knapsack`. We define `Succinct 0-1 Knapsack` as the `0-1 Knapsack` problem where the input is given as a boolean formula θ with variables $x_1,\ldots,x_{k+m}$ where $k,m \in \mathbb{Z}_{>0}$ are given in unary. Here θ represent the numbers $a_1,\ldots,a_{2^k-1},b \in \mathbb{N}$ each with precisely 2^m bits (leading 0s permitted) as follows:

- The ith bit of $\mathbf{bin}(b)$ is defined to be $x \in \{0,1\}$ iff the formula θ evaluates to x when $x_1,\ldots,x_{k+m}$ are evaluated to $0^k\mathbf{bin}^m(i)$.
- The ith bit of $\mathbf{bin}(a_j)$ is defined to be $x \in \{0,1\}$ iff the formula θ evaluates to x when the inputs to $x_1,\ldots,x_{k+m}$ are $\mathbf{bin}^k(j)\mathbf{bin}^m(i)$.

The problem is to check whether $\sum_{i=1}^{2^k-1} a_i z_i = b$ for some $z_1,\ldots,z_{2^k-1} \in \{0,1\}$. The problem of `Succinct 0-1 Knapsack` can be shown to be NEXP-complete in the same way that the problem `Succinct Knapsack`, where natural numbers can be assigned to z_i's (instead of only $\{0,1\}$), is shown in [33] to be NEXP-complete.

We now show a NEXP lower bound for the complement of LTL model checking over $(1,1)$-PCo by reducing from `Succinct 0-1 Knapsack`. The idea of the reduction is the same as for the case of fixed formulas, but we will have to use the LTL formula to count up to doubly exponential values.

The stack alphabet includes $\#, c_1^0, c_1^1, \ldots, c_m^0, c_m^1$. If the ith bit of a_j is 1 and z_j has been guessed to be 1, then we need to add 2^i to the counter. E.g., suppose we're on the $11\ldots1$th bit of the a_0. Using the techniques below, we calculate the value of this bit. If it is 1, we increment $2^{(2^m-1)}$ times. To do this, we push the following sequence on the stack, again using techniques described below.

$$0c_m^0 \ldots c_1^0 \# 0 c_m^1 c_{m-1}^0 \ldots c_1^0 \# \ldots \ \ldots \ \ldots \# 0 c_m^1 \ldots c_1^1 \# \qquad (*)$$

That is, a bit-string with 2^m many 0s, annotated with their bit positions (least significant bit on the top/left of the stack). From this stack configuration, the system counts up to $1c_m^0 \ldots c_1^0 \# 1 c_m^1 c_{m-1}^0 \ldots c_1^0 \# \ldots \ \ldots \ \ldots \# 1 c_m^1 \ldots c_1^1 \#$, i.e., a bit-string with 2^m many 1s, taking $2^{(2^m-1)}$ steps. We increment the counter $+1$ at each step.

To complete the proof, we need to know how to: (1) enumerate all assignments to $x_1, \ldots, x_{k+m}$ and evaluate θ, (2) initialise the large counter described above, and (3) increment the large counter from $00\ldots0$ to $11\ldots1$.
Problem 1: We store the current assignment on the bottom of the stack, using $x_1^0, x_1^1, \ldots, x_{k+m}^0, x_{k+m}^1$. Initially, we push $x_1^0, \ldots, x_{k+m}^0$, making the pushed characters visible using suitable action symbols. We then guess whether θ evaluates to 1. An LTL formula encoding θ asserts this guess is correct. To move to the lexicographically next assignment, we erase the stack, making the characters visible, and then guess the next assignment, using the LTL to check it.
Problem 2: Guess and push $0c_1^{y_1} \ldots c_m^{y_m} \#$, using the formula to check this matches the $x_{k+1}^{y_1}, \ldots, x_{k+m}^{y_m}$ on the stack. We then push $0c_1^{Y_1} \ldots c_m^{Y_m} \#$ (with the values of $Y_1, \ldots, Y_m \in \{0,1\}$ guessed) to the stack. This is done arbitrarily many times and the PCo may stop at some point. To make sure we obtain (*) on the top of the stack, we assert the successor property using the LTL formula.
Problem 3: This uses the idea for the fixed formula case: pop from the stack until the first 0, then push a correct number of 1s annotated with the bit positions. For this, an LTL formula asserts the successor property, cf. [28].

Undecidability results for EF-logic We now turn our attention to model checking EF-logic over PCo. It turns out that the problem is already undecidable for a fixed formula with two operators. We reduce from the emptiness of linear bounded Turing machines (LTM), which is undecidable (cf. [27]). Given an LTM $\mathcal{M}$ that accept/rejects an input w using at most $c|w|$ space, we compute a $(1,1)$-PCo $\mathcal{P}$, an EF formula φ, and a start state q_0 of $\mathcal{P}$ such that $\mathfrak{S}_{\mathcal{P}}, (q_0, \epsilon) \models \varphi$ iff $\mathcal{M}$ is nonempty. We make $\mathcal{P}$ guess a word w and an accepting computation path of $\mathcal{M}$, which is stored in the stack. The length $c|w|$ is stored in the counter. Once the path has been guessed, it suffices to show that: (P1) the length of *each* guessed configuration is $c|w|$, and (P2) *each* non-initial configuration is a successor of its previous configuration. The guessing and checking stages incur one alternation for the EF formula. Since checking P2 requires us to check at

most four consecutive tape cells of $\mathcal{M}$ (once a cell is chosen), we can remember this location by decrementing the counter, moving to the previous configuration of $\mathcal{M}$ that is stored in the stack, and then further decrementing the counter making sure that the end value is 0. See the full version for the proof.

Theorem 4. *Model checking EF-logic over* $(1,1)$*-PCo is undecidable even for a fixed formula with two* EF *operators.*

6 Adding clocks

To extend our results to the case of PCC, we use the region construction of Alur and Dill [1] to reduce a PCC to a PCo of exponential size (in exponential time) such that every run of the PCC can be projected, state-for-state, onto a run of the PCo. From S4, we obtain a coNEXP (resp. NEXP) upper bound for LTL model checking (resp. reachability) over PCC. We next provide a lower bound.

Theorem 5. *Reachability is* NEXP*-hard for PDS with discrete clocks and one 1-reversal-bounded counter. When clock constraint constants are given in binary, only three clocks and one single-reversal counter are needed. Similarly, LTL model-checking is* coNEXP*-hard, even for a fixed formula.*

We first consider unary constants, and a non-fixed number of clocks. We adapt the previous reduction from `Succinct 0-1 Knapsack`, except the formula can no longer be used to evaluate the boolean formulas. Instead, we encode bits with two clocks, which are equal iff the bit is 1. We evaluate boolean formulas using the transition guards. To test all assignments to $x_1, \ldots, x_{k+m}$ we store the valuation in the counters (not the stack). This is straightforward. Then, to build the large counter on the stack, we use another set of clocks to store the bit position values of the last two blocks pushed on to the stack. These clocks can be used to ensure the successor relation between the two values. For binary clock constraints, we use Courcoubetis and Yannakakis [8] to reduce to three clocks.

Recall that model checking EF-logic over PCo is undecidable. It turns out that this problem is decidable over PCl. See the full version for a proof.

Theorem 6. *Model checking EF over PCl is* EXPSPACE*-complete. The lower bound holds for fixed formulas with two clocks with binary constraint constants, or with a non-fixed number of clocks when the constraints are in unary.*

7 Experimental Results

We provide a prototypical C++ implementation of an optimised version of the reduction in Section 4. In particular, from the Verma *et al.*, we can derive, at no cost, the number of times each rule of the PCo is fired. From this, we infer the number of action symbols output, and hence, do not need additional variables and transitions for these. In addition, the per mode information per counter uses a single variable. Finally, we look for pairs of pushdown rules

$\langle (q_1, a_1, \top), \epsilon, (q_2, a_2, \emptyset, \emptyset^n) \rangle$ and $\langle (q_2, a_2, \top), \epsilon, (q_3, a_3, \emptyset, \emptyset^n) \rangle$ such that a_1 and a_2 are single characters and the pair q_2, a_2 does not appear together in any other rule. Furthermore, a_2 does not appear in a rule $\langle (q, w, \theta), a, (q', w' a_2 w'', Y, \mathbf{w}) \rangle$ where $|w'| > 0$. These rules can be replaced by $\langle (q_1, a_1, \top), \epsilon, (q_3, a_3, \emptyset, \emptyset^n) \rangle$. Then we remove unreachable productions from the generated CFGs. We used the 64-bit Linux binary of Z3 2.16 [10] on the Presburger formulas on a quad-core, 2.4Gz Intel® Xeon® machine with 12GB of RAM, running Fedora™12. We report the time from the Linux command `time` for the translation and the run of Z3.

Double free in dm-target.c. In version 2.5.71 of the Linux kernel, a double free was introduced to `drivers/md/dm-target.c` [30]. This was introduced to fix a perceived memory-leak. When registering a new target, memory was allocated and a check made to see if the target was already known. If so, it was freed and an "exists" flag set. Otherwise the target was added to the target list. Before returning, the exists flag was checked and the object was freed (again) if it was set. We created, by hand, a model of this file using a counter to store the length of the target list. The complete control flow of the file was maintained and data only tracked when relevant to the memory management. The model outputs special symbols to mark when memory is allocated or freed. We then look for a run where, either an item was removed from the empty list, the number of free calls was greater than the number of allocations, or, the code exited normally, but more memory was allocated than freed. We were able to verify that the code contained a memory error in version 2.5.71 and that the memory management was correct in earlier versions (for a bounded number of reversals) provided as many targets were registered as unregistered. Note, the counter is required to track the size of the list which ensures that the number of allocations matches the number of frees. The size of the `dm-target.c` is approximately 175 lines without comments. Detecting the bug took 2s, and proving correctness took 1.7s, 2.9s, 16s, 24s and 77s for 1, 2, 3, 4 and 5 reversals respectively.

Memory leak in aer_inject.c. In version 2.6.32 of the Linux kernel, the file `drivers/pci/pcie/aer/aer_inject.c` contains a memory leak that was patched in the next version [25]: two lists of allocated objects are maintained, but, when exiting, the code empties the items from the first list and frees them, then, empties and frees the first list again, instead of the second. We created a model of this driver with two counters to track the size of the lists and searched for memory errors as in the previous example. Only one reversal was required to detect the memory leak. We showed that the patch corrects the problem (up to one reversal). Note, without counters, it would always be possible for the number of allocations to differ from the number of frees. The file `aer_inject.c` is approximately 470 lines without comments. Detecting the bug required 220s and proving correctness for a single reversal took 508s.

Buffer overflow. Jhala and McMillan have a buffer overflow example which their technique, SatAbs and Magic, failed to verify [21]. There are three buffers, x, y and z of sizes $100, 100$ and 200 and two counters i and j. First, i is used to copy up to 100 positions of x into z. Then, counters i and j are used to copy up to 100 positions of y into the remainder of z. There is overflow if i, which indexes

z, finishes greater than 199. This simply encodes into our model[1] and could be verified correct (since the example is trivially reversal bounded) in 1.6s.

David Gries's coffee can problem [15]. We have an arbitrary number of black and white coffee beans in a can. We pick two beans at random. If they are the same colour, they are discarded and an extra black bean is put in the tin. If they differ, the white is kept but the black discarded. The last bean in the can is black iff the number of white beans is odd. This problem can be modelled without abstraction by using the stack to count the number of black beans, and a single-reversal counter to track the number of white beans. We verified in 1.3s that the last bean cannot be white if the number of white beans is odd, and in 1.1s that the last bean may be white if the number of white beans is even.

8 Extensions and Future Work

We prove, in the full version of this paper, that (i) reachability and LTL for a prefix-recognisable version of PCo is NEXP-complete and coNEXP-complete respectively, even with one 1-reversal-bounded counter and a fixed formula; (ii) LTL model checking for a fixed formula is coNEXP when the number of reversals is given in binary, whereas reachability is in NEXP (a matching lower bound is in [18], even without the stack); and (iii) the reachability problem for second-order pushdown systems (cf. [17]) with reversal bounded counters is undecidable.

For future work we may investigate counter-example guided abstraction refinement. We would need counter-examples from the models of the existential Presburger formula, and refinement techniques to add counters and reversals as well as predicates. Furthermore, as we allow user defined numerical constraints on reachability, we may also restrict LTL model checking to runs satisfying additional numerical fairness constraints.

Acknowledgments. We thank the authors of [7] and [33] for their correspondence, V. D'Silva for recommending [21], J. Leroux and P. Ruemmer for suggesting Z3, A. Sangnier for [13], and anonymous referees for their comments. We thank EPSRC (EP/F036361 and EP/H026878/1) for their support.

References

1. R. Alur and D. L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
2. T. Ball, B. Cook, V. Levin and S. K. Rajamani. SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft. In *IFM'04*, pages 1–20.
3. T. Ball and S. K. Rajamani. Bebop: A Symbolic Model Checker for Boolean Programs. In *SPIN'00*, pages 113–130.
4. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR'97*, pages 135–150.
5. P. Bouyer. Model-checking timed temporal logics. *Electr. Notes Theor. Comput. Sci.*, 231:323–341, 2009.
6. T. Cachat. Uniform Solution of Parity Games on Prefix-Recognizable Graphs. *Electr. Notes Theor. Comput. Sci.*, 68(6), 2002.

[1] Comparison with constants is not implemented. We adjusted the formula by hand.

7. R. Chadha, A. Legay, P. Prabhakar and M. Viswanathan. Complexity Bounds for the Verification of Real-Time Software. In VMCAI'10, pages 95–111.
8. C. Courcoubetis and M. Yannakakis. Minimum and Maximum Delay Problems in Real-Time Systems. *FMSD*, 1(4):385–415, 1992.
9. Z. Dang *et al.* Binary Reachability Analysis of Discrete Pushdown Timed Automata. In CAV'00, pages 69–84.
10. L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, 2008.
11. V. D'Silva, D. Kroening and G. Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(7):1165–1178, 2008.
12. J. Esparza, A. Kucera and S. Schwoon Model checking LTL with regular valuations for pushdown systems. *Inf. Comput.*, 186(2):355–376, 2003.
13. E. Filiot, J.-F. Raskin, P.-A. Reynier, F. Servais, and J.-M. Talbot. Properties of visibly pushdown transducers. In *MFCS*, 2010.
14. A. Finkel and A. Sangnier. Reversal-bounded counter machines revisited. In *MFCS*, 2008.
15. D. Gries. *The Science of Programming*. Springer-Verlag, 1981.
16. E. M. Gurari and O. H. Ibarra. The Complexity of Decision Problems for Finite-Turn Multicounter Machines. *J. Comput. Syst. Sci.*, 22(2):220–229, 1981.
17. M. Hague. *Saturation Methods for Global Model-Checking Pushdown Systems*. PhD thesis, Oxford University Computing Laboratory, 2009.
18. R. Howell and L. Rosier. An Analysis of the Nonemptiness Problem for Classes of Reversal-Bounded Multicounter Machines. *J. Comput. Syst. Sci.*, 34(1):55–74.
19. O. H. Ibarra. Reversal-Bounded Multicounter Machines and Their Decision Problems. *J. ACM*, 25(1):116–133, 1978.
20. O. Ibarra, J. Su, Z. Dang, T. Bultan, R. Kemmerer. Counter machines and verification problems. *Theor. Comput. Sci.* 289 (2002), 165–189.
21. R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In *TACAS*, 2006.
22. F. Laroussinie, A. Meyer, E. Petonnet. Counting CTL. In *FoSSaCS'10*, p. 206–220.
23. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.
24. R. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.
25. A. Patterson. PCI: fix memory leak in aer_inject. https://patchwork.kernel.org/patch/53058/, 2003.
26. B. Scarpellini. Complexity of Subcases of Presburger Arithmetic. *Trans. of AMS*, 284(1):203–218, 1984.
27. M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Co., 1997.
28. A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM* 32(3):733–749 (1985)
29. D. Suwimonteerabuth, S. Schwoon and J. Esparza. jMoped: A Java Bytecode Checker Based on Moped. In *TACAS'05*, pages 541–545.
30. J. Thornbur. dm: Fix memory leak in dm_register_target(). http://lkml.org/lkml/2003/6/9/70, 2003.
31. A. W. To. *Model Checking Infinite-State Systems: Generic and Specific Approaches*. PhD thesis, School of Informatics, University of Edinburgh, 2010.
32. M. Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *LICS'86*, pages 332–344.
33. H. Veith. Languages Represented by Boolean Formulas. *Inf. Process. Lett.*, 63(5):251–256, 1997.
34. K. N. Verma, H. Seidl and T. Schwentick. On the Complexity of Equational Horn Clauses. In CADE'05, pages 337–352.
35. I. Walukiewicz. Model Checking CTL Properties of Pushdown Systems. In *FSTTCS'00*, pages 127–138.