Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Theoretical Computer Science 299 (2003) 687–706

Theoretical
Computer Science

www.elsevier.com/locate/tcs

Eliminating the storage tape in reachability constructions☆

Oscar H. Ibarra[a,*,1], Zhe Dang[b]

[a] *Department of Computer Science, University of California, Santa Barbara, CA 93106, USA*
[b] *School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164, USA*

Received 3 October 2001; accepted 7 June 2002
Communicated by D.-Z. Du

Abstract

A discrete pushdown timed automaton is a pushdown machine with integer-valued clocks. It has been shown recently that the binary reachability of a discrete pushdown timed automaton can be accepted by a two-tape pushdown acceptor with reversal-bounded counters. We improve this result by showing that the stack can be eliminated from the acceptor, i.e., the binary reachability can be accepted by a two-tape finite-state acceptor with reversal-bounded counters. We calso obtain similar results for other machine models. Our results can be used to verify certain properties concerning these machines that were not verifiable before using previous techniques. For example, we are able to formulate a subset of Presburger LTL that is decidable for satisfiability checking with respect to these machines. We also discuss the "boundedness problem" for reachability sets. Finally, we explain how the storage tape elimination technique can be applied to machines with real-valued clocks.

Keywords: Pushdown timed automaton; Reachability; Reversal-bounded counters; Presburger arithmetic

1. Introduction

Developing verification techniques for infinite-state systems is an important ongoing effort, motivated to a large extent by the successes of model-checking techniques for

☆ A short version of this paper appeared in the *Proceedings* of the International Symposium on Algorithms and Computation (ISAAC 2001), Lecture Notes in Computer Science 2223 (Springer, Berlin, 2001) 244–256.

* Corresponding author.
E-mail address: ibarra@cs.ucsb.edu (O.H. Ibarra).

[1] His work was supported in part by NSF grants IRI-9700370 and IIS-0101134.

PII: S0304-3975(02)00545-5

finite-state systems [22]. Unlike for finite-state systems, there is a decidability boundary for infinite-state systems: machines with two counters (i.e., "Minsky machines") are Turing complete. Therefore, we must seek a balance between the computing power of infinite-state systems and their decidability.

Many infinite-state models have been shown decidable for various model-checking problems. These models include timed automata [2], pushdown automata and pushdown processes [3,13,11], various versions of counter machines [6,8,10,12,21], and various queue machines [1,4,17,19,23].

Pushdown systems are of particular interest, since, in practice, they are related to programs with procedure calls and, in theory, they are well studied in automata theory. A pushdown machine can be obtained by augmenting a finite-state machine with a pushdown stack. A configuration of a pushdown machine without an input tape (PM), is a string $\alpha = wq$, where w is the stack content and q is the state (we assume that the stack alphabet is disjoint from the state set). If M is a PM and S is a set of configurations, define the backward and forward reachability sets of M with respect to S by: $pre^*(M,S) = \{\alpha \mid \text{configuration } \alpha \text{ can reach some configuration in } S\}$ and $post^*(M,S) = \{\alpha \mid \text{configuration } \alpha \text{ is reachable from some configuration in } S\}$. It is known that if S is regular, then $pre^*(M,S)$ and $post^*(M,S)$ are also regular (see, e.g., [3,11,13]). One can also show that the binary reachability of M, $Binary(M) = \{(\alpha,\beta) \mid \beta \text{ is reachable from } \alpha\}$, can be accepted by a two-tape FA, i.e., a finite-state acceptor with two one-way input tapes. (Note that a one-tape FA is the usual finite automaton.)

A PM augmented with finitely many real-valued clocks is called a pushdown timed automaton, which is a generalization of a timed automaton [2]. It is discrete if the clocks can only assume nonnegative integer values (definitions are in Section 4). A characterization of the binary reachability of discrete pushdown timed automata has recently been given in [9]. It was shown in [9] (see also [20]) that the binary reachability of a discrete pushdown timed automaton can be accepted by a two-tape pushdown acceptor augmented with reversal-bounded counters. A counter (which, we assume w.l.o.g., can only store nonnegative integers, since the sign can be remembered in the states) is reversal-bounded if it can be tested for zero and can be incremented or decremented by one, but the number of alternations between nondecreasing mode and nonincreasing mode in any computation is bounded by a given constant; e.g., a counter whose values change according to the pattern 0 1 1 2 3 4 $\underline{4\ 3}$ 2 1 $\underline{0\ 1}$ $\underline{1\ 0}$ is three-reversal, where the reversals are underlined. It follows that the backward and forward reachability sets can be accepted by (one-tape) pushdown acceptors with reversal-bounded counters. These results and the fact that the emptiness problem for multitape pushdown acceptors with reversal-bounded counters is decidable [16,17] have been used recently to prove the decidability of certain verification problems for infinite-state transition systems [9,17,18,20,21,7,8,10,19].

In this paper, we improve the above results by showing that the pushdown stack can be eliminated from the acceptors. Specifically, we show that the binary (backward or forward) reachability can be accepted by a two-tape (one-tape) finite-state acceptor with reversal-bounded counters. In fact, we show that the results hold, even if the discrete pushdown timed automaton is augmented with reversal-bounded counters. Note that

equipping the pushdown timed automaton with counters is an important and nontrivial generalization, since it is known that reversal-bounded counters can "verify" Presburger relations on clock values [16]. This stack elimination technique can also be applied to the case of real-valued clocks by using the recent "pattern technique" in [7] that treats a real-valued clock as an integral part and a fractional part. Fractional parts of clocks are discretized by a clock pattern [7].

The results in this paper can be used to verify properties that were not verifiable before using previous techniques. For example, we can now show that $pre^*(M,S)$, where M is a discrete pushdown timed automaton (possibly augmented with reversal-bounded counters) and S is a set of configurations accepted by a pushdown acceptor (possibly augmented with reversal-bounded counters), can be accepted by a pushdown acceptor with reversal-bounded counters. Hence, the emptiness of $pre^*(M,S)$ is decidable. Note that a direct construction of a machine accepting this set would require two stacks, and this will not yield a decidable emptiness since, in general, a two-stack machine can simulate a Turing machine. As another example, consider the satisfiability checking (the dual of model checking) of a property $\Diamond(P_1 \wedge \Diamond P_2)$ concerning a discrete pushdown timed automaton with reversal-bounded counters M, where P_1 and P_2 are Presburger formulas on stack symbol counts and clock and counter values. This problem is reducible to checking the emptiness of $pre^*(M, pre^*(M,S_2) \cap S_1)$, where S_1 (respectively, S_2) is the set of all configurations satisfying P_1 (respectively, P_2). Now S_1 and S_2 can be accepted by finite-state acceptors with reversal-bounded counters [16]. Hence, using our new result, we can construct a finite-state acceptor with reversal-bounded counters accepting $pre^*(M,S_2)$, and then construct from this another such machine accepting $pre^*(M,S_2) \cap S_1$. Finally, we construct a finite-state acceptor with reversal-bounded counters accepting $pre^*(M, pre^*(M,S_2) \cap S_1)$, from which the decidability of emptiness follows. Again, a direct construction would require two stacks: one is for the machine accepting $pre^*(M,S_2) \cap S_1$, and the other for accepting $pre^*(M, pre^*(M,S_2) \cap S_1)$, and this does not yield the decidability of emptiness.

We also look at discrete timed automata with reversal-bounded counters, and a read/write worktape (instead of a pushdown stack), but restricted to be *finite crossing*, i.e., in any computation, the number of times the read/write head crosses the boundary between any two adjacent worktape cells is bounded by a given constant. We show that the binary (backward or forward) reachability set of this machine can also be accepted by a two-tape (one-tape) finite-state acceptor with reversal-bounded counters. This improves the corresponding results in [18] where the acceptors needed a finite-crossing read/write tape. Note that without the "finite-crossing" requirement, the model becomes a Turing machine.

The boundedness problem for reachability sets is that of deciding, given a machine M in a class C_1 and a set of configurations S accepted by an acceptor in a class C_2, whether $R_\alpha(M) = \{\beta \mid \text{configuration } \beta \text{ is reachable from } \alpha\}$ is finite for every configuration α in S. We give a decision procedure for this problem for various C_1's and C_2's, e.g., when C_1 is the class of discrete pushdown timed automata with reversal-bounded counters, and C_2 is the class of pushdown acceptors with reversal-bounded counters.

We will use the following notation. We use the suffix 'M' to indicate that the model has no input tape and 'A' when the model has one-way input tape(s). All models are

nondeterministic.

(1) PM: Pushdown machine.
(2) PA: Pushdown acceptor.
(3) PCM: Pushdown machine with reversal-bounded counters.
(4) PCA: Pushdown acceptor with reversal-bounded counters.
(5) FM: Finite-state machine.
(6) FA: Finite-state acceptor.
(7) CM: Finite-state machine with reversal-bounded counters.
(8) CA: Finite-state acceptor with reversal-bounded counters.
(9) WCM: Finite-state machine with a read/write worktape and reversal-bounded counters.
(10) WCA: Finite-state acceptor with a read/write worktape and reversal-bounded counters.
(11) k-tape PCA (FA, CA, WCA) is a PCA (FA, CA, WCA) with k input tapes, with one head per tape. A 1-tape PCA (FA, ...) will simply be referred to as a PCA (FA, ...).
(12) PTCM (WTCM) is a PCM (WCM) augmented with discrete clocks.

The importance of multi-tape PCAs and finite-crossing WCAs is that the emptiness problem for these acceptors is decidable [16–18,20,15].

The paper has six section in addition to this section. Section 2 shows that the binary reachability of a PCM can be accepted by a two-tape CA and that the backward and forward reachability sets can be accepted by CAs. Section 3 shows that these results hold for finite-crossing WCMs. Section 4 generalizes the results to PCMs and finite-crossing WCMs with "clocks" (i.e., the timed versions of the models). Section 5 proposes a subset of Presburger LTL whose satisfiability checking is decidable. Section 6 discusses the boundedness problem. Section 7 (conclusion) explains how the pushdown stack (or finite-crossing tape) elimination technique can be applied to machines with real-valued clocks.

2. PCMs

We first look at the simple case of a PM (pushdown machine without counters). We assume that the pushdown stack has a "bottom" symbol B_0, and is associated with two kinds of stack operations: push(q,Z,q'), i.e., push symbol Z onto the stack and switch from state q to state q', and pop(q,Z,q'), i.e., pop the top Z from the stack and switch from state q to state q'. Replacing the top symbol of the stack with another symbol can be implemented by a push followed by a pop.

Let M be a PM. Define predicates push* and pop* as follows: push$^*(q,T,Z,q')$ is true, if there is a sequence of moves of M such that, starting from state q with stack top symbol T, M does not pop this T, and the last move is a push of Z on top of this T ending in state q'. (Notice that, prior to this last move, the sequence may involve many pushes/pops.) Similarly, pop$^*(q,Z,T,q')$ is true, if there is a sequence of moves of M such that, starting from state q with stack top symbol T (and Z the symbol directly under T), M does not pop this Z and the result of the moves makes this Z

the top of the stack and state q'. We also define stay$^*(q,T,q')$ to be true if M, starting from state q with stack top symbol T, can reach state q' without performing any stack operations.

Lemma 2.1. *Given a PM M, we can effectively compute the predicates* push*, pop*, *and* stay*.

Proof. First consider push*. Given a tuple (q,T,Z,q'), it easy to construct from M a PA (pushdown acceptor) $A(q,T,Z,q')$ such that push$^*(q,T,Z,q')$ iff $A(q,T,Z,q')$ accepts a nonempty language. The result follows since the emptiness problem for PAs is decidable. Similarly, we can handle pop* and stay*. □

Let M be a PM, and S,T be two sets of configurations. Define $Binary(M,S,T) = \{(\alpha,\beta) \mid \text{configuration } \alpha \in S \text{ can reach configuration } \beta \in T \text{ in } M\}$. When $S = T =$ the set of all configurations, $Binary(M,S,T)$ will simply be written $Binary(M)$.

Theorem 2.2. *$Binary(M)$ of a PM M can be accepted by a 2-tape FA.*

Proof. From definition, $Binary(M) = \{(wq, w'q') \mid \text{configuration } wq \text{ can reach configuration } w'q' \text{ in } M\}$. We construct a two-tape FA A to accept $Binary(M)$. A incorporates the (finite) predicates push*, pop* and stay* in its states. Given $Z_1 \cdots Z_k q$ and $Z'_1 \cdots Z'_{k'} q'$ on the first and the second tape, respectively, A works as follows. A reads the two tapes in parallel and makes sure that the symbol under head 1 is the same as the symbol under head 2. Nondeterministically, A starts to operate in the following way. Assume, now, both heads are at the mth ($m \geqslant 1$) cell with $Z_1 \ldots Z_{m-1} = Z'_1 \cdots Z'_{m-1}$. There are four cases to consider (nondeterministically chosen):

Case 1: $m \leqslant k$ and $m \leqslant k'$.
Case 2: $m \leqslant k$ and $m = k' + 1$.
Case 3: $m = k + 1$ and $m \leqslant k'$.
Case 4: $m = k + 1 = k' + 1$.

For Case 1, A reads the input of the first tape and guesses a sequence of tuples (when $m = 1$, treat Z_{m-1} as the stack bottom B_0)

$$(q_0, Z_{m-1}, Z_m, q_1), \ldots, (q_{k-m}, Z_{k-1}, Z_k, q_{k-m+1})$$

such that pop$^*(q_{i+1}, Z_{i+m-1}, Z_{i+m}, q_i)$ for $i = 0,\ldots,k-m$, and $q_{k-m+1} = q$. That is, configuration $Z_1 \cdots Z_k q$ can reach configuration $Z_1 \cdots Z_{m-1} q_0$ through a sequence of moves that never pops symbol Z_{m-1} out. A then reads the rest of the second input tape and guesses a sequence of tuples

$$(p_0, Z'_{m-1}, Z'_m, p_1), \ldots, (p_{k'-m}, Z'_{k'-1}, Z'_{k'}, p_{k'-m+1})$$

such that $p_0 = q_0$, push$^*(p_i, Z'_{i+m-1}, Z'_{i+m}, p_{i+1})$ for each $i = 0,\ldots,k'-m$, and $p_{k'-m+1} = q'$. That is, configuration $Z'_1 \ldots Z'_{m-1} p_0$ (which is exactly $Z_1 \cdots Z_{m-1} q_0$) can reach configuration $Z'_1 \cdots Z_{k'} q'$. A accepts if all the guesses are successful, i.e., $Z_1 \cdots Z_k q$ can reach $Z'_1 \cdots Z_{k'} q'$.

The remaining three cases can be handled similarly. Hence $Binary(M)$ can be accepted by a two-tape FA. □

Corollary 2.3. *Let M be a PM, and, S and T be sets of configurations of M accepted by FAs. Then $Binary(M,S,T)$ can be accepted by a two-tape FA.*

Proof. From Theorem 2.2, $Binary(M)=\{(\alpha,\beta)\mid \alpha,\beta$ are configurations and β is reachable from $\alpha\}$ can be accepted by a two-tape FA A. Clearly, using the FAs A_S and A_T accepting S and T, we can modify the two-tape FA A to a two-tape FA B accepting $Binary(M,S,T)$. □

The backward and forward reachability sets of a PM M with respect to a regular set of configurations S is regular [3,13,11]. This result is easily obtained from the corollary above.

Corollary 2.4. *Let M be a PM and S be a set of configurations of M accepted by an FA A_S. Then $pre^*(M,S)=\{\alpha\mid$ configuration α can reach some configuration β in $S\}$ and $post^*(M,S)=\{\beta\mid$ configuration β is reachable from some configuration α in $S\}$ can be accepted by FAs.*

Proof. Let T be the set of all configurations of M. It is straightforward to construct from the two-tape FA B accepting $Binary(M,S,T)$ an FA accepting $post^*(M,S)$, which is just the projection of $Binary(M,S,T)$ on the second coordinate. $pre^*(M,S)$ can be handled similarly. □

We now consider the PCMs. The reversal-bounded counters in the PCMs complicate the constructions, since now we have to incorporate counters into the acceptors of the reachability sets. We need to prove some intermediate results.

Let $\mathbb{N}$ be the set of nonnegative integers and n be a positive integer. A subset S of $\mathbb{N}^n$ is a *linear set* if there exist vectors $\vec{v}_0,\ldots,\vec{v}_t$ in $\mathbb{N}^n$ such that

$$S=\{\vec{v}\mid\vec{v}=\vec{v}_0+a_1\vec{v}_1+\ldots+a_t\vec{v}_t,\ \forall 1\leqslant i\leqslant t,\ a_i\in\mathbb{N}\}.$$

The vectors $\vec{v}_0$ (the constant vector) and $\vec{v},\ldots,\vec{v}_t$ (the periods) are called *generators*. A set S is *semilinear* if it is a finite union of linear sets. Semilinear sets are precisely the sets definable by *Presburger formulas* [14].

If $S\subseteq\mathbb{N}^n$ and $a_1,\ldots,a_n$ are distinct symbols, let $L(S)=\{a_1^{i_1}\ldots a_n^{i_n}\mid(i_1,\ldots,i_n)\in S\}$. We say that an automaton A accepts the relation S if $L(S)$ is accepted by A. The following result is from [16].

Lemma 2.5. *The following statements are (effectively) equivalent for $S\subseteq\mathbb{N}^n$:*
(1) *S is a semilinear set.*
(2) *$L(S)$ can be accepted by a CA.*
(3) *$L(S)$ can be accepted by a PCA.*

A language L is *bounded* if it is a subset of $a_1^* \cdots a_n^*$, for some (not necessarily distinct) symbols $a_1, \ldots, a_n$. However, we assume that a_i is different from a_{i+1} for $1 \leqslant i \leqslant n-1$. For example, a language $\subseteq a_1^* a_2^* a_1^* a_2^* a_3^* a_1^*$ is bounded. But $(a_1 + a_2)^*$ is not bounded. For a bounded language $L \subseteq a_1^* \cdots a_n^*$, there corresponds a subset $S(L)$ of $\mathbb{N}^n$ defined by $S(L) = \{(i_1, \ldots, i_n) \mid a_1^{i_1} \ldots a_n^{i_n} \in L\}$.

We can easily show the following result from Lemma 2.5:

Lemma 2.6. *The following statements are equivalent for a bounded language L:*
(1) *$S(L)$ is a semilinear set.*
(2) *L can be accepted by a CA.*
(3) *L can be accepted by a PCA.*

There is a simple automaton that characterizes semilinear sets. Let M be a nondeterministic finite-state machine (*without* an input tape) with n counters for some $n \geqslant 1$. The computation of M starts with all the counters zero and the automaton in the start state. An atomic move of M consists of incrementing at most one counter by 1 and changing the state (decrements are not allowed). An n-tuple $\vec{v} = (i_1, \ldots, i_n) \in \mathbb{N}^n$ is generated by M if M, when started from its initial configuration, halts in an accepting state with $\vec{v}$ as the contents of the counters. The set of all n-tuples generated by M is denoted by $G(M)$. We call this machine a C-generator. If the C-generator is augmented with a pushdown stack, the machine is called a PC-generator. Notice that counters in a generator are nondecreasing, i.e., 0-reversal bounded. We will need the following lemma.

Lemma 2.7. *The following statements are equivalent for $S \subseteq \mathbb{N}^n$:*
(1) *S is a semilinear set.*
(2) *S can be generated by a C-generator.*
(3) *S can be generated by a PC-generator.*

Proof. From the definition of a semilinear set S, it is straightforward to construct a C-generator M such that $G(M) = S$. Now suppose M is a C-generator (PC-generator) with n counters. We construct a CA (PCA) with n reversal-bounded counters M' which operates as follows. Given an input x, M' first simulates M by generating in n counters an n-tuple $(i_1, \ldots, i_n)$. Then M' checks and accepts if the input $x = a_1^{i_1} \cdots a_n^{i_n}$, where the a_i's are distinct symbols. Clearly, $S(L(M')) = G(M)$, and therefore, by Lemma 2.6, $G(M)$ is a semilinear set. □

Consider a PCM M with n counters. A configuration of M is now represented as a string $\alpha = wqd_1^{x_1} d_2^{x_2} \cdots d_n^{x_n}$, where w is the stack content, q is the state, $d_1, \ldots, d_n$ are distinct symbols, and $x_1, x_2, \ldots, x_n$ are the values of the counters (thus the counter values are represented in unary). We will show that the binary reachability $Binary(M)$ can be accepted by a two-tape CA.

To simplify matters, we convert M to another PCM M' with many more counters than M. Assume M starts from a configuration α and reaches another configuration β. M' operates like M, except that the counters can make at most one reversal. M'

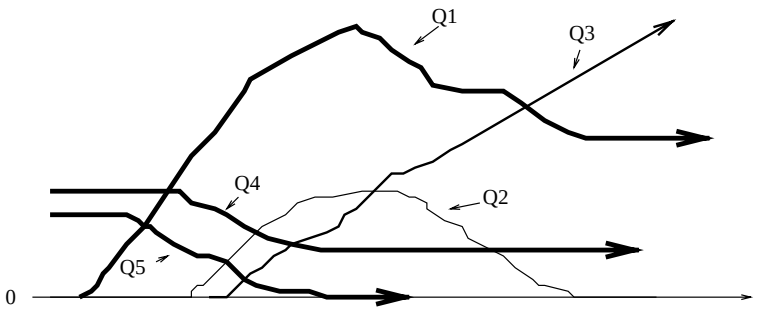

Fig. 1. Behavior patterns for a normalized counter.

simulates M faithfully, except that when a counter c of M makes a reversal from nonincreasing to increasing or c starts to increment before any decrements were made after starting from configuration α, M' suspends the simulation but continues decreasing this counter to zero while simultaneously increasing a new counter c' (starting at zero). When c reaches zero, c' has the old value of c before the simulation was suspended. M' then resumes the simulation with c' taking the role of c. If c' later reverses from nonincreasing to increasing, a new counter c'' is deployed like before. In this way, each counter c of M making r reversals can be replaced by $(r+1)/2$ counters $c_1,\ldots,c_{(r+1)/2}$, where each one makes at most one reversal. Moreover, a configuration α of M translates to a corresponding configuration of M', where the value of a counter c of M is identified with the value of one of the counters $c_1,\ldots,c_{(r+1)/2}$. Clearly, if we can construct a two-tape CA A' to accept $Binary(M')$, we can modify A' to a two-tape CA A to accept $Binary(M)$.

Let M be a PCM. From the discussion above, we assume that the counters have been normalized, i.e., during the computation from one configuration to another, each counter c behaves as one of the following five patterns (Fig. 1):

- c starts at zero, becomes positive, reverses (i.e., decrements), but remains positive.
- c starts at zero, becomes positive, reverses, becomes zero (and remains zero).
- c starts at zero, becomes positive, and does not reverse.
- c starts at a positive value, remains nonincreasing and positive.
- c starts at a positive value, remains nonincreasing, becomes zero (and remains zero).

We do not include the case when a counter remains at zero during the entire computation, since this can be simulated by an increment by 1 followed by a decrement by 1. Call the behaviors above Q_1, Q_2, Q_3, Q_4 and Q_5, respectively.

Consider a counter c that has behavior Q_1. During the computation, c makes a mode change at three different instances: when it started at 0, became positive, and when it reversed. We denote these instances by $0,+,rev$. Note that c is positive at the end of the computation, since it has behavior Q_1. In the construction to be described below, c will be simulated by two counters, c^+ and c^-, the first to record increments and the second to record decrements. If c is tested for zero during any segment of the simulation, the simulator assumes that it is zero before the mode changes to $+$ and positive after the mode has changed to $+$. (Note that the simulator knows when the

mode changes.) At the end of the entire simulation, the simulator verifies that c is indeed positive by checking that $c^+ - c^-$ is positive.

Similarly, for a counter c with behavior Q_2, the mode-change instances are: $0, +$, *rev*, *zero*. As in the above case, c will be simulated by two counters c^+ and c^-, and the simulator's action when testing for zero is like in the above case before the mode changes to *zero*. The point when the counter becomes zero (i.e., the mode changes to *zero*) is "guessed" by the simulator. After the mode has changed to *zero*, the simulator assumes that the counter is always zero when it is being tested (and c^+ and c^- will remain the same in the rest of the computation). At the end of the simulation, the simulator verifies that c is zero by checking that $c^+ = c^-$.

For the case of a counter c with behavior Q_3, the mode-change instances are $0, +$. Like in the case for Q_1, the simulator assumes the counter is zero before the mode changes to $+$ and positive after the mode has changed to $+$. Then c^+ is exactly c, and c^- will remain zero during the entire simulation. Note that for this case, there is nothing to verify at the end of the simulation.

For the case for Q_4, the mode-change instance is *rev*. Counter c stays positive and c^+ will remain zero during the entire computation. The simulator checks that c^- is less than the starting value of c.

For the case of a counter c with behavior Q_5, the mode-change instances are *rev*, *zero*. The simulator assumes the counter is positive before the mode changes to *zero*. Notice that the point that c becomes zero can be guessed by the simulator as described in the case for Q_2. c^+ will remain zero during the entire simulation. Then the simulator checks that c^- is exactly the starting value of c.

When we say that a counter starts with mode m and ends with mode m' in a certain segment of the computation, we mean:

(1) The counter is already in mode m, i.e., the mode-change to m has already been executed earlier;
(2) If $m' \neq m$, the mode-change to m' occurs during the segment of computation under consideration.

In describing a subcomputation of the machine, we refer to $\langle c, Q_i, m, m' \rangle$ as a mode vector for c, and this means that counter c has behavior Q_i $(i = 1, 2, 3, 4, 5)$ and in the subcomputation, c starts with mode m and ends with mode m'. We denote $\langle c, Q_i, m, m' \rangle$ simply as $V(c, Q_i)$, when m and m' are understood.

Let M be a PCM with n counters: $c_1, \ldots, c_n$. We associate with each counter c two counters c^+ and c^-. Given $Q_{i_1}, \ldots, Q_{i_n}, q, Z, q'$ (each $i_j \in \{1, 2, 3, 4, 5\}$), define a set of $2n$-tuples of nonnegative integers $\text{push}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), T, Z, q')$ as follows: $(u_1, \ldots, u_n, v_1, \ldots, v_n)$ is in $\text{push}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), T, Z, q')$ if there is a sequence of moves of M such that,

(1) The computation starting from state q with stack top symbol T, M does not pop this T, and the last move is a push of Z on top of this T ending in state q' (notice that, prior to this last move, the sequence may involve many pushes/pops).
(2) The computation remains within the specified mode vectors of the counters.
(3) For $i = 1, \ldots, n$, u_i (v_i) is the number of times counter c_i is incremented (decremented) by 1. So, for example, for $V(c_1, Q_2, 0, 0)$, $u_1 = 0$ and $v_1 = 0$; for $V(c_1, Q_2, 0, +)$, $u_1 > 0$ and $v_1 = 0$; for $V(c_1, Q_2, 0, rev)$, $u_1 > 0$ and $v_1 > 0$; for $V(c_1, Q_2, rev,$

rev), $u_1 = 0$ and $v_1 \geqslant 0$; for $V(c_1, Q_2, rev, zero)$, $u_1 = 0$ and $v_1 \geqslant 0$; for $V(c_1, Q_2, zero, zero)$, $u_1 = 0$ and $v_1 = 0$, etc.

Thus, $\text{push}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), T, Z, q')$ gives separate counts of the total increments and total decrements for each counter of M during the computation.

Similar to $\text{pop}^*(q, Z, T, q')$ and $\text{stay}^*(q, T, q')$ for a PM, we can define the sets $\text{pop}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), Z, T, q')$ and $\text{stay}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), T, q')$.

Lemma 2.8. *We can construct C-generators for* $\text{push}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), T, Z, q')$, $\text{pop}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), Z, T, q')$, *and* $\text{stay}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), T, q')$.

Proof. First we construct a PC-generator B with $2n$ counters which simulates the computation of M starting in state q with its stack top T. During the simulation, B makes sure that items 1 and 2 are satisfied. The simulation halts when M writes Z on the top of symbol T and moves right in state q'. From Lemma 2.7, B can be converted to an equivalent C-generator for

$$\text{push}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), T, Z, q').$$

Similarly, we can construct C-generators for

$$\text{pop}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), Z, T, q')$$

and $\text{stay}^*(q, V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}), T, q')$. □

For notational convenience, $(V(c_1, Q_{i_1}), \ldots, V(c_n, Q_{i_n}))$ will simply be denoted by V and will be called a global mode vector. Note that there are only a finite number of distinct global mode vectors. We use $A_{\text{push}}(q, V, T, Z, q')$, $A_{\text{pop}}(q, V, Z, T, q')$, and $A_{\text{stay}}(q, V, T, q')$ to denote the C-generators for $\text{push}^*(q, V, T, Z, q')$, $\text{pop}^*(q, V, Z, T, q')$ and $\text{stay}^*(q, V, T, q')$, respectively.

Let V and V' be two global mode vectors. Let $\langle c, Q_i, m, m' \rangle$ be a mode vector for c in V and $\langle c, Q_j, m'', m''' \rangle$ the corresponding mode vector for c in V'. We say that V and V' are compatible with respect to counter c if $Q_i = Q_j$, $m' = m''$, and m''' is a proper mode for Q_i (so, e.g., *rev* and *zero* are not proper for Q_3; *zero* is not proper for Q_1). Two global mode vectors V and V' are compatible if they are compatible with respect to every counter c. We are now ready to prove:

Theorem 2.9. *Binary*(M) *of a PCM* M *can be accepted by a two-tape CA.*

Proof. From definition, $Binary(M) = \{(\alpha, \beta) \mid \text{configuration } \alpha \text{ can reach configuration } \beta \text{ in } M\}$. We construct a two-tape CA B to accept $Binary(M)$. The specifications of all the C-generators $A_{\text{push}}(q, V, T, Z, q')$, $A_{\text{pop}}(q, V, Z, T, q')$, and $A_{\text{stay}}(q, V, T, q')$ are incorporated in the states of B. We describe the operation of B when given configurations α and β on its first and second input tapes, respectively. Let $\alpha = wqd_1^{x_1} \cdots d_n^{x_n}$ and $\beta = w'q'd_1^{x_1'} \cdots d_n^{x_n'}$. Let $w = Z_1 \ldots Z_k$ and $w' = Z_1' \ldots Z_{k'}'$.

B reads the two tapes in parallel and makes sure that the symbol under head 1 is the same as the symbol under head 2. Nondeterministically, B starts to operate

in the following way. Assume that both heads are at the mth ($m \geqslant 1$) cell with $Z_1 \cdots Z_{m-1} = Z'_1 \cdots Z'_{m-1}$. There are four cases to consider (nondeterministically chosen):

Case 1: $m \leqslant k$ and $m \leqslant k'$.
Case 2: $m \leqslant k$ and $m = k' + 1$.
Case 3: $m = k + 1$ and $m \leqslant k'$.
Case 4: $m = k + 1 = k' + 1$.

Consider Case 1. B operates in two phases. In the first phase, B reads the rest of the first input tape and guesses a sequence of pop-generators (when $m = 1$, treat Z_{m-1} as the stack bottom B_0)

$$A_{\text{pop}}(q_0, V_0, Z_{m-1}, Z_m, q_1), \ldots, A_{\text{pop}}(q_{k-m}, V_{k-m}, Z_{k-1}, Z_k, q_{k-m+1})$$

such that V_{i+1} and V_i are compatible and each $\text{pop}^*(q_{i+1}, V_{i+1}, Z_{i+m-1}, Z_{i+m}, q_i)$ is not empty for $i = 0, \ldots, k-m$, and $q_{k-m+1} = q$. Further, V_{k-m} is consistent with the starting counter values $x_1, \ldots, x_n$, e.g., if the behavior of counter c_1 in V_{k-m} is Q_2 (c_1 starts from 0), then x_1 must be 0. Note that each counter c in M is associated with two counters c^+ and c^- in the C-generators to keep track of the increments and decrements in counter c. In order to decide the counter values at the beginning of the second phase below, B guesses the value y_i for each counter c_i, and verifies, at the end of phase 1 by using auxiliary counters, that $y_i + \Sigma c_i^- - \Sigma c_i^+ = x_i$ where Σc_i^+ (resp. Σc_i^-) is the total increments (decrements) made to counter c_i for all the pop generators in phase 1. Note that "increments" in each pop generator essentially means "decrements" to y_i, since the pop generators are supposed to change the values of the c_i's from x_i's to y_i's. Doing this ensures that configuration $Z_1 \cdots Z_k q d_1^{x_1} \ldots d_n^{x_n}$ can reach the intermediate configuration γ (i.e., $Z_1 \cdots Z_{m-1} q_0 d_1^{y_1} \ldots d_n^{y_n}$) through a sequence of moves that never pops symbol Z_{m-1}.

Now, B starts phase 2, with counter values $y_1, \ldots, y_n$ for counters $c_1, \ldots, c_n$ in M. B then reads the rest of the second input tape and guesses a sequence of push generators

$$A_{\text{push}}(p_0, U_0, Z'_{m-1}, Z'_m, p_1), \ldots, A_{\text{push}}(p_{k'-m}, U_{k'-m}, Z'_{k'-1}, Z'_{k'}, p_{k'-m+1})$$

such that $p_0 = q_0$ and, V_0 and U_0 are compatible (i.e., M continues its computation from the intermediate configuration γ that was reached from the starting configuration α). B also checks that U_i and U_{i+1} are compatible and each $\text{push}^*(p_i, U_i, Z'_{i+m-1}, Z'_{i+m}, p_{i+1})$ is not empty for $i = 0, \ldots, k'-m$, and $p_{k'-m+1} = q'$. In order to verify the intermediate configuration γ can reach configuration β, B needs to check (similar to phase 1) that $y_i - \Sigma c_i^- + \Sigma c_i^+ = x'_i$ where Σc_i^+ (resp. Σc_i^-) is the total increments (decrements) made to counter c_i for all the push generators in phase 2. Finally, B needs to check that the ending counter values $x'_1, \ldots, x'_n$ are consistent with the last mode vector $U_{k'-m}$. For instance, if counter c_1 has behavior pattern Q_4 in $U_{k'-m}$, then x'_1 must be positive. B accepts if all the guesses are successful, i.e., α can reach β. Counters in B are reversal bounded. This is because, the counters in C-generators are 0-reversal, and the checking for counter values (such as $y_i + \Sigma c_i^- - \Sigma c_i^+ = x_i$ in phase 1 and $y_i - \Sigma c_i^- + \Sigma c_i^+ = x'_i$ in phase 2) needs only finitely many counter reversals (with auxiliary reversal-bounded counters). Therefore B is indeed a two-tape CA.

Cases 2–4 are handled similarly, where the C-generators $A_{\text{stay}}(q,V,T,q')$ for stay* (q,V,T,q') are also used in the construction.

Hence, $Binary(M)$ can be accepted by a two-tape CA B. □

As in Corollaries 2.3 and 2.4, we have:

Corollary 2.10. *Let M be a PCM and S and T be sets of configurations of M accepted by CAs. Then $Binary(M,S,T)$ can be accepted by a two-tape CA, and $pre^*(M,S)$ and $post^*(M,S)$ can be accepted by CAs.*

3. Finite-crossing WCMs

In this section we show that the binary reachability of a finite-crossing WCM can be accepted by a two-tape CA. First we consider a special case of binary reachability, called simply, *reachability*, which is defined as follows for a finite-crossing WCM M: $R(M)=\{\alpha \mid \alpha \text{ is reachable from the initial configuration}\}$, where the initial configuration corresponds to the start state, blank worktape, and zero counters.

Let M be a finite-crossing WCM with n counters. A configuration of M is a tuple $\alpha=(w_1qw_2,X)$, where $w=w_1w_2$ is the content of the read/write worktape with the head at the leftmost symbol of w_2, the state is q, and X the values of the counters. Note that α can be represented as a string $w_1qw_2d_1^{x_1}d_2^{x_2}\cdots d_n^{x_n}$, where $d_1,\ldots,d_n$ are symbols distinct from the states and worktape symbols, and $x_1,x_2,\ldots,x_n$ are the values of the counters.

We say that a finite-crossing WCM M is *nonsitting*, if in any computation of M the read/write head does not "sit" on any tape cell, i.e., it always moves left or right of a cell in every step. Note that M' cannot just simulate a sitting step by a left (or right) move followed by a right (or left) move. This is because the read/write head can sit on a cell an unbounded number of steps (because this depends on the values of the counters), and this would make M' not finite crossing. However, by using a "dummy" symbol, say #, on the worktape, M can be converted to a nonsitting M' [18]. We describe the idea briefly. M' begins by writing a finite-length sequence of #'s on the worktape, the length being chosen nondeterministically. Then M' simulates M, but whenever M writes a symbol on a *new* tape cell, M also writes to the right of this cell a finite-length sequence of #'s, the length of which is chosen nondeterministically. Thus, at any time, the worktape contains a word where every pair of nondummy symbols is separated by a word of #'s. During the simulation, M' moves on the #'s to simulate the sitting moves of M, which is possible if there are enough #'s between any pair of nondummy symbols. To simulate a nonsitting move of M, M' may need to "skip over" the #'s to get to the correct nondummy symbol. Clearly, if $R(M')$ can be accepted by a CA, then we can construct from this CA another CA accepting $R(M)$.

Theorem 3.1. *Let M be a finite-crossing WCM. Then $R(M)$ can be accepted by a CA A.*

Proof. We may assume that M is nonsitting. We will construct a CA A accepting $R(M)$. We describe the operation of A when given an input string α (we can assume that α is a valid configuration since the set of all valid configurations is a regular set).

As in Section 2 we assume that the counters of M have been normalized, i.e., during the computation from the initial configuration to α, each counter c behaves as follows:

- c starts at zero, becomes positive, reverses, but remains positive.
- c starts at zero, becomes positive, reverses, becomes zero (and remains zero).
- c starts at zero, becomes positive, and does not reverse.

As in Section 2 call the behavior patterns above Q_1, Q_2, Q_3. Note that the other behavior patterns do not occur, since the computation starts from the initial configuration.

A begins by guessing the behavior of each counter c of M as defined above.

Number the worktape cells by $1, 2, \ldots$ from left to right. Consider a computation of M from the initial configuration to α. Let n be the rightmost worktape cell that M visits. Now look at cell p of the worktape, $p = 1, 2, \ldots, n$. In the computation, cell p may be visited several times, Let $t_1, \ldots, t_m$ be the times M visits p.

Corresponding to the time sequence $(t_1, \ldots, t_m)$ associated with p, we define a *crossing vector* $V = (I_1, \ldots, I_m)$, where for each i, $I_i = (d_1, q_1, r_1, r_2, d_2)$,

- $d_1 \in \{-1, +1\}$ is the direction from which the head entered p at time t_i;
- q_1 is the state when M entered p;
- r_1 is the instruction that was used in the move above;
- r_2 is the instruction that was used at time $t_i + 1$ when M left p;
- $d_2 \in \{-1, +1\}$ is the direction to which M left p at time $t_i + 1$.

Note that instructions r_1 and r_2 specify the updates to be made on the counters and the worktape cells.

A uses two counters for each counter with behavior Q_1 or Q_2, and one counter for each counter with behavior Q_3. A simulates the computation of M by nondeterministically guessing the sequence of crossing vectors $V_1, \ldots, V_n$ as A processes the worktape from left to right, making sure that V_j and V_{j+1} are compatible for $1 \leqslant j \leqslant n-1$. The testing for zero of a counter and the "guessing" when a counter with behavior Q_2 becomes zero are handled like in the construction for the case of the PCM. From these vectors, A can also determine the last symbol that each cell is rewritten with and where the read/write head stops. Thus, A can match these symbols against the input tape. During the computation each counter c with behavior Q_1 or Q_2 will be simulated by two counters c^+ and c^-. The increments ($+1$'s) in counter c specified in the vectors will be interpreted as adding 1's to counter c^+. Similarly, the decrements (-1's) in counter c in the vectors correspond to adding 1's to counter c^-. After V_n has been examined, A computes $d = c^+ - c^-$ for each counter c with behavior Q_1 and checks that the value of c in α is d. Similarly, A checks that $c^+ = c^-$ for each counter c with behavior Q_2 and that the value of c in α is zero. Finally, A checks that the final value of each counter c with behavior Q_3 matches that in configuration α. If all checks out, A accepts α. □

We now modify the construction above to show the following:

Theorem 3.2. *Let M be a finite-crossing WCM. Then $Binary(M)$ can be accepted by a two-tape CA.*

Proof. We may assume that M is nonsitting. Let C be the set of n counters of M. We first construct another finite-crossing WCM M' which has $2n$ counters: the counters in C and duplicate counters C'. M' operates in three phases:

(1) Starting from the initial configuration, M' writes w_1qw_2 on the worktape (w_1, w_2, q are chosen nondeterministically). It then nondeterministically loads the counters in C with some initial values, making duplicate copies in the counters in C'. While loading the counters, M' moves its read/write head to the right at every step, writing dummy symbols \$'s. At some point, M' enters phase 2.

(2) When this phase is entered, the worktape contains the string $w_1qw_2\$^k$ for some k, and the counters in C and C' contain values X. M' modifies the worktape (by moving left and erasing the \$'s, etc.) so that the worktape contains w_1w_2 with the read/write head on the leftmost symbol of w_2. Then M' enters phase 3.

(3) M' simulates M starting in configuration (w_1qw_2, X), using the counters in C' to simulate the counters of M, leaving the counters in C unchanged.

Note that M' is nonsitting, and the counters in C are 0-reversal. As in the proof of Theorem 3.1, we can convert M' to another nonsitting finite-crossing WCM M'', where each counter is at most 1-reversal. The counters of M'' will consist of the counters in C (since they are 0-reversal) and counters C'' (these are counters in C' and other counters that are added to make all counters at most 1-reversal).

We now describe the two-tape CA A accepting $Binary(M'')$. Given (α, β) on its two input tapes, A operates as described in the proof of Theorem 3.1 to check that the target configuration β is reachable. At the time when A is guessing the sequence of crossing vectors, as it processes the worktape from left to right, it also checks that the first non-\$ symbols written on the worktape (which constitute w_1qw_2) match those in α. At the end of the simulation, A also checks that the values X in counters C (which are left unchanged in M' and, hence, also in M'') are the values specified in α. Clearly, from the two-tape CA A accepting $Binary(M'')$, we can construct a two-tape CA B accepting $Binary(M)$. □

As in Corollary 2.10, we can show:

Corollary 3.3. *Let M be a finite-crossing WCM and S and T be sets of configurations of M accepted by CAs. Then $Binary(M, S, T)$ can be accepted by a two-tape CA, and $pre^*(M, S)$ and $post^*(M, S)$ can be accepted by CAs.*

One can check from the above constructions that when there are no counters in the machines, the preceding corollary reduces to:

Corollary 3.4. *Let M be a finite-crossing WCM without counters (i.e., the only memory structure is a finite-crossing read/write worktape) and S and T be*

regular sets. Then Binary(M,S,T) *can be accepted by a two-tape FA, and pre*$^*(M,S)$ *and post*$^*(M,S)$ *can be accepted by FAs.*

4. WCMs and PCMs with clocks

A timed automaton is a finite-state machine *without* an input tape augmented with finitely many real-valued unbounded clocks [2]. All the clocks progress synchronously with rate 1, except that when a nonempty subset of clocks are reset to 0 at some transition, the other clocks do not progress. A transition between states fires if a clock constraint is satisfied. A clock constraint is a Boolean combination of atomic clock constraints in the following form: $x\#c, x-y\#c$ where # denotes $\leqslant, \geqslant, <, >$, or $=$, c is an integer, x, y are clocks. Here we only consider integer-valued clocks, i.e., discrete timed automata. A discrete pushdown timed automaton (finite-crossing work-tape timed automaton) is a discrete time automaton with a pushdown stack (finite-crossing read/write tape). We can further generalize these models by augmenting them with reversal-bounded counters, call them PTCM and finite-crossing WTCM, respectively. Thus, a PTCM (finite-crossing WTCM) is a PCM (finite-crossing WCM) with clocks. A configuration of a PTCM (finite-crossing WTCM) now contains the values of the clocks. It is known that the binary reachability of a PTCM (finite-crossing WTCM) can be accepted by a two-tape PCA (finite-crossing WCA) [9,20]. Following the constructions in [9,20] and in the previous section we can prove:

Theorem 4.1. *Let M be a PTCM (or a finite-crossing WTCM). Then Binary*(M) *can be accepted by a two-tape CA.*

Corollary 4.2. *Let M be a PTCM (or a finite-crossing WTCM) and S and T be sets of configurations of M accepted by CAs. Then Binary*(M,S,T) *can be accepted by a two-tape CA, and pre*$^*(M,S)$ *and post*$^*(M,S)$ *can be accepted by CAs.*

Corollary 4.3. *Let M be a PTCM or a finite-crossing WTCM, and S be a set of configurations accepted by a PCA (respectively, finite-crossing WCA)* A_S. *Then pre*$^*(M,S)$ *and post*$^*(M,S)$ *can be accepted by PCAs (respectively, finite-crossing WCAs). Hence, the emptiness of these sets is decidable.*

Proof. Consider $W = pre^*(M,S)$. We can construct a PCA (respectively, finite-crossing WCA) accepting W as follows. From Theorem 4.1, we first construct a two-tape CA A_M accepting $Binary(M)$. Then we construct from A_M and A_S a two-tape PCA (respectively, finite-crossing WCA) accepting the set $Binary(M,S) = \{(\alpha,\beta) \mid (\alpha,\beta)$ in $Binary(M), \beta$ in $S\}$. Finally, we construct a PCA (respectively, a finite-crossing WCA) accepting W, which is simply the projection of $Binary(M,S)$ on the first coordinate. Hence, the emptiness of W is decidable. The case of $post^*(M,S)$ is handled similarly. □

5. Model checking and satisfiability checking

It is important to formulate what kinds of temporal properties are decidable for the machine models discussed in this paper. Given a machine (a PM, PCM, finite-crossing WCM, or its timed version) M and a configuration α, we use α_{c_i} to denote the value of counter c_i in α, $\alpha_{\#_a}$ to denote the number of appearances of symbol a in the stack/tape content in α, $\alpha_{\mathbf{q}}$ to denote the state in α. Let P be a Presburger formula on variables α_{c_i}, $\alpha_{\#_a}$, and $\alpha_{\mathbf{q}}$. Since the solutions of P can be accepted by a deterministic CA [16], it is obvious that the set of configurations satisfying P can be accepted by a deterministic CA. Particularly, if P is a Boolean combination of atomic formulas like $x>k$, $x=k$, where x is a variable (α_{c_i}, $\alpha_{\#_a}$, or $\alpha_{\mathbf{q}}$), and k is an integer, then P is called a *regular* formula. Obviously, the set of configurations satisfying a regular formula A can be accepted by an FA.

Now, we describe a (subset of a) Presburger linear temporal logic $\mathbf{L}$ as follows. This logic is inspired by the recent work in [5] on model checking a special form of counter automata without nested cycles. Formulas in $\mathbf{L}$ are defined as

$$f ::= P \mid A \mid P \wedge f \mid f \vee f \mid \circ f \mid A\ U\ f,$$

where P is a Presburger formula, A is a regular formula, $\circ$ and U stand for *next* and *until*, respectively. Formulas in $\mathbf{L}$ are interpreted on (finite) sequences p of configurations of M in a usual way. We use p^i to denote the sequence resulting from the deletion of the first i configurations from p. We use p_i to indicate the ith element in p. The satisfiability relation $\models$ is recursively defined as follows, for each sequence p and for each formula $f \in \mathbf{L}$ (written $p \models f$):

$p \models P$ if $p_1 \in P$,
$p \models A$ if $p_1 \in A$,
$p \models P \wedge f$ if $p \models P$ and $p \models f$,
$p \models f_1 \vee f_2$ if $p \models f_1$ or $p \models f_2$,
$p \models \circ f$ if $p^1 \models f$,
$p \models AUf$ if there exists j (which is not greater than the length of p) such that $p^j \models f$ and $\forall k < j(p^k \models A)$.

We use the convention that $\Diamond f$ (eventual) abbreviates ($trueU\ f$). The satisfiability-checking problem is to check whether there is an execution p of M satisfying $p \models f$, for a given M and $f \in \mathbf{L}$. The model-checking problem, which is the dual of the satisfiability-checking problem, is to check whether for all execution p of M satisfying $p \models f$, for a given M and $f \in \mathbf{L}$. The results of this paper show that:

Theorem 5.1. *The satisfiability-checking problem is decidable for* $\mathbf{L}$ *with respect to the following machine models*: *PM*, *PCM*, *finite-crossing WCM*, *and their timed versions.*

Proof (Sketch). Given $f \in \mathbf{L}$, we use $[f]$ to denote the set of p such that $p \models f$. For each of the machine models, we will show $[f]$ can be accepted by a CA. Therefore, the theorem follows by noticing that the satisfiability-checking problem is equivalent to testing the emptiness of the CA, which is decidable.

We will only look at PCM; all the other models can be handled similarly. The proof is based upon an induction on the structure of **L**. Obviously, $[P]$ and $[A]$ can be accepted by CAs; so can $[f_1 \vee f_2]$ if both $[f_1]$ and $[f_2]$ can. $[P \wedge f]$ can be accepted by a CA, since $[f]$ can be accepted by a CA and $[P]$ can be accepted by a deterministic CA. For the case of $[A\,U\,f]$, notice that the set $[A\,U\,f]$ is very similar to $Pre^*(M,[f])$—the only difference is that $[A\,U\,f]$ further requires that each intermediate configuration on the path leading to $[f]$ to be in A. This requirement can be easily fulfilled by slightly modify M, thanks to the fact that A is regular. Therefore, Corollary 2.10 still applies to show that $[A\,U\,f]$ can be accepted by a CA. The case for $[\circ f]$ is simpler, since we only look at one move. □

L is quite powerful. For instance, it can express a property like $\Diamond(P_1 \wedge \Diamond P_2)$. We should point out that without using the results in this paper, this property cannot be checked. For instance, the timed version of PM was studied in [9]. In that paper, it was shown that $[P_1 \wedge \Diamond P_2]$ can be accepted by a PCA—this is bad, since it is not possible to characterize $[\Diamond(P_1 \wedge \Diamond P_2)]$ from here (a machine accepting $[\Diamond(P_1 \wedge \Diamond P_2)]$ may need two stacks (i.e., Turing): one stack is for the PM, the other is for $[P_1 \wedge \Diamond P_2]$). But now, we have a stronger characterization for $[P_1 \wedge \Diamond P_2]$: it can be accepted by a CA. Therefore, the results in this paper give a CA characterization for $[\Diamond(P_1 \wedge \Diamond P_2)]$.

Since the model-checking problem is the dual of the satisfiability-checking problem, we conclude that

Theorem 5.2. *The model-checking problem is decidable for* $\neg$**L** (*taking negation of each formula in* **L**) *with respect to the following machine models*: *PM*, *PCM*, *finite-crossing WCM*, *and their timed versions.*

6. The boundedness problem

The boundedness problem for reachability sets is that of deciding, given a machine M in a class C_1 and a set of configurations S accepted by an acceptor in a class C_2, whether $R_\alpha(M) = \{\beta \mid \text{configuration } \beta \text{ is reachable from } \alpha\}$ is finite for every configuration α in S. We can show that the boundedness problem is decidable for C_1 the class of PTCMs or finite-crossing WTCMs and C_2 the class of PCAs or finite-crossing WCAs.

A special case of the boundedness problem is that of determining for a given α, whether $R_\alpha(M)$ is bounded. This problem has been studied in various places in the literature for other models like communicating finite-state machines. For example, [24] considers a model consisting of two finite-state machines with two queues (a queue to send a message from one machine to the other), but one of the two machines is allowed to send only a single type of message. The machines operate asynchronously but can test for queue emptiness at both the input and output channels. For these systems, [24] shows that the boundedness problem is decidable. (Here, a configuration consists of the states of the machines and the contents of the queues.)

Theorem 6.1. *The boundedness problem for PTCMs with respect to PCAs* (*or finite-crossing WCAs*) *is decidable.*

Proof. Let M be a PTCM and S be a set of configurations of M accepted by a PCA A_S. From Theorem 4.1, $Binary(M)$ can be accepted by a two-tape CA A_M. Using the PCA A_S accepting S, we can modify the two-tape CA A_M to a two-tape PCA B accepting the relation $Binary(M,S)=\{(\alpha,\beta)\,|\,(\alpha,\beta)\in Binary(M),\ \alpha\in S\}$. Now every tuple (α,β) in $Binary(M,S)$ is a pair of configurations, where a configuration (w,q,X) is represented as a string $wqd_1^{x_1}\cdots d_n^{x_n}$. Assume that there are r states $q_1,\ldots,q_r$. Let e and s be new symbols and h be a homomorphism that maps each symbol in the pushdown alphabet to e, q_i to s^i for $i=1,\ldots,r$ (thus a state is now represented in unary) and leaves the $d_1,\ldots,d_n$ unchanged. Let $h(Binary(M,S))$ be the relation obtained by applying h to the tuples in $Binary(M,S)$. Clearly, $h(Binary(M,S))$ can be accepted by a two-tape PCA C. Let # be a new symbol and define the language $L(M,S)=\{y_1\#y_2\,|\,(y_1,y_2)\in h(Binary(M,S))\}$. Thus, every string in $L(M,S)$ is just a concatenation (separated by a marker #) of a tuple in $h(Binary(M,S)$. Clearly, we can construct from the two-tape PCA C a PCA D accepting $L(M,S)$. (D reads the input tape and stores the unary values in y_1 and y_2 into distinct counters, and simulates C using these values with the help of auxiliary counters to simulate the counters of C.) Since $L(M,S)$ is a bounded language, it is semilinear (by Lemma 2.6) and, hence, a union of linear sets $T_1,\ldots,T_k$. We need only check if there is a period (vector) in the specification of some T_i that has at least one nonzero value in a position related to the second configuration (i.e., β) and zeros in all positions related to the first configuration (i.e., α). If so, $R_\alpha(M)$ is infinite for some α.

Similar constructions work when A_S is a finite-crossing WCA. In this case, the acceptor D accepting the language $L(M,S)$ would be a finite-crossing WCA, and since this language is bounded, it is semilinear [15]. □

Similarly, we can prove:

Theorem 6.2. *The boundedness problem for finite-crossing WTCMs with respect to PCAs (or finite-crossing WCAs) is decidable.*

Remark. One can define boundedness with respect to some specific component(s) of the reachable configurations. So, e.g., in a PTCM, we might be interested in deciding if the number of configurations reachable from the initial configuration is unbounded with respect to the pushdown stack content. This problem is also decidable, since in this case, we need only check that some S_i has a nonzero value in the position corresponding to the stack in the second configuration and zeros in all positions related to the first configuration.

7. Conclusion

In this paper, we showed that the binary (respectively, backward or forward) reachability sets of PCMs and finite-crossing WCMs as well as their "timed versions" (i.e., PTCMs and finite-crossing WTCMs) can be accepted by two-tape CAs (respectively, one-tape CAs), improving previous results which required a pushdown stack or a

finite-crossing tape in the acceptors. These results make it possible for us to formulate a subset of Presburger LTL for PCMs and finite-crossing WCMs. The logic includes formulas like $\diamond(P_1 \wedge \diamond P_2)$ which was not possible to be checked using previous techniques. We also showed that the boundedness problem for reachability sets is decidable.

Finally, we note that the pushdown stack (or finite-crossing tape) elimination technique can be applied to machines with real-valued clocks. In [7], a decidable characterization of the binary reachability of a PTCM with real-valued clocks was given. It was shown that for a PTCM M, $Binary(M)$ can be characterized in terms of a two-tape PCA M' and a parameter D_M. The characterization uses a pattern technique to separate a real-valued clock into an integral part and a fractional part, and D_M concerns the fractional parts of the clocks. The result can also be shown to hold when M is a finite-crossing WTCM, but now M' will be a two-tape WCA. Using the techniques in this paper, we can strengthen these results and show that in both cases, M' need only be a two-tape CA.

Acknowledgements

We wish to thank Nicholas Tran for helpful discussions concerning this work.

References

[1] P. Abdulla, B. Jonsson, Verifying programs with unreliable channels, Inform. Comput. 127 (2) (1996) 91–101.
[2] R. Alur, D. Dill, The theory of timed automata, Theoret. Comput. Sci. 126 (2) (1994) 183–236.
[3] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: application to model-checking, CONCUR'97, Lecture Notes in Computer Science, Vol. 1243, Springer, Berlin, pp. 135–150.
[4] G. Cece, A. Finkel, Programs with quasi-stable channels are effectively recognizable, CAV'97, Lecture Notes in Computer Science, Vol. 1254, Springer, Berlin, pp. 304–315.
[5] H. Comon, V. Cortier, Flatness is not a weakness, VCL'00, Lecture Notes in Computer Science, Vol. 1862, Springer, Berlin, pp. 262–276.
[6] H. Comon, Y. Jurski, Multiple counters automata, safety analysis and Presburger arithmetic, CAV'98, Lecture Notes in Computer Science, Vol. 1427, Springer, Berlin, pp. 268–279.
[7] Z. Dang, Binary reachability analysis of timed pushdown automata with dense clocks, CAV'01, Lecture Notes in Computer Science, Vol. 2102, Springer, Berlin, pp. 506–517.
[8] Z. Dang, O.H. Ibarra, Liveness verification of reversal-bounded counter machines with a free counter, FSTTCS'01, Lecture Notes in Computer Science, Vol. 2245, Springer, Berlin, pp. 132–243.
[9] Z. Dang, O.H. Ibarra, T. Bultan, R.A. Kemmerer, J. Su, Binary reachability analysis of discrete pushdown timed automata, CAV'00, Lecture Notes in Computer Science, Vol. 1855, Springer, Berlin, pp. 69–84.
[10] Z. Dang, P. San Pietro, R.A. Kemmerer, On Presburger liveness of discrete timed automata, STACS'01, Lecture Notes in Computer Science, Vol. 2010, Springer, Berlin, pp. 132–143.
[11] J. Esparza, D. Hansel, P. Rossmanith, S. Schwoon, Efficient algorithms for model checking pushdown systems, CAV'00, Lecture Notes in Computer Science, Vol. 1855, Springer, Berlin, pp. 232–247.

[12] A. Finkel, G. Sutre, Decidability of reachability problems for classes of two counter automata, STACS'00, Lecture Notes in Computer Science, Vol. 1770, Springer, Berlin, pp. 346–357.
[13] A. Finkel, B. Willems, P. Wolper, A direct symbolic approach to model checking pushdown systems, Electronic Notes in Theoretical Computer Science, Vol. 9, Elsevier, Amsterdam, 2000.
[14] S. Ginsburg, E. Spanier, Bounded algol-like languages, Trans. Amer. Math. Soc. 113 (1964) 333–368.
[15] T. Harju, O.H. Ibarra, J. Karhumaki, A. Salomaa, Decision questions concerning semilinearity, morphisms and commutation of languages, ICALP'01, Lecture Notes in Computer Science, Vol. 2076, Springer, Berlin, 2001, pp. 579–590.
[16] O.H. Ibarra, Reversal-bounded multicounter machines and their decision problems, J. ACM 25 (1978) 116–133.
[17] O.H. Ibarra, Reachability and safety in queue systems with counters and pushdown stack, Proc. Internat. Conf. on Implementation and Application of Automata, 2000, pp. 120–129.
[18] O.H. Ibarra, T. Bultan, J. Su, Reachability analysis for some models of infinite-state transition systems, Proc. 10th Internat. Conf. on Concurrency Theory, 2000, pp. 183–198.
[19] O.H. Ibarra, Z. Dang, P. San Pietro, Verification in loosely synchronous queue-connected discrete timed automata, Theoret. Comput. Sci. 290 (2003) 1713–1735.
[20] O.H. Ibarra, J. Su, Generalizing the discrete timed automaton, Proc. Internat. Conf. on Implementation and Application of Automata, 2000, pp. 206–215.
[21] O.H. Ibarra, J. Su, T. Bultan, Z. Dang, R.A. Kemmerer, Counter machines: decidable properties and applications to verification problems, MFCS'00, Lecture Notes in Computer Science, Vol. 1893, Springer, Berlin, pp. 426–435.
[22] K.L. McMillan, Symbolic model-checking—an approach to the state explosion problem, Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University, 1992.
[23] W. Peng, S. Purushothaman, Analysis of a class of communicating finite state machines, Acta Inform. 29 (6/7) (1992) 499–522.
[24] L.E. Rosier, H.-C. Yen, Boundedness, empty channel detection, and synchronization for communicating finite automata, Theoret. Comput. Sci. 44 (1986) 69–105.