Discrete Pushdown Timed Automata: Binary Reachability and Safety Analysis*

Zhe Dang[1] **, Oscar H. Ibarra[2], Tevfik Bultan[2], Richard A. Kemmerer[2] and Jianwen Su[2]

[1] School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164
zdang@eecs.wsu.edu

[2] Department of Computer Science
University of California
Santa Barbara, CA 93106
{ibarra,bultan,kemm,su}@cs.ucsb.edu

Abstract. We introduce discrete pushdown timed automata (DPTAs) that are timed automata with integer-valued clocks augmented with a pushdown stack. A configuration of a discrete pushdown timed automaton includes a control state, clock values and a stack word. Using a pure automata-theoretic approach, we show that the binary reachability (i.e., the set of all pairs of configurations (α, β), encoded as strings, such that α can reach β through 0 or more transitions) can be accepted by a reversal-bounded nondeterministic pushdown multicounter machine (NPCM). Since a discrete timed automaton (DTA) can be treated as a discrete pushdown timed automaton without the pushdown stack, we can show that the binary reachability of a discrete timed automaton can be accepted by a reversal-bounded nondeterministic multicounter machine. Thus, the binary reachability is Presburger. By using the known fact that the emptiness problem is decidable for reversal-bounded NPCMs, the results can be used to verify a number of safety properties that can not be expressed by timed temporal logics for discrete timed automata and CTL* for pushdown systems.

Key words: Model-checking, timed automata, pushdown systems, automated verification, binary reachability

* A preliminary version [15] of this paper appeared in the *Proceedings of the 12th International Conference on Computer-aided Verification* (CAV'00), Lecture Notes in Computer Science 1855, Springer-Verlag, 2000, pp. 69-84. The work by Zhe Dang and Richard A. Kemmerer have been supported in part by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Material Command, USAF, under agreement number F30602-97-1-0207. The work by Oscar H. Ibarra and Jianwen Su have been supported in part by NSF grants IRI-9700370 and IIS-9817432. The work by Tevfik Bultan has been supported in part by NSF grant CCR-9970976 and NSF CAREER award CCR-9984822.

** Corresponding author.

1 Introduction

After the introduction of efficient automated verification techniques such as symbolic model-checking [29], finite state machines have been widely used for modeling reactive systems. Due to the limited expressiveness, however, they are not suitable for specifying most infinite state systems. Thus, searching for models to represent more general transition systems and analyzing the decidability of their verification problems such as reachability or model-checking is an important research issue. In this direction, several models have been investigated such as timed automata [3], pushdown systems [8, 19, 17], various versions of counter machines [12, 18, 25], and various queue machines [7, 10, 30].

A timed automaton [3] can be considered as a finite automaton augmented with a finite number of clocks. The clocks can be reset to zero or progress at the same rate, and can be tested against clock constraints in the form of clock regions (i.e., comparisons of a clock or the difference of two clocks against an integer constant, e.g., $x - y > 5$, where x and y are clocks.). Timed automata are widely regarded as a standard model for real-time systems, because of their ability to express quantitative time requirements. In particular, by using the standard region technique, it has been shown that region reachability for timed automata is decidable [3]. This fundamental result and the technique help researchers, both theoretically and practically, in formulating various timed temporal logics [2, 4–6, 23, 26, 31, 32] and developing verification tools [21, 34, 27] (see [1, 35] for surveys).

In this paper, we consider integer-valued clocks. We call a timed automaton with integer-valued clocks a *discrete timed automaton* (DTA). A strictly more powerful system, called a discrete pushdown timed automaton (DPTA), can be obtained by augmenting discrete timed automaton with a pushdown stack. We give a characterization of binary reachability $\leadsto$, defined as the set of pairs of configurations (control state and clock values, plus the stack word if applicable) (α, β) such that α can reach β through 0 or more transitions. Binary reachability characterization is a fundamental step towards developing a model checking algorithm for DPTAs. From classical automata theory, it is known that the binary reachability of pushdown automata is context-free. For timed automata (with either real-valued clocks or integer-valued clocks), the region technique is not enough to give a characterization of the binary reachability. Recently, Comon et. al. [13] showed that the binary reachability of timed automata (with real-valued clocks) is expressible in the additive theory of reals. They show that a timed automaton with real-valued clocks can be flattened into one without nested cycles. Their technique also works for DTAs. However, it is not easy to deduce a characterization of the binary reachability of DPTAs by combining the above results. The reason is, as pointed out in their paper, this flattening destroys the structure of the original automaton. That is, the flattened timed automaton accepts different sequences of transitions, though the binary reachability is still the same. Thus, their approach cannot be used to show the binary reachability of the DPTAs proposed in this paper, since by flattening the sequence of stack operations cannot be maintained. A class of Pushdown Timed Systems (with continuous clocks) was discussed in [9]. However, that paper focuses on region reachability instead of binary reachability.

In this paper, we develop a new automata-theoretic technique to characterize the binary reachability of a DPTA. Our technique does not use the region technique [3] nor the flattening technique [13]. Instead, a nondeterministic pushdown multicounter machine (NPCM), which is a nondeterministic pushdown automaton augmented with counters, is used. Obviously, without restricting the counter behaviors, even the halting problem is undecidable, since machines with two counters already have an undecidable halting problem. A NPCM is reversal-bounded if the number of counter reversals (a counter changing mode between nondecreasing and nonincreasing and vice-versa) is bounded by some fixed number independent of the computation [24]. We show that the binary reachability of a DPTA can be accepted by a reversal-bounded NPCM. We also discuss the safety analysis problem. That is, given a property P and an initial condition I, which are two sets of configurations of a DPTA $\mathcal{A}$, determine whether, starting from a configuration in I, $\mathcal{A}$ can only reach configurations in P. Using the above characterization and the known fact that the emptiness problem for reversal-bounded NPCMs is decidable, we show that the safety analysis problem is decidable for DPTAs, as long as both the safety property and the initial condition are accepted by reversal-bounded nondeterministic multicounter machines (NCMs), i.e., reversal-bounded NPCMs without the pushdown stack. For instance, the property

> "whenever configuration α can reach configuration β, $\alpha_{x_1} + 2\beta_{x_2} - \beta_{x_1} - \alpha_{x_2} > \#_a(\alpha_{\mathbf{w}}) - \#_b(\beta_{\mathbf{w}})$ holds."

can be verified for a DPTA, where α_{x_1} is the integer value for clock x_1 in α, $\#_a(\alpha_{\mathbf{w}})$ is the number of symbols a in the stack word of α. Notice that the property defines a constraint of clocks not in the form of clock regions.

It is known that Presburger relations can be accepted by reversal-bounded NCMs [24]. Therefore, it is immediate that the safety analysis problem is decidable as long as both the safety property and the initial condition are Presburger formulas on clocks. A DTA can be treated as a DPTA without the pushdown stack. We can show that the binary reachability of a DTA can be accepted by a reversal-bounded NCM. That is, the binary reachability of a discrete timed automaton is Presburger. This result shadows the result in [13] that the binary reachability of a timed automaton with real-valued clocks is expressible in the additive theory of reals, although our approach is totally different.

This paper only considers discrete time. Doing this makes it possible to apply, as underlying theoretical tools, a number of automata-theoretic techniques and results. Besides the facts that discrete clocks are usually easier to handle than dense clocks also for practitioners, and that discrete timed automata are useful by themselves as a model of real-time systems [5], results on discrete timed automata may give insights into corresponding properties of dense timed automata [22]. Indeed, the results presented in this paper have recently been extended to timed pushdown automata with dense clocks [14]. In contrast to the safety problems discussed in this paper, liveness problems have recently been investigated in [16]. In that paper, the technique of eliminating clock tests from a discrete timed automaton given in this paper plays a key role.

The paper has four sections in addition to this section. Section 2 recalls a number of definitions and, in particular, defines discrete pushdown timed automata. Section 3 shows that the binary reachability of a DPTA can be accepted by a reversal-bounded NPCM. Section 4 investigates the safety analysis problems for DPTAs. Section 5 is a brief conclusion.

2 Discrete Pushdown Timed Automata

A timed automaton [3] is a finite state machine augmented with a number of real-valued clocks. All the clocks progress synchronously with rate 1, except a clock can be reset to 0 at some transition. In this paper, we consider integer-valued clocks. A *clock constraint* is a Boolean combination of *atomic clock constraints* in the following form:

$$x\#c, x-y\#c$$

where $\#$ denotes $\leq, \geq, <, >$, or $=$, c is an integer, x, y are integer-valued clocks. Let $\mathcal{L}_X$ be the set of all clock constraints on clocks X. Let $\mathbf{Z}$ be the set of integers with $\mathbf{Z}^+$ for nonnegative integers. Formally, a *discrete timed automaton* (DTA) $\mathcal{A}$ is a tuple

$$\langle S, X, E \rangle$$

where

- S is a finite set of *(control) states*,
- X is a finite set of *clocks* with values in $\mathbf{Z}^+$,
- $E \subseteq S \times 2^X \times \mathcal{L}_X \times S$ is a finite set of *edges* or *transitions*.

Each edge

$$\langle s, \lambda, l, s' \rangle$$

denotes a transition from state s to state s' with *enabling condition* $l \in \mathcal{L}_X$ and a set of clock resets $\lambda \subseteq X$. Note that λ may be empty. Also note that since a state may be connected to more than one state through multiple edges with the same enabling condition, $\mathcal{A}$ is, in general, nondeterministic.

The semantics is defined as follows. $\alpha \in S \times (\mathbf{Z}^+)^{|X|}$ is called a *configuration* with α_x being the value of clock x and α_q being the state under this configuration.

$$\alpha \rightarrow^{\langle s, \lambda, l, s' \rangle} \alpha'$$

denotes a *one-step transition* along an edge $\langle s, \lambda, l, s' \rangle$ in $\mathcal{A}$ satisfying

- The state s is set to a new location s', i.e., $\alpha_q = s, \alpha'_q = s'$.
- Each clock changes according to the edge given. If there are no clock resets on the edge, i.e., $\lambda = \emptyset$, then clocks progress by one time unit, i.e., for each $x \in X$, $\alpha'_x = \alpha_x + 1$. If $\lambda \neq \emptyset$, then for each $x \in \lambda$, $\alpha'_x = 0$ while for each $x \notin \lambda$, $\alpha'_x = \alpha_x$.
- The clock values in α satisfy the enabling condition l on the edge.

We simply write $\alpha \rightarrow \alpha'$ if α can reach α' by a one-step transition. A *path* is a sequence

$$\alpha_0 \cdots \alpha_n$$

for some $n \geq 1$ such that $\alpha_i \rightarrow \alpha_{i+1}$ for each $0 \leq i < n$. We write

$$\alpha \leadsto^{\mathcal{A}} \beta$$

if α reaches β through a path. Given a set P of configurations of $\mathcal{A}$, write the *preimage* $Pre^*(P)$ of P as the set of configurations that can reach a configuration in P, i.e.,

$$Pre^*(P) =_{def} \{\alpha : \text{ for some } \beta \in P, \alpha \leadsto^{\mathcal{A}} \beta\}.$$

The following figure shows an example of a discrete timed automaton with two clocks x_1 and x_2. The following sequence of configurations is a path: $\langle s_0, x_1 = 0, x_2 = 0\rangle, \langle s_1, x_1 = 0, x_2 = 0\rangle, \langle s_0, x_1 = 1, x_2 = 1\rangle, \langle s_1, x_1 = 1, x_2 = 0\rangle$.

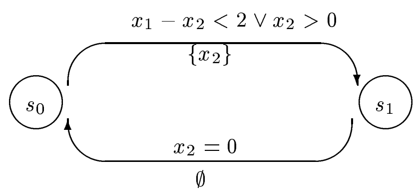

Figure 1. An example discrete timed automaton

The above defined $\mathcal{A}$ is a little different from the standard (discrete) timed automaton given in [3]. In that model, each state is assigned with a clock constraint called an *invariant* in which $\mathcal{A}$ can remain in the same control state with all the clocks synchronously progressing with rate 1 as long as the invariant is satisfied. It is easy to see that, when integer-valued clocks are considered, $\mathcal{A}$'s remaining in a state can be replaced by a self-looping transition with the invariant as the enabling condition and without clock resets. Each execution of such a transition causes all the clocks to progress by one time unit. Another difference is that in a standard timed automaton a state transition takes no time, even when the transition has no clock resets. In order to translate a standard timed automaton to our definition, we introduce a dummy clock. Thus, for each state transition t in a standard timed automaton the translated transition t' is exactly the same except the dummy clock is reset in t'. Thus, doing this will ensure that all clock values remain the same when t has no clock resets. Thus, standard timed automata can be easily transformed into the ones defined above. Since we focus on the clock behaviors of a DTA, instead of the ω-language accepted by it, input symbols (or event labels) are not considered in our definition. (The input to a timed automaton is always one-way. Thus, input symbols can always be built into states.)

Discrete timed automata can be further extended by allowing a pushdown stack. A *discrete pushdown timed automaton* (DPTA) $\mathcal{A}$ is a tuple

$$\langle \Gamma, S, X, E\rangle$$

where Γ is the stack alphabet, and S, X, E are the same as in the definition of a discrete timed automaton except that each edge includes a stack operation.

That is, each edge e is in the form of

$$\langle s, \lambda, (\eta, \eta'), l, s' \rangle$$

where $s, s' \in S$, $\lambda \subseteq X$ is the set of clock resets and $l \in \mathcal{L}_X$ is the enabling condition. The stack operation is characterized by a pair (η, η') with $\eta \in \Gamma$ and $\eta' \in \Gamma^*$. That is, replacing the top symbol of the stack η by a word η'. A configuration $\alpha \in (\mathbf{Z}+)^{|X|} \times S \times \Gamma^*$ with $\alpha_w \in \Gamma^*$ indicating the stack content. $\alpha \rightsquigarrow^{\mathcal{A}} \beta$ can be similarly defined assuming that the stack contents in α and β are consistent with the sequence of stack operations along the path.

This paper focuses on the characterization of binary reachability $\rightsquigarrow^{\mathcal{A}}$ for both discrete timed automata and discrete pushdown timed automata. Before we proceed to show the results, some further definitions are needed.

A *nondeterministic multicounter machine* (NCM) is a nondeterministic machine with a finite set of *(control) states* $Q = \{1, 2, \cdots, |Q|\}$, and a finite number of counters $x_1, \cdots, x_k$ with integer counter values. Each counter can add 1, subtract 1, or stay unchanged. Those counter assignments are called *standard assignments.* M can also test whether a counter is equal to, greater than, or less than an integer constant. Those tests are called *standard tests.*

A NCM can be augmented with a pushdown stack. A *nondeterministic pushdown multicounter machine* (NPCM) M is a nondeterministic machine with a finite set of *(control) states* $Q = \{1, 2, \cdots, |Q|\}$, a pushdown stack with stack alphabet Γ, and a finite number of counters $x_1, \cdots, x_k$ with integer counter values. Both assignments and tests in M are standard. In addition, M can pop the top symbol from the stack or push a word in Γ^* on the top of the stack. It is well-known that counter machines with two counters have an undecidable halting problem [28], and obviously the undecidability holds for machines augmented with a pushdown stack. Thus, we have to restrict the behaviors of the counters. One such restriction is to limit the number of reversals that a counter can make. A counter is *n-reversal-bounded* if it changes mode between nondecreasing and nonincreasing at most n times. For instance, the following sequence of a counter values: $0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 3, 2, 1, 1, 1, 1, \cdots$ demonstrates only one counter reversal. A counter is *reversal-bounded* if it is *n-reversal-bounded* for some n. We note that a reversal-bounded M (i.e., each counter in M is reversal-bounded) does not necessarily limit the number of moves or the number of reachable configurations to be finite.

Let $(j, v_1, \cdots, v_k, w)$ denote the *configuration* of M when it is in state $j \in Q$, counter x_i has value $v_i \in \mathbf{Z}$ for $i = 1, 2, \cdots, k$, and the string in the pushdown stack is $w \in \Gamma^*$ with the rightmost symbol being the top of the stack. Each integer counter value v_i can be represented by a unary string 0^{v_i} (1^{v_i}) when v_i positive (negative). Thus, a configuration $(j, v_1, \cdots, v_k, w)$ can be represented as a string by concatenating the unary representations of each j,$v_1, \cdots, v_k$ as well as the string w with a separator $\# \notin \Gamma$. For instance, $(1, 2, -2, w)$ can be represented by $0^1\#0^2\#1^2\#w$. Similarly, an integer tuple $(v_1, \cdots, v_k)$ can also be represented by a string. Thus, in this way, a set of configurations (or a set of integer tuples) can be treated as a set of strings, i.e., a language. It is noticed that a configuration α of a discrete (pushdown) timed automaton $\mathcal{A}$ can be similarly encoded as a string $[\alpha]$.

Note that the above defined M does not have an input tape; in this case it is used as a system specification rather than a language recognizer, in which we are more interested in the behaviors that M generates. When a NPCM (or NCM) M is used as a language recognizer, we attach a separate one-way read-only input tape to the machine and assign a state in Q as the final state. M *accepts* an input iff it can reach the final state. When M is reversal-bounded, the emptiness problem, i.e., whether M accepts some input, is known to be decidable:

Theorem 1. *The emptiness problem for reversal-bounded nondeterministic pushdown multicounter machines with a one-way input tape is decidable [24].*

It has been shown in [20] that the emptiness problem for reversal-bounded NCMs with one-way input is decidable in n^{ckr} time for some constant c, where n is the size of the machine, k is the number of counters, and r is the reversal-bound on each counter. We believe that a similar bound could be obtained for the case of reversal-bounded NPCMs.

Actually, Theorem 1 can be strengthened for the case of NCMs:

Theorem 2. *A set of n-tuples of integers is definable by a Presburger formula iff it can be accepted by a reversal-bounded nondeterministic multicounter machine [24].*

A language is *bounded* if there exist finitely many words $w_1, \cdots, w_n$ such that each element can be represented as $w_1^* \cdots w_n^*$. A reversal-bounded nondeterministic multicounter machine can be made deterministic on bounded languages.

Theorem 3. *If a bounded language L is accepted by a reversal-bounded nondeterministic multicounter machine, then L can also be accepted by a reversal-bounded deterministic multicounter machine [24].*

For a NPCM M, we can define the *preimage* $Pre^*(P)$ of a set of configurations P similarly to be the set of all predecessors of configurations in P, i.e.,

$$Pre^*(P) = \{t \mid t \text{ can reach some configuration } t' \text{ in } P \text{ in 0 or more moves}\}.$$

We can show that $Pre^*(P)$ can be accepted by a reversal-bounded NPCM assuming that M is reversal-bounded and P is accepted by a reversal-bounded NCM.

Theorem 4. *Let M be a reversal-bounded nondeterministic pushdown multicounter machine. Suppose a set of configurations P is accepted by a reversal-bounded nondeterministic multicounter machine. Then $Pre^*(P)$ with respect to M can be accepted by a reversal-bounded nondeterministic pushdown multicounter machine.*

Proof. Let M be a reversal-bounded NPCM. Assume M has k counters. Let P, a set of configurations of M, be accepted by a reversal-bounded NCM M' with a one-way input tape. We construct a reversal-bounded NPCM M'', with a one-way input tape, accepting $Pre^*(P)$ with respect to M.

Given a configuration $(j, v_1, \cdots, v_k, w)$ on its first $k+1$ counters and the stack, M'' simulates M starting in configuration $(j, v_1, \cdots, v_k, w)$. At some point, M'' guesses that it has reached a configuration $(j', v'_1, \cdots, v'_k, w')$ in P. M'' then checks that the string representation of $(j', v'_1, \cdots, v'_k, w')$ is accepted by M' by simulating M'. M'' accepts if M' accepts. Clearly M'' accepts $Pre^*(P)$ with respect to M. In the simulation of M' on $(j', v'_1, \cdots, v'_k, w')$, M'' uses additional k counters to store $v'_1, \cdots, v'_k$ and store j', since j' is chosen from a finite set of control states, in its finite control. w' is stored in the pushdown stack of M''. M'' thus uses the additional counters and the pushdown stack which contains w' to simulate the action of M' on its input. Since the input tape of M' is one-way and the additional counters of M'' are reversal-bounded, when M' reads w' from the input tape, M'' simulates this by reading w' from the right by "popping" the stack of M''. There is a slight problem in that M' will be working on the reverse of w', not w'. However, it is known that if a language L is accepted by a reversal-bounded nondeterministic multicounter machine, then the reverse L^r of L can also be accepted by a reversal-bounded nondeterministic multicounter machine [24]. Hence, we can use the machine accepting the P^r in the computation of M'' to decide if $(j', v'_1, \cdots, v'_k, w')$ is in P. $\square$

3 A Binary Reachability Characterization for DPTAs

Let $\mathcal{A}$ be a discrete pushdown timed automaton with clocks $x_1, \cdots, x_k$. The binary reachability $\leadsto^{\mathcal{A}}$ can be treated as a language

$$\{[\alpha]\#[\beta]^r : \alpha \leadsto^{\mathcal{A}} \beta\}$$

where $[\alpha]$ is the string encoding of configuration α, $[\beta]^r$ is the reverse string encoding of configuration β. [1] The two encodings are separated by a delimiter "#". The main result claims that the binary reachability $\leadsto^{\mathcal{A}}$ can be accepted by a reversal-bounded NPCM (using standard tests and assignments). $\mathcal{A}$ itself can be regarded as a NPCM, when we refer to a clock as a counter. However, tests in $\mathcal{A}$ as clock constraints are not standard tests. Furthermore, $\mathcal{A}$ is not reversal-bounded since clocks can be reset for an unbounded number of times.

The proof of the main result proceeds as follows. We first show that the tests in $\mathcal{A}$ can be replaced by finite table look-ups. That is, $\leadsto^{\mathcal{A}}$ can be accepted by a reversal-bounded NPCM using standard tests and nonstandard assignments. Then, we show that the nonstandard assignments can be made standard and the counters are reversal-bounded.

Recall that, instead of standard tests, $\mathcal{A}$ has nonstandard tests that check an enabling condition by comparing the difference of two counters against an integer constant. Also the assignments include only $x_i := x_i + 1$ (progress) and $x_i := 0$ (reset) for $1 \le i \le k$ in $\mathcal{A}$, which are not standard assignments. The following theorem says the nonstandard tests can be made standard.

Theorem 5. *The binary reachability $\leadsto^{\mathcal{A}}$ of $\mathcal{A}$ can be accepted by a reversal-bounded NPCM using standard tests and nonstandard assignments that are of the form $x_i := x_i + 1$ and $x_i := 0$ with $1 \le i \le k$.*

Proof. We construct the reversal-bounded NPCM M as required. Given a pair of string encodings of configurations α and β (separated by a delimiter "#" not in the stack alphabet, also recall that the encoding of β has the stack word in β reversed.) of $\mathcal{A}$ on M's one-way input tape, M first copies α into its k counters $x_1, \cdots, x_k$ and the stack. Thus, M's input head stops at the beginning of β. M starts simulating $\mathcal{A}$ as follows with the stack operations in $\mathcal{A}$ being exactly simulated on its own stack.

[1] The reason that we use the reverse encoding of β will become apparent in the Proof of Theorem 5.

Tests in $\mathcal{A}$ are Boolean combinations of $x_i\#c$, $x_i - x_j\#c$ for $1 \le i, j \le k$. Using only standard tests, M cannot directly compare the difference of two counter values against an integer c by storing $x_i - x_j$ in another counter, since each time this "storing" is done it will cause at least a counter reversal, and we don't have a bound on the number of such tests. Here we demonstrate a technique to eliminate these tests, based on the definition of a finite table lookup $a_{ij}\#c$ and $b_i\#c$ to replace tests $x_i - x_j\#c$ and $x_i\#c$.

Assume m is one plus the maximal absolute value of all the integer constants that appear in the tests in $\mathcal{A}$. Denote the finite set $[m] =_{def} \{-m, \cdots, 0, \cdots, m\}$. M uses its finite control to build a finite table. For each $1 \le i, j \le k$, there are entries a_{ij} and b_i. Each entry can be regarded as a finite state control variable with states in $[m]$. Intuitively, a_{ij} is used to record the difference between two clock values of x_i and x_j, and b_i is used to record the clock value of x_i. During the computation of $\mathcal{A}$, when the difference $x_i - x_j$ (or the value x_i) goes beyond m or below $-m$, a_{ij} (or b_i) keeps the values m or $-m$, respectively.

The procedure for updating the entries is given below, in which "$\oplus 1$" means adding one if the result does not exceed m, else it keeps the same value. "$\ominus 1$" means subtracting one if the result is not less than $-m$, else it keeps the same value. We modify $\mathcal{A}$ as follows. Let e be an edge of $\mathcal{A}$. If the set of clock resets on e is $\lambda = \emptyset$, then the entries are updated by adding the following instructions to e, for each $1 \le i \le k$:

- $a_{ij} := a_{ij}$ for each $1 \le j \le k$. Recall that all the clocks progress after this edge, thus the difference is unchanged.
- $b_i := b_i \oplus 1$. That is, clocks progress by one time unit.

If the set of clock resets is $\lambda \ne \emptyset$, the entries are updated by adding the following instructions to e, for each $1 \le i, j \le k$:

- $a_{ij} := 0$ if $x_i \in \lambda$ and $x_j \in \lambda$. In this case, both the clocks x_i and x_j reset to 0.
- $a_{ij} := -b_j$ if $x_i \in \lambda$ and $x_j \notin \lambda$. In this case, x_i resets but x_j does not. So the difference should be $-x_j$.
- $a_{ij} := b_i$ if $x_i \notin \lambda$ and $x_j \in \lambda$.
- $a_{ij} := a_{ij}$ if $x_i \notin \lambda$ and $x_j \notin \lambda$.

followed by adding the following instructions:

- $b_i := b_i$ if $x_i \notin \lambda$. In this case, x_i does not reset.
- $b_i := 0$ if $x_i \in \lambda$. In this case, x_i resets.

The initial values of a_{ij} and b_i can be constructed directly from the values α_{x_i} of clocks x_i in configuration α on the input tape, for each $1 \le i, j \le k$:

- $a_{ij} := \alpha_{x_i} - \alpha_{x_j}$ if $|\alpha_{x_i} - \alpha_{x_j}| \le m$,
- $a_{ij} := m$ if $\alpha_{x_i} - \alpha_{x_j} > m$,
- $a_{ij} := -m$ if $\alpha_{x_i} - \alpha_{x_j} < -m$,

and, noticing that clocks are nonnegative,

- $b_i := \alpha_{x_i}$ if $\alpha_{x_i} \le m$,
- $b_i := m$ if $\alpha_{x_i} > m$.

M then simulates $\mathcal{A}$ exactly except using $a_{ij}\#c$ for test $x_i - x_j\#c$ and using $b_i\#c$ for test $x_i\#c$. We claim that doing this by M is valid,

Claim. By adding the above entry updating instructions to $\mathcal{A}$, after $\mathcal{A}$ executes any transition, the following conditions hold for all $1 \le i, j \le k$ and for each integer c, $-m < c < m$:
(1). $x_i - x_j\#c$ iff $a_{ij}\#c$,
(2). $x_i\#c$ iff $b_i\#c$.

Proof of the Claim. We prove it by induction. Obviously, the Claim holds for the initial values of all clocks x_i (in configuration α) and the corresponding entries a_{ij} and b_i, by the choice of m. Suppose that $\mathcal{A}$ is currently at configuration γ and the Claim holds. Thus, we assume the induction hypothesis as follows. For all $1 \le i, j \le k$ and for each integer $c \in [m-1]$,

$$\gamma_{x_i} - \gamma_{x_j}\#c \text{ iff } a_{ij}\#c$$

and

$$\gamma_{x_i}\#c \text{ iff } b_i\#c$$

hold. Therefore, γ satisfies an enabling condition $x_i - x_j\#c$ (resp. $x_i\#c$) in $\mathcal{A}$ iff the entry a_{ij} (resp. b_i) satisfies the same enabling condition by replacing $x_i - x_j$ (resp. x_i) with a_{ij} (resp. b_i), noticing that m is chosen such that it is greater than the absolute value of any constant in all the enabling conditions in $\mathcal{A}$. Assume γ satisfies the enabling condition on an edge e and $\mathcal{A}$ will execute e next. Thus, M, using the entries a_{ij} and b_i to test the enabling condition, will also execute the same edge. We use γ' to denote the configuration after executing the edge, and use a'_{ij} and b'_i to denote the entry values after executing the edge. We need to show, for all $1 \le i, j \le k$ and for each integer $c \in [m-1]$,

(*) $\gamma'_{x_i} - \gamma'_{x_j} \# c$ iff $a'_{ij} \# c$

and

(**) $\gamma'_{x_i} \# c$ iff $b'_i \# c$

hold. There are two cases to consider according to the form of the assignment on edge e.

Case 1. The assignment is a clock progress $x_i := x_i + 1$ for all $1 \le i \le k$. After firing the edge, $\gamma'_{x_i} = \gamma_{x_i} + 1$ for each $1 \le i \le k$. On the other hand, according to the entry updating instructions, a'_{ij} and b'_i are updated for each $1 \le i, j \le k$ as follows:

- $a'_{ij} = a_{ij}$,
- $b'_i = b_i \oplus 1$.

Therefore,

- Claim (*) holds. This is because $\gamma'_{x_i} - \gamma'_{x_j} \# c$ iff $\gamma_{x_i} - \gamma_{x_j} \# c$ iff (induction hypothesis) $a_{ij} \# c$ iff $a'_{ij} \# c$.
- Claim (**) holds. This can be shown by considering the following two subcases, noticing that b_i is nonnegative:
  - the subcase for $b_i < m$. Notice that $b'_i = b_i + 1$, and from the induction hypothesis, $\gamma_{x_i} = b_i$. We have $\gamma'_{x_i} \# c$ iff $\gamma_{x_i} + 1 \# c$ iff $b_i + 1 \# c$ iff $b'_i \# c$.
  - the subcase for $b_i = m$. Notice that $b'_i = b_i \oplus 1 = b_i$, and from the induction hypothesis, $\gamma_{x_i} \ge m$. Hence, $\gamma'_{x_i} > m$. By the choice of c, we have $\gamma'_{x_i} \# c$ iff $m \# c$ iff $b'_i \# c$.

Case 2. The assignment is a clock reset in the form of $x_i := 0$ for $x_i \in \lambda$, where $\lambda \neq \emptyset$ is the set of clock resets on edge e. Notice that, in this case, if $x_i \in \lambda$, then $\gamma'_{x_i} = 0$, else $\gamma'_{x_i} = \gamma_{x_i}$. In order to show Claim (*) holds, there are four subcases to consider according to the entry updating instructions for a_{ij}: for all $1 \le i, j \le k$,

- If $x_i, x_j \in \lambda$, then $\gamma'_{x_i} = \gamma'_{x_j} = 0$, $a'_{ij} = 0$ and therefore $\gamma'_{x_i} - \gamma'_{x_j} = a'_{ij} = 0$. Thus, Claim (*) holds.
- If $x_i \in \lambda$ and $x_j \notin \lambda$, then, $\gamma'_{x_i} = 0$, $\gamma'_{x_j} = \gamma_{x_j}$, $a'_{ij} = -b_i$. Thus, for each $c \in [m-1]$, $\gamma'_{x_i} - \gamma'_{x_j} \# c$ iff $-\gamma_{x_j} \# c$ iff (induction hypothesis) $-b_i \# c$ iff $a'_{ij} \# c$. Thus, Claim (*) holds.
- If $x_i \notin \lambda$ and $x_j \in \lambda$, similar as above.
- If $x_i \notin \lambda$ and $x_j \notin \lambda$, then, $\gamma'_{x_i} = \gamma_{x_i}$, $\gamma'_{x_j} = \gamma_{x_j}$, and $a'_{ij} = a_{ij}$. Thus, Claim (*) holds.

Now we prove Claim (**) holds. There are two subcases to consider according to the entry updating instructions for b_i: for all $1 \le i \le k$,

– If $x_i \notin \lambda$, then $b'_i = b_i$, and $\gamma'_{x_i} = \gamma_{x_i}$. Claim (**) holds.
– If $x_i \in \lambda$, then $b'_i = 0$, and $\gamma'_{x_i} = 0$. Claim (**) also holds.

This ends the proof of the Claim. Thus, it is valid for M to use $a_{ij}\#c$ to do each test $x_i - x_j\#c$ and to use $b_i\#c$ to do each test $x_i\#c$. At some point, M guesses that it has reached the configuration β by comparing the counter values and the stack content with β through reading the rest of the input tape. M accepts iff such a comparison succeeds. Clearly M accepts $\rightsquigarrow^{\mathcal{A}}$. There is a slight problem when M compares its own stack content with the one β_w on the one-way input tape in β by popping the stack. The reason is that popping the stack contents reads the reverse of the stack content. However, recall that the encoding of the stack word β_w on the input tape is reversed. Thus, such a comparison can be proceeded. $\square$

Assignments in M constructed in the above proof, in the form of, $x_i := x_i + 1$ and $x_i := 0$ with $1 \leq i \leq k$, are still not standard. We now show that these assignments can be made standard, while the machine is still reversal-bounded. Let M' be a NPCM that is exactly the same as M. M' simulates M's computation from the configuration α. Initially, each $x_i := \alpha_{x_i}$ and M' calculates the initial values for all entries a_{ij} and b_i from configuration α, as indicated in the above proof. The calculations can be implemented under the help of a number of auxiliary reversal-bounded counters. However, for all $1 \leq i \leq k$, each time that M executes an assignment $x_i := x_i + 1$ or $x_i := 0$, M' **does nothing** to x_i. The stack operations in M are faithfully simulated by M' on its own stack. For each $1 \leq i \leq k$, at some point, either initially or at a moment $x_i := 0$ is being executed by M, M' guesses (only once for each i) that x_i will not reset afterwards. When the guess for i happens at $x_i := 0$, M' decrements x_i to 0. After such a guess for i, an execution of $x_i := x_i + 1$ will also cause x_i incremented by 1. However, a later execution of $x_i := 0$ in M will cause M' to abort abnormally (without accepting the input). At some point after **all** $1 \leq i \leq k$ have been guessed, M' guesses that it has reached the configuration β. Then, M' compares its current configuration with the one on the rest of the input tape β (recall that the stack word in β is reversed on the input tape.). M' accepts iff such a comparison succeeds. Clearly, M' uses only standard assignments and, more importantly, each x_i in M' is reversal-bounded. Thus, M' is a reversal-bounded NPCM accepting $\rightsquigarrow^{\mathcal{A}}$. Therefore,

Theorem 6. *The binary reachability of a discrete pushdown timed automaton can be accepted by a reversal-bounded nondeterministic pushdown multicounter machine using standard tests and assignments.*

A discrete timed automaton is a special case of a discrete pushdown timed automaton without the pushdown stack. The above proofs still work for discrete timed automata without considering stack operations. That is,

Theorem 7. *The binary reachability of a discrete timed automaton can be accepted by a reversal-bounded multicounter machine using standard tests and assignments.*

Combining Theorem 7 and Theorem 2, it is immediate that the binary reachability of a discrete timed automaton is Presburger over clocks. This result shadows the result in [13] that the binary reachability of a timed automaton with real-valued clocks is expressible in the additive theory of reals. However, our proof is totally different from the flattening technique in [13].

Theorem 8. *The binary reachability of a discrete timed automaton is Presburger. (also shown in [13])*

4 Safety Analysis Problems

The importance of the characterization of $\leadsto^{\mathcal{A}}$ for a discrete pushdown timed automaton $\mathcal{A}$ is that the emptiness of reversal-bounded NPCMs is decidable from Theorem 1. In this section, we will formulate a number of properties that can be verified for discrete pushdown timed automata.

We first need some notation. We use $\alpha, \beta \cdots$ to denote variables ranging over configurations. We use $\mathbf{q}, \mathbf{x}, \mathbf{w}$ to denote variables ranging over control states, clock values and stack words respectively. Note that α_{x_i}, α_q and α_w are still used to denote the value of clock x_i, the control state and the stack word of α. We use a count variable $\#_a(\mathbf{w})$ to denote the number of occurrences of a character $a \in \Gamma$ in a stack word variable $\mathbf{w}$. A NPCM-term t is defined as follows:[2]

$$t \ ::= \ n \mid \mathbf{q} \mid \mathbf{x} \mid \#_a(\alpha_w) \mid \alpha_{x_i} \mid \alpha_q \mid t - t \mid t + t$$

[2] Control states can be interpreted over a bounded range of integers. In this way, an arithmetic operation on control states is well-defined.

where n is an integer and $a \in \Gamma$. A NPCM-formula f is defined as follows:

$$f \ ::= \ t > 0 \mid t \bmod n = 0 \mid \neg f \mid f \vee f$$

where $n \neq 0$ is an integer[3]. Thus, f is a Presburger formula over control state variables, clock value variables and count variables. Let F be a formula in the following format:

$$\bigvee (f_i \wedge \ \alpha^i \rightsquigarrow \beta^i)$$

where f_i, α^i and β^i are a number of NPCM-formulas and configuration variables. Write $\exists F$ to be a closed formula such that each free variable in F is existentially quantified. Then, the property $\exists F$ can be verified. [4]

Theorem 9. *The truth value of $\exists F$ with respect to a discrete pushdown timed automaton $\mathcal{A}$ for any NPCM-formula F is decidable.*

Proof. Consider each disjunctive subformula $f_i \wedge \ \alpha^i \rightsquigarrow \beta^i$ in F. Since f_i is Presburger, (the domain of) f_i can be accepted by a NCM M_{f_i} from Theorem 2, and further from Theorem 3, since the domain of f_i can be encoded as a set of integer tuples (thus, bounded), M_{f_i} can be made deterministic. From Theorem 7, we can construct a reversal bounded NPCM accepting (the domain of) $\alpha^i \rightsquigarrow \beta^i$. It is obvious that $f_i \wedge \ \alpha^i \rightsquigarrow \beta^i$ can be accepted by a reversal-bounded NPCM M_i by "intersecting" the two machines. Now, F is a union of M_{f_i}. Since reversal-bounded NPCMs are closed under union [24], we can construct a reversal-bounded M to accept F. Since $\exists F = false$ is equivalent to testing the emptiness of M, the theorem follows from Theorem 1. $\square$

Although $\exists F$ cannot be used to specify liveness properties, it can be used to specify many interesting safety properties. For instance, the following property:

"for any configurations α and β with $\alpha \rightsquigarrow^{\mathcal{A}} \beta$, clock x_2 in β is the sum of clocks x_1 and x_2 in α, and symbol a appears in β twice as many times as symbol b does in α."

---

[3] We use $(i, j, \mathbf{w})$ to denote the i-th character of $\mathbf{w}$ is the j-th symbol in Γ. Then, Theorem 9 is still correct when we include (i, j, α_w) as an atomic formula in a NPCM-formula with $i, j \in \mathbf{Z}^+$.

[4] If stack words in α^i and β^i are bounded, i.e., in the form of $w_1^* \cdots w_k^*$ with $w_1, \cdots, w_k$ fixed, then F can be further extended to allow disjunctions *and* conjunctions over $(f_i \wedge \ \alpha^i \rightsquigarrow \beta^i)$ formulas and the following theorem still holds. The reason is that reversal-bounded NPCMs are closed under conjunction when languages are bounded [24].

This property can be expressed as, $\forall\alpha\forall\beta(\alpha \leadsto^{\mathcal{A}} \beta \rightarrow (\beta_{x_2} = \alpha_{x_1} + \alpha_{x_2} \wedge \#_a(\beta_w) = 2\#_b(\alpha_w)))$. The negation of this property is equivalent to $\exists F$ for some F. Thus, it can be verified. We also need to point out that

– Even without clocks, $\#_a(\beta_w) = 2\#_b(\alpha_w)$ indicate a nonregular set of stack word pairs. Thus, this property cannot be verified by the model checking procedures for pushdown systems [8, 19, 33],
– Even without the pushdown stack, $\beta_{x_2} = \alpha_{x_1} + \alpha_{x_2}$ is not a clock region [3]. Thus, the classical region techniques (include [9] for Pushdown Timed Systems) can not verify this property. This is also pointed out in [13].

Note that in a NPCM-formula, the use of a stack word is limited to count the occurrences of a symbol, e.g., $\#_a(\alpha_w)$. In fact we can have the following more general use. Given P and I, two sets of configurations of a discrete pushdown timed automaton $\mathcal{A}$. If, starting from a configuration in I, $\mathcal{A}$ can only reach configurations in P, then P is a *safety property* with respect to the initial condition I. The *safety analysis problem* is whether P is a safety property with respect to the initial condition I, given P and I. The following theorem asserts that the safety analysis problem is decidable for discrete pushdown timed automata, when both P and I are bounded languages and are accepted by reversal-bounded NCMs.

Theorem 10. *The safety analysis problem is decidable for discrete pushdown timed automata where both the safety property and the initial condition are bounded languages and accepted by reversal-bounded nondeterministic multicounter machines.*

Proof. Let $\mathcal{A}$ be a discrete pushdown timed automaton. Let P and I be accepted by reversal-bounded NCMs M_P and M_I, respectively. Note that P is a safety property with respect to the initial condition I iff $I \cap Pre^*(\neg P) = \emptyset$. That is, if P is a safety property, then, starting from a configuration, $\mathcal{A}$ can not reach a configuration that is in the complement $\neg P$ of P. From Theorem 3, since P and I are bounded, both M_P and M_I can be made deterministic. Thus, $\neg P$ can also be accepted by a reversal-bounded NCM. Therefore, from Theorem 6 and Theorem 4, we can construct a reversal-bounded NPCM M_{Pre^*} accepting $Pre^*(\neg P)$. It is obvious that $I \cap Pre^*(\neg P)$ can also be accepted by a reversal-bounded NPCM M' by "intersecting" M_I and M_{Pre^*}. The theorem follows by noticing that $I \cap Pre^*(\neg P) = \emptyset$, i.e., testing the emptiness of M' is decidable from Theorem 1. $\square$

Thus, from the above theorem, the following property can be verified:

"starting from a configuration α with the stack word $a^n b^{2n}$ for some n, $\mathcal{A}$ can only reach a configuration β satisfying: the clock x_1 in β is the sum of clock x_2 and x_3 in β, and the stack word is $a^n b^{n+2m}$ for some n and m."
The reason is that $a^n b^{2n}$ and $a^n b^{n+2m}$ can be encoded as Presburger tuples (thus bounded). Therefore they can be accepted by reversal-bounded NCMs.

Now let's look at $\mathcal{A}$ without a pushdown stack, i.e., a discrete timed automaton. Obviously, the above two theorems still hold for such $\mathcal{A}$. However, since $\rightsquigarrow^{\mathcal{A}}$ now is Presburger from Theorem 7, we can do more. A NCM-term t is defined as follows:

$$t \ ::= \ n \mid \mathbf{q} \mid \mathbf{x} \mid \alpha_{x_i} \mid \alpha_q \mid t - t \mid t + t$$

where n is an integer and $a \in \Gamma$. A NCM-formula f is defined as follows:

$$f \ ::= \ t > 0 \mid \neg f \mid f \vee f \mid \alpha \rightsquigarrow^{\mathcal{A}} \beta \mid \forall \alpha(f) \mid \forall \mathbf{x}(f) \mid \forall \mathbf{q}(f).$$

Thus, f is a Presburger formula over control state variables, clock value variables and configuration variables. Thus, if f is closed (i.e., without free variables), then the truth value of f is decidable since f is Presburger. Thus, a property formulated as a closed NCM-formula can be verified. Even clocks are real values, this is still true from [13]. The following is an example property which can be verified:

"for all configuration α there exists a configuration β such that $\alpha \rightsquigarrow^{\mathcal{A}} \beta$ and the clock x_1 in β is the sum of clocks x_1 and x_2 in α."
This property can be written in the following closed NCM-formula,

$$\forall \alpha \exists \beta (\alpha \rightsquigarrow^{\mathcal{A}} \beta \ \wedge \ \beta_{x_1} = \alpha_{x_1} + \alpha_{x_2}).$$

NCM-formulas can be extended by considering event labels as follows. Consider a discrete timed automata $\mathcal{A}$. Recall that an edge in $\mathcal{A}$ does not have a label. Now we assume that, just as a standard timed automaton, each edge in $\mathcal{A}$ is labeled by a letter in a finite alphabet Γ. Denote $R(\alpha, \beta, \mathbf{n}_\Gamma)$ to be a predicate meaning α can reach β through a path that for each $a \in \Gamma$, the number of occurrence of label a in the path is equal to $\mathbf{n}_a$ with $\mathbf{n}_\Gamma$ being an array of $\mathbf{n}_a$ for $a \in \Gamma$. By introducing a new counter for each $a \in \Gamma$ and increasing the counter whenever M executes a transition labeled by a, we can construct a reversal-bounded NCM M as in the proof of Theorem 7 (and the actual proof is in Theorem 6.). From Theorem 2, we have,

Theorem 11. *R is Presburger.*

Thus, we can add atomic terms $\mathbf{n}_a$ for $a \in \Gamma$ and an atomic formula $R(\alpha, \beta, \mathbf{n}_\Gamma)$ to the above definition of NCM-formulas. Then such closed NCM-formulas can be verified for discrete timed automata with labels. The following is an example property:

"For any configuration α there exists a configuration β such that if the clock x_1 in β is the sum of clocks x_1 and x_2 in α, then α can reach β through a path with the number of transitions labeled by a being twice the number of transitions labeled by b."

This property can be written in a closed NCM-formula,

$$\forall \alpha \exists \beta (\beta_{x_1} = \alpha_{x_1} + \alpha_{x_2} \rightarrow \exists \mathbf{n}_a \exists \mathbf{n}_b (R(\alpha, \beta, \mathbf{n}_a, \mathbf{n}_b) \wedge \mathbf{n}_a = 2\mathbf{n}_b)).$$

Thus, it can be verified.

5 Conclusion

We consider discrete pushdown timed automata that are timed automata with integer-valued clocks augmented with a pushdown stack. Using a pure automata-theoretic approach, we show that the binary reachability can be accepted by a reversal-bounded nondeterministic pushdown multicounter machine. The proof reveals that, by replacing enabling conditions with a finite table, the control part (testing clock constraints) and the clock behaviors (clock progresses and resets) can be separated for a discrete (pushdown) timed automaton, while maintaining the structure of the automaton. By using the known fact that the emptiness problem for reversal-bounded nondeterministic pushdown multicounter machines is decidable, we show a number of properties that can be verified.

Binary reachability characterization is a fundamental step towards a more general model-checking procedure for discrete pushdown timed automata. It is immediate to see that the region reachability in [3] can be similarly formulated by using Theorem 10, as long as stack words are regular. Thus, we can use the idea in [8] as well as in [19] to demonstrate a subset of μ-calculus that has a decidable decision procedure for a class of timed pushdown processes. We plan to investigate this issue in future work. In the future we would also like to investigate the complexity of the verification procedures we have developed in this paper. The techniques in this paper will be used in the implementation of a symbolic model checkers for real-time specifications written in the specification language ASTRAL [11] and a discrete real-time language with procedure calls.

References

1. R. Alur. *"Timed automata"*, In *CAV'99*, **LNCS** 1633, pp. 8-22
2. R. Alur, C. Courcoibetis, and D. Dill. *"Model-checking in dense real time," Information and Computation*, **104** (1993): 2-34
3. R. Alur and D. Dill. *"A Theory of Timed Automata," Theoretical Computer Science*, **126** (1994): 183-236
4. R. Alur, T. Feder, and T. A. Henzinger. *"The benefits of relaxing punctuality," J. ACM*, **43** (1996): 116-146
5. R. Alur, T. A. Henzinger. *"Real-time logics: complexity and expressiveness," Information and Computation*, **104** (1993): 35-77
6. R. Alur, T. A. Henzinger. *"A really temporal logic," J. ACM*, **41** (1994): 181-204
7. P. Abdulla and B. Jonsson. *"Verifying programs with unreliable channels," Information and Computation*, **127** (1996): 91-101
8. A. Bouajjani, J. Esparza, and O. Maler. *"Reachability Analysis of Pushdown Automata: Application to Model-Checking,"*, In *CONCUR'97*, **LNCS** 1243, pp. 135-150
9. A. Bouajjani, R. Echahed, and R. Robbana. *"On the Automatic Verification of Systems with Continuous Variables and Unbounded Discrete Data Structures,"*, In *Hybrid System II*, **LNCS** 999, 1995, pp. 64-85
10. G. Cece and A. Finkel. *"Programs with Quasi-Stable Channels are Effectively Recognizable,"* In *CAV'97*, **LNCS** 1254, pp. 304-315
11. A. Coen-Porisini, C. Ghezzi and R. Kemmerer. *"Specification of real-time systems using ASTRAL," IEEE Transactions on Software Engineering*, **23** (1997): 572-598
12. H. Comon and Y. Jurski. *"Multiple counters automata, safety analysis and Presburger arithmetic,"* In *CAV'98*, **LNCS** 1427, pp. 268-279.
13. H. Comon and Y. Jurski. *"Timed Automata and the Theory of Real Numbers,"* In *CONCUR'99*, **LNCS** 1664, pp. 242-257
14. Z. Dang. *"Binary Reachability Analysis of Pushdown Timed Automata with Dense Clocks,"* In *CAV'01*, to appear
15. Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer and J. Su. *"Binary Reachability Analysis of Discrete Pushdown Timed Automata,"* IN *CAV'00*, **LNCS** 1855, pp. 69-84
16. Z. Dang, P. San Pietro, and R. A. Kemmerer. *"On Presburger Liveness of Discrete Timed Automata,"* In *STACS'01*, **LNCS** 2010, pp. 132-143
17. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. *"Efficient Algorithms for Model Checking Pushdown Systems,"* In *CAV'00*, **LNCS** 1855, pp. 232-247
18. A. Finkel and G. Sutre. *"Decidability of Reachability Problems for Classes of Two Counter Automata,"* In *STACS'00*, **LNCS** 1770, pp. 346-357
19. A. Finkel, B. Willems and P. Wolper. *"A direct symbolic approach to model checking pushdown systems,"* In *INFINITY'97*.
20. E. Gurari and O. Ibarra. *"The Complexity of Decision Problems for Finite-Turn Multicounter Machines," J. Computer and System Sciences*, **22** (1981): 220-229
21. T. A. Henzinger and Pei-Hsin Ho. *"HyTech: the Cornell hybrid technology tool,"* In *Hybrid Systems II*, **LNCS** 999, pp. 265-294
22. T. A. Henzinger, Z. Manna, and A. Pnueli. *"What good are digital clocks?,"* In *ICALP'92*, **LNCS** 623, pp. 545-558
23. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. *"Symbolic Model Checking for Real-time Systems," Information and Computation*, **111** (1994): 193-244
24. O. H. Ibarra. *"Reversal-bounded multicounter machines and their decision problems," J. ACM*, **25** (1978): 116-133

25. O. H. Ibarra, J. Su, T. Bultan, Z. Dang, and R. A. Kemmerer. *"Counter Machines: Decidable Properties and Applications to Verification Problems,"*, In *MFCS'00*, **LNCS** 1893, pp. 426-435
26. F. Laroussinie, K. G. Larsen, and C. Weise. *"From timed automata to logic - and back,"* In *MFCS'95*, **LNCS** 969, pp. 529-539
27. K. G. Larsen, P. Pattersson, and W. Yi. *"UPPAAL in a nutshell," International Journal on Software Tools for Technology Transfer*, **1** (1997): 134-152
28. M. Minsky. *"Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines," Ann. of Math.*, **74** (1961): 437-455
29. K. L. McMillan. *"Symbolic model checking - an approach to the state explosion problem,"* PhD thesis, Carnegie Mellon University, 1992.
30. W. Peng and S. Purushothaman. *"Analysis of a Class of Communicating Finite State Machines," Acta Informatica*, 29 (1992): 499-522
31. J. Raskin and P. Schobben. *"State clock logic: a decidable real-time logic,"* In *HART'97*, **LNCS** 1201, pp. 33-47
32. T. Wilke. *"Specifying timed state sequences in powerful decidable logics and timed automata,"* **LNCS** 863, pp. 694-715, 1994
33. I. Walukiewicz. *"Pushdown processes: games and model checking,"* In *CAV'96*, **LNCS** 1102, pp. 62-74
34. S. Yovine. *"A verification tool for real-time systems," International Journal on Software Tools for Technology Transfer*, **1** (1997): 123-133
35. S. Yovine. *"Model checking timed automata,"* In *Embedded Systems'98*, **LNCS** 1494, pp. 114-152