# Past Pushdown Timed Automata

**(Extended Abstract)**

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

[1] School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164
[2] Department of Computer Science
University of California
Santa Barbara, CA 93106

**Abstract.** We consider past pushdown timed automata that are discrete pushdown timed automata [15] with past-formulas as enabling conditions. Using past formulas allows a past pushdown timed automaton to access the past values of the finite state variables in the automaton. We prove that the reachability (i.e., the set of reachable configurations from an initial configuration) of a past pushdown timed automaton can be accepted by a nondeterministic reversal-bounded multicounter machine augmented with a pushdown stack (i.e., a reversal-bounded NPCM). Using the known fact that the emptiness problem for reversal-bounded NPCMs is decidable, we show that model-checking past pushdown timed automata against Presburger safety properties on discrete clocks and stack word counts is decidable. An example ASTRAL specification is presented to demonstrate the usefulness of the results.

## 1 Introduction

As far as model-checking is concerned, the most successful model of infinite state systems that has been investigated is probably timed automata [2]. A timed automaton can be considered as a finite automaton augmented with a number of clocks. Enabling conditions in a timed automaton are in the form of (clock) *regions*: a clock or the difference of two clocks is tested against an integer constant, e.g., $x - y < 8$. The region technique [2] has been used to analyze region reachability, to develop a number of temporal logics [1, 3–5, 20, 24, 26, 29] and for model-checking tools [19, 23, 30].

The region technique is useful, but obviously not enough. For instance, it is not possible, using the region technique, to verify whether clock values satisfying a non-region property $x_1 - x_2 > x_3 - x_4$ are reachable for a timed automaton. The verification calls for a decidable characterization for the binary reachability (all the configuration pairs such that one can reach the other) of a timed automaton. The characterizations have recently been established in [9] for timed automata and in [15, 12] for timed automata augmented with a pushdown stack.

In this paper, we consider a class of discrete timed systems, called past pushdown timed automata. In a past pushdown timed automaton, the enabling condition of a transition can access some finite state variable's past values. For instance, consider discrete clocks $x_1, x_2$ and $now$ (a clock that never resets, indicating the current time). Suppose that $a$ and $b$ are two Boolean variables. An enabling condition could be in the form of a past formula:

$\forall 0 \leq y_1 \leq now \exists 0 \leq y_2 \leq now((x_1 - y_1 < 5 \wedge a(x_1) = b(y_2)) \rightarrow (y_2 < x_2 + 4))$, in which $a(x_1)$ and $b(y_2)$ are (past) values of $a$ and $b$ at times $x_1$ and $y_2$, respectively. Thus, past pushdown timed automata are history dependent; that is, the current state depends upon the entire history of the transitions leading to the state. The main result of this paper shows that the reachability of past pushdown timed automata can be accepted by reversal-bounded multicounter machines augmented with a pushdown stack (i.e., reversal-bounded NPCMs). Since the emptiness problem for reversal-bounded NPCMs is decidable [21], we can show that checking past pushdown timed automata against Presburger safety properties on discrete clocks and stack word counts is decidable. This result is not covered by region-based results for model-checking timed pushdown systems [7], nor by model-checking pushdown systems [6, 16].

Besides their own theoretical interest, history-dependent timed systems have practical applications. It is a well known principle that breaking a system into several loosely independent functional modules greatly eases both verification and design work. The ultimate goal of modularization is to partition a large system, both conceptually and functionally, into several small modules and to verify each small module instead of verifying the large system as a whole. That is, verify the correctness of each module without looking at the behaviors of the other modules. This idea is adopted in a real-time specification language ASTRAL [8], in which a module (called a process) is provided with an interface section, which is a first-order formula that abstracts its environment. It is not unusual for these formulas to include complex timing requirements that reflect the patterns of variable changes. Thus, in this way, even a history independent system can be specified as a number of history dependent modules (see [8, 10, 13] for a number of interesting real-time systems specified in ASTRAL). Thus, the results in this paper have immediate applications in implementing an ASTRAL symbolic model checker.

Past formulas are not new. In fact, they can be expressed in TPTL [5], which is obtained by including clock constraints (in the form of clock regions) and freeze quantifiers in the Linear Temporal Logic (LTL) [25]. But, in this paper, we put a past formula into the enabling condition of a transition in a generalized timed system. This makes it possible to model a real-time machine that is history-dependent. Past formulas can be expressed through S1S (see Thomas [28] and Straubing [27] for details), which can be characterized by Buchi (finite) automata. This fact does not imply (at least not in an obvious way) that timed automata augmented with these past formulas can be simulated by finite automata. In this extended abstract, the complete ASTRAL specification, as well as some substantial proofs, in particular for Theorems 3, 4 and 6 are omitted. For a complete exposition see [11].

## 2 Preliminaries

A *nondeterministic multicounter machine (NCM)* is a nondeterministic machine with a finite set of *(control) states* $Q$, and a finite number of counters with integer counter values. Each counter can add 1, subtract 1, or stay unchanged. These counter assignments are called *standard assignments*. The machine can also test whether a counter is equal to, greater than, or less than an integer constant, and these tests are called *standard tests*.

An 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* $Q$, a pushdown stack with stack alphabet $\Pi$, and a finite number of counters 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 $\Pi^*$ on the top of the stack. It is well-known that counter machines with two counters have undecidable halting problem, and obviously the undecidability holds for machines augmented with a pushdown stack. Thus, we have to restrict the behaviors of the counters. A counter is *$n$-reversal-bounded* if it changes mode between nondecreasing and nonincreasing at most $n$ times. For instance, the following sequence of 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 $n$ that is independent of the computations. We note that a reversal-bounded $M$ (i.e., each counter in $M$ is reversal-bounded) does not necessarily limit the number of moves to be finite.

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. When an NPCM $M$ 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 (i.e., accepting) state. $M$ *accepts* an input iff it can reach an accepting 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 [21].*

An NCM can be regarded as an NPCM without the pushdown stack. Thus, the above theorem also holds for reversal-bounded NCMs. We can also give a nice characterization. If $S$ is a set of $n$-tuples of integers, let $L(S)$ be the set of strings representing the tuples in $S$ (each component, i.e., an integer, in a tuple is encoded as a unary string). It was shown in [21] that if $L(S)$ is accepted by a reversal-bounded nondeterministic multicounter machine, then $S$ is semilinear. The converse is obvious. Since $S$ is semilinear if and only if it is definable by a Presburger formula, we have:

**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 [21].*

## 3 Past Formulas

Let $A$ be a finite set of finite state variables, i.e., their domains are a bounded range of integers. We use $a, b \cdots$ to denote them. Without loss of generality, we assume they are Boolean variables. All clocks are discrete. Let *now* be the clock representing the current time. Let $X$ be a finite set of integer-valued variables. *Past-formulas* are defined as

$f ::= a(y) \mid y < n \mid y < z + n \mid \Box_x f[0, now] \mid f \vee f \mid \neg f,$

where $a \in A$, $y$ and $z$ are in $X \cup \{now\}$, $x \in X$, and $n$ is an integer. Intuitively, $a(y)$ is the variable $a$'s value at time $y$, i.e., $\texttt{Past}(a, y)$ in ASTRAL [8]. *Quantification*

$\Box_x f[0, now]$, with $x \neq now$ (i.e., *now* can not be quantified), means, for all $x$ from 0 to *now*, $f$ holds. An appearance of $x$ in $\Box_x f[0, now]$ is called *bounded*. We assume any $x$ is bounded by at most one $\Box_x$. $x$ is *free* in $f$ if $x$ is not bounded in $f$. $f$ is *closed* if *now* is the only free variable. Past-formulas are interpreted on a history of Boolean variables. A history consists of a sequence of boolean values for each variable $a \in A$. The length of every sequence is $\mathbf{n}$+1, where $\mathbf{n}$ is the value of *now*. Formally, a *history* $H$ is a pair $\langle \{||a||\}_{a \in A}, \mathbf{n} \rangle$, where $\mathbf{n} \in \mathbf{Z}^+$ is a nonnegative integer representing the value of *now*, and for each $a \in A$, the mapping $||a|| : 0..\mathbf{n} \to \{0, 1\}$ gives the Boolean value of $a$ at each time point from 0 to $\mathbf{n}$. Let $\mathcal{B} : X \to \mathbf{Z}^+$ be a *valuation* for variables in $X$. Thus $\mathcal{B}(x) \in \mathbf{Z}^+$ denotes the value of $x \in X$ under this valuation $\mathcal{B}$. We use $\mathcal{B}(n/x)$ to denote replacing $x$'s value in the valuation with a non-negative integer $n$. Given a history $H$ and a valuation $\mathcal{B}$, the interpretations of past-formulas are as follows, for each $y, z \in X \cup \{now\}$ and each $x \in X$,

$$||x||_{H,\mathcal{B}} = \mathcal{B}(x), \qquad ||now||_{H,\mathcal{B}} = \mathbf{n},$$
$$||a(y)||_{H,\mathcal{B}} \iff ||a||(||y||_{H,\mathcal{B}}), \qquad ||y < n||_{H,\mathcal{B}} \iff ||y||_{H,\mathcal{B}} < n,$$
$$||\neg f||_{H,\mathcal{B}} \iff \text{not } ||f||_{H,\mathcal{B}}, \qquad ||f_1 \vee f_2||_{H,\mathcal{B}} \iff ||f_1||_{H,\mathcal{B}} \text{ or } ||f_2||_{H,\mathcal{B}}.$$
$$||y < z + n||_{H,\mathcal{B}} \iff ||y||_{H,\mathcal{B}} < ||z||_{H,\mathcal{B}} + n,$$
$$||\Box_x f[0, now]||_{H,\mathcal{B}} \iff \text{for all } k \text{ with } 0 \leq k \leq \mathbf{n},\ ||f||_{H,\mathcal{B}(k/x)},$$

When $f$ is a closed formula, we write $||f||_H$ instead of, for all $\mathcal{B}$, $||f||_{H,\mathcal{B}}$. We use $\Diamond_x$ to denote $\neg\Box_x\neg$.

A history $H$ can be regarded as a sequence of snapshots $S_0, \cdots, S_{\mathbf{n}}$ such that each snapshot gives a value for each $a \in A$. When *now* progresses from $\mathbf{n}$ to $\mathbf{n} + 1$ history $H$ is updated to a new history $H'$ by adding a new snapshot $S_{\mathbf{n}+1}$ to history $H$. This newly added snapshot represents the new values of $a \in A$ at the new current time $\mathbf{n} + 1$. Is there any way to calculate the truth value of the formula under $H'$ by using the new snapshot $S_{\mathbf{n}+1}$ and the truth value of the formula under $H$? If this can be done, the truth value of the formula can be updated along with the history's update from $\mathbf{n}$ to $\mathbf{n} + 1$, without looking back at the old snapshots $S_0, \cdots, S_{\mathbf{n}}$. The rest of this section shows that this can be done.

A *Boolean function* is a mapping $\mathbf{Z}^+ \to \{0, 1\}$. A Boolean predicate is a mapping $\{0, 1\}^m \to \{0, 1\}$ for some $m$. We use $v_1, \cdots, v_{|A|}$ to denote the Boolean functions representing the truth value of each $a \in A$ at each time point. Obviously, $v_1, \cdots, v_{|A|}$ can be obtained by extending $\mathbf{n}$ to $\infty$ in a history. When $v_1, \cdots, v_{|A|}$ are given, a closed past formula can be regarded as a Boolean function: the truth value of the formula at time $t$ is the interpretation of the formula under the history $v_i(0), \cdots, v_i(t)$ for each $1 \leq i \leq |A|$. Given closed past formulas $f, g_1, \cdots, g_k$ (for some $k$), we use $u, u_1, \cdots, u_k$ to denote the Boolean functions for them, respectively.

**Theorem 3.** *For any closed past formula $f$, there are closed past formulas $g_1, \cdots, g_k$, and Boolean predicates $O, O_1, \cdots, O_k$ such that, for any given Boolean functions $v_1$, $\cdots$, $v_{|A|}$, the Boolean functions $u, u_1, \cdots, u_k$ (defined above) satisfy: for all t in $\mathbf{Z}^+$,*

$$u(t + 1) = O(v_1(t + 1), \cdots, v_{|A|}(t + 1), u_1(t), \cdots, u_k(t))$$

*and for each i, $1 \leq i \leq k$,*

$$u_i(t + 1) = O_i(v_1(t + 1), \cdots, v_{|A|}(t + 1), u_1(t), \cdots, u_k(t)).$$

Therefore, $u(t + 1)$ (the truth value of formula $f$ at time $t + 1$), as well as each $u_i(t + 1)$, can be recursively calculated by using the values of $v_1, \cdots, v_{|A|}$ at $t + 1$, and values of $u_1, \cdots, u_k$ at $t$. As we mentioned before, past formulas can be expressed in TPTL [5]. A tableau technique is proposed in [5] to show that validity checking of TPTL is decidable. A modification of the technique can be used to prove Theorem 3.

To conclude this section, we point out that once the functions $v_1, \cdots, v_{|A|}$ representing each $a(now)$ for $a \in A$ are known, each closed past formula can be recursively calculated as in Theorem 3. In the next two sections, we will build past formulas into transition systems.

## 4 Past Machines: A Simpler Case

A *past-machine* $M$ is a tuple $\langle S, A, E, now\rangle$, where $S$ is a finite set of *(control) states*. $A$ is a finite set of Boolean variables, and *now* is the only clock in $M$. *now* is used to indicate the current time. $E$ 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* $l$ and *assignments* $\lambda$ to the Boolean variables in $A$. $l$ is a closed past-formula. $\lambda : A \to \{0, 1\}$ denotes the new value $\lambda(a)$ of each variable $a \in A$ after an execution of the transition. Execution of a transition causes the clock *now* to progress by 1 time unit. A *configuration* $\alpha$ of $M$ is a pair $\langle \alpha_{\mathbf{q}}, \alpha_H\rangle$ where $\alpha_{\mathbf{q}}$ is a state and $\alpha_H = \langle \{||a||^{\alpha_H}\}_{a\in A}, \mathbf{n}^{\alpha_H}\rangle$ is a history. $\alpha \to^{\langle s,\lambda,l,s'\rangle} \beta$ denotes a one-step transition along edge $\langle s, \lambda, l, s'\rangle$ in $M$ satisfying:

- The state $s$ is set to a new location, i.e., $\alpha_{\mathbf{q}} = s, \beta_{\mathbf{q}} = s'$.
- The enabling condition is satisfied, i.e., $||l||_{\alpha_H}$ holds under the history $\alpha_H$.
- The clock *now* progresses by one time unit, i.e., $\mathbf{n}^{\beta_H} = \mathbf{n}^{\alpha_H} + 1$.
- The history $\alpha_H$ is extended to $\beta_H$ by adding the resulting values (given by the assignment $\lambda$) of the Boolean variables after the transition. That is, for all $a \in A$, for all $t$, $0 \le t \le \mathbf{n}^{\alpha_H}$, history $\beta_H$ is consistent with history $\alpha_H$; i.e., $||a||^{\beta_H}(t) = ||a||^{\alpha_H}(t)$. In addition, $\beta_H$ extends $\alpha_H$; i.e., for each $a \in A$, $||a||^{\beta_H}(\mathbf{n}^{\beta_H}) = \lambda(a)$.

We write $\alpha \to \beta$ if $\alpha$ can reach $\beta$ by a one-step transition. A *path* $\alpha_0 \cdots \alpha_k$ satisfies $\alpha_i \to \alpha_{i+1}$ for each $i$. We write $\alpha \leadsto \beta$ if $\alpha$ reaches $\beta$ through a path. $\alpha$ is *initial* if the current time is 0, i.e., $\mathbf{n}^{\alpha_H} = 0$. There are only finitely many initial configurations. Denote $R = \{\langle \alpha, \beta\rangle : \alpha \text{ is initial}, \alpha \leadsto \beta\}$.

Let $M$ be a past machine as specified above. $M$, starting from an initial configuration $\alpha$ (i.e., with $now = 0$) can be simulated by a counter machine with reversal-bounded counters. In the following, we will show the construction. Each enabling condition $l$ on an edge $e \in E$ of $M$ is a closed past-formula. From Theorem 3, each $l$ can be associated with a number of Boolean functions $O_l, O_{1,l}, \cdots, O_{k,l}$, and a number of Boolean variables $u^l_1, \cdots, u^l_k$ (updated while *now* progresses). $l$ itself can be considered as a Boolean variable $u^l$. We use a primed form to indicate the previous value of a variable – here, a variable changes with time progressing. Thus, from Theorem 3,[1] these variables are updated as, $u^l := O_l(A, {u^l_1}', \cdots, {u^l_k}')$ and for all $u^l_i$ $u^l_i := O_{i,l}(A, {u^l_1}', \cdots, {u^l_k}')$. Thus, $M$ can be simulated by a counter machine $M'$ as follows. $M'$ is exactly the same as $M$ except that each test of an enabling condition of

[1] We simply use $A$ to indicate the current values of $a \in A$ with the assumption that the "current time" can be figured out from the context.

$l$ in $M$ is replaced by a test of a Boolean variable $u^l$ in $M'$. Furthermore, whenever $M$ executes a transition, $M'$ does the following (sequentially):

- increase the counter *now* by 1,
- change the values of Boolean variables $a \in A$ according to the assignment given in the transition in $M$,
- for each enabling condition of $l$ in $M$, $M'$ has Boolean variables $u^l, u_1^l, \cdots, u_k^l$. $M'$ updates (as given above) $u_1^l, \cdots, u_k^l$ and $u^l$ for **each** $l$. Of course, during the process, the new values of $a \in A$ will be used, which were already updated above.

The initial values of Boolean variables $a \in A$, $u^l$ and $u_j^l$ can be assigned using the initial values of $a \in A$ in $\alpha$. $M'$ contains only one counter *now*, which never reverses. Essentially $M'$ is a finite state machine augmented by one reversal-bounded counter. It is obvious that $M'$ faithfully simulates $M$. A configuration $\beta$ can be encoded as a string composed of the control state $\beta_q$, the current time $\mathbf{n}^{\beta_H}$ (as a unary string), and the history concatenated by the values of $a \in A$ at time $0 \leq t \leq \mathbf{n}^{\beta_H}$. All components are separated by a delimiter "#" as follows:
$1^{\beta_q}\#\pi_0\#\cdots\#\pi_{\mathbf{n}^{\beta_H}}\#1^{\mathbf{n}^{\beta_H}}$
where $\pi_t$ is a binary string with length $|A|$ indicating the values of all $a \in A$ at $t$. Thus, a set of configurations can be considered as a language. Denote $R_\alpha$ to be the set of configurations $\beta$ with $\alpha \rightsquigarrow \beta$. Then,

**Theorem 4.** *$R_\alpha$ is accepted by a reversal-bounded nondeterministic multicounter machine.*

Since there are only finitely many initial configurations, $R = \{\langle \alpha, \beta \rangle : \alpha \text{ initial }, \beta \in R_\alpha\}$ can be accepted by a reversal-bounded nondeterministic multicounter machine.

**Theorem 5.** *$R$ is accepted by a reversal-bounded nondeterministic multicounter machine.*

The reason that past machines are simple is that they contain only closed past formulas. Thus, we have to extend past machines by allowing a number of clock variables in the system, as shown in the next section.

## 5 Past Pushdown Timed Automata: A More Complex Case

Past machines can be extended by allowing extra free variables, in addition to *now*, in an enabling condition. We use $Z = \{z_1, \cdots, z_k\} \subseteq X$ to denote the variables other than *now*. A *past pushdown timed automaton* $M$ is a tuple $\langle S, A, Z, \Pi, E, now \rangle$ where $S$ and $A$ are the same as those for a past machine. $Z$ is a finite set of clocks with $now \notin Z$ and $\Pi$ is a finite stack alphabet. Each edge, from state $s$ to state $s'$, in $E$ is denoted by $\langle s, \delta, \lambda, (\eta, \eta'), l, s' \rangle$. $l$ and $\lambda$ have the same meaning as in a past machine, though the enabling condition $l$ may contain, in addition to *now*, free (clock) variables in $Z$. $\delta \subseteq Z$

denotes a set of clock jumps. [2] $\delta$ may be empty. The stack operation is characterized by a pair $(\eta, \eta')$ with $\eta \in \Pi$ and $\eta' \in \Pi^*$. That is, replacing the top symbol of the stack $\eta$ by a word $\eta'$. A configuration $\alpha$ of $M$ is a tuple $\langle \alpha_{\mathbf{q}}, \alpha_H, \alpha_Z, \alpha_w \rangle$ where $\alpha_{\mathbf{q}}$ is a state, $\alpha_H$ is a history as defined in a past machine, and $\alpha_Z \in (\mathbf{Z}^+)^{|Z|}$ is a valuation of clock variables $Z$. We use $\alpha_z$ to denote the value of $z \in Z$ under this configuration. $\alpha_w \in \Pi^*$ indicates the stack content.

$\alpha \rightarrow^{\langle s,\delta,\lambda,(\eta,\eta'),l,s' \rangle} \beta$ denotes a one-step transition along edge $\langle s, \delta, \lambda, (\eta, \eta'), l, s' \rangle$ in $M$ satisfying:

- The state $s$ is set to a new location, i.e., $\alpha_{\mathbf{q}} = s, \beta_{\mathbf{q}} = s'$.
- The enabling condition is satisfied, i.e., $||l||_{\alpha_H, \mathcal{B}(\alpha_Z/Z)}$ holds for any $\mathcal{B}$. That is, $l$ is evaluated under the history $\alpha_H$ and replacing each free clock variable $z \in Z$ by the value $\alpha_z$ in the configuration $\alpha$.
- Each clock changes according to the edge given.
  - If $\delta = \emptyset$, i.e., there are no clock jumps on the edge, then the now-clock progresses by one time unit. That is, $\mathbf{n}^{\beta_H} = \mathbf{n}^{\alpha_H} + 1$. All the other clocks do not change; i.e., for each $z \in Z$, $\beta_z = \alpha_z$.
  - If $\delta \neq \emptyset$, then all the clocks in $\delta$ jump to $now$, and the other clocks do not change. That is, for each $z \in \delta$, $\beta_z = \mathbf{n}^{\alpha_H}$. In addition, for each $z \notin \delta$ , $\beta_z = \alpha_z$, and the clock $now$ does not progress, i.e., $\mathbf{n}^{\beta_H} = \mathbf{n}^{\alpha_H}$.
- The history is updated similarly as for past machines. That is,
  - If $\delta = \emptyset$, then $now$ progresses, for all $a \in A$, for all $t$, $0 \leq t \leq \mathbf{n}^{\alpha_H}$, $||a||^{\beta_H}(t) = ||a||^{\alpha_H}(t)$, and $||a||^{\beta_H}(\mathbf{n}^{\beta_H}) = \lambda(a)$.
  - If $\delta \neq \emptyset$, then $now$ does not progress, for all $a \in A$, for all $t$, $0 \leq t \leq \mathbf{n}^{\alpha_H} - 1$, $||a||^{\beta_H}(t) = ||a||^{\alpha_H}(t)$, and $||a||^{\beta_H}(\mathbf{n}^{\beta_H}) = \lambda(a)$. Thus, even though the now-clock does not progress, the current values of variables $a \in A$ may change according to the assignment $\lambda$.
- According to the stack operation $(\eta, \eta')$, the stack word $\alpha_w$ is updated to $\beta_w$.

$\alpha$ is initial if the stack word is empty and all clocks including $now$ are 0. Similar to the case for past machines, we define $\alpha \leadsto \beta$ and $R$. Again, $R$ can be considered as a language by encoding a configuration into a string. The main result in this section is that $R$ can be accepted by a reversal-bounded NPCM. The major difference between a past machine and a past pushdown timed automaton is that the enabling condition on an edge in the past pushdown timed automaton is not necessarily a closed past formula. The proof, which can be found in [11], shows that an enabling condition $l$ with free variables in $Z$ can be made closed.

**Theorem 6.** *The set $R$ of a past pushdown timed automaton can be accepted by a reversal-bounded NPCM.*

The importance of the automata-theoretic characterization of $R$ is that the Presburger safety properties over clocks and stack word counts are decidable. We use $\beta$ to

[2] Here we use clock jumps (i.e., $x := now$) instead of clock resets ($x := 0$). The reason is that, in this way, the start time of a transition can be directly modeled as a clock. Obviously, a transform from $x$ to $now - x$ will give a "traditional" clock with resets.

denote variables ranging over configurations. We use $\mathbf{q}, \mathbf{z}, \mathbf{w}$ to denote variables ranging over control states, clock values and stack words, respectively. Note that $\beta_{z_i}$, $\beta_q$ and $\beta_w$ are still used to denote the value of clock $z_i$, the control state and the stack word of $\beta$. We use a count variable $\#_a(\mathbf{w})$ to denote the number of occurrences of a character $a \in \Pi$ in a stack word variable $\mathbf{w}$. An NPCM-term $t$ is defined as follows: [3] $t \ ::= \ n \mid \mathbf{q} \mid \mathbf{x} \mid \#_a(\beta_w) \mid \beta_{x_i} \mid \beta_q \mid t - t \mid t + t$, where $n$ is an integer and $a \in \Pi$. An NPCM-formula $P$ is defined as follows: $P \ ::= \ t > 0 \mid t \bmod n = 0 \mid \neg P \mid P \vee P$, where $n \neq 0$ is an integer. Thus, $P$ is a Presburger formula over control state variables, clock value variables and count variables. The Presburger safety analysis problem is: given a past pushdown timed automata and an NPCM-formula $P$, is there a reachable configuration satisfying $P$? From Theorem 1, Theorem 2, Theorem 6, and the proof of the Theorem 10 in [15], we have,

**Theorem 7.** *The Presburger safety analysis problem for past pushdown timed automata is decidable.*

Because of Theorem 7, the following statement can be verified:

"A given past pushdown timed automaton can reach a configuration satisfying $z_1 - z_2 + 2z_3 > \#_a(w) - 4\#_b(w)$."

where $z_1$, $z_2$, and $z_3$ are clocks and $w$ is the stack word in the configuration. Obviously, Theorem 6 and Theorem 7 still hold when a past pushdown timed automaton is augmented with a number of reversal-bounded counters, i.e., a past reversal-bounded pushdown timed automaton. The reason is as follows. In the proof of Theorem 6, clocks in a past pushdown timed automaton are simulated by reversal-bounded counters. Therefore, by replacing past-formulas in a past pushdown timed automaton with Boolean formulas in the proof, a past pushdown timed automaton is simulated by a reversal-bounded NPCM. When a number of reversal-bounded counters are added to the past pushdown timed automaton, the automaton can still be simulated by a reversal-bounded NPCM: clocks are simulated by reversal-bounded counters and the added reversal-bounded counters remain. Hence, Theorem 6 and Theorem 7 still hold for past reversal-bounded pushdown timed automata. An unrestricted counter is a special case of a pushdown stack. Therefore, the results for past reversal-bounded pushdown timed automata imply the same results for past timed automata with a number of reversal-bounded counters and an unrestricted counter. These results are helpful in verifying Presburger safety properties for history-dependent systems containing parameterized (unspecified) integer constants, as illustrated by the example in the next section.

## 6 An Example

This section considers an ASTRAL specification [22] of a railroad crossing system, which is a history-dependent and parameterized real-time system with a Presburger safety property that needs to be verified. The system description is taken from [17]. The system consists of a set of railroad tracks that intersect a street where cars may

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

cross the tracks. A gate is located at the crossing to prevent cars from crossing the tracks when a train is near. A sensor on each track detects the arrival of trains on that track. The critical requirement of the system is that whenever a train is in the crossing the gate must be down, and when no train has been in between the sensors and the crossing for a reasonable amount of time, the gate must be up. The complete ASTRAL specification of the railroad crossing system can be found in [22] and at `http://www.cs.ucsb.edu/∼dang`. The ASTRAL specification was proved to be correct by using the PVS-based ASTRAL theorem prover [22] and was tested by a bounded-depth symbolic search technique [14].

The ASTRAL specification looks at the railroad crossing system as two interactive modules or process specifications: `Gate` and `Sensor`. Each process has its own (parameterized) constants, local variables and transition system. Requirement descriptions are also included as a part of a process specification. ASTRAL is a rich language and has strong expressive power. For a detailed introduction to ASTRAL and its formal semantics the reader is referred to [8, 10, 22]. For the purpose of this paper, we will show that the `Gate` process can be modeled as a past pushdown timed automaton with reversal-bounded counters. By using the results in the previous section, a Presburger safety property specified in `Gate` can be automatically verified.

We look at an instance of the `Gate` process by considering the specification with one railroad track (i.e., `n_track`=1, and therefore there is only one `Sensor` process instance.) and assigning concrete values to parameterized constants as follows (in order to make the enabling conditions in the process in the form of past formulas):

`raise_dur`=1, `up_dur`=1, `lower_dur`=1, `down_dur`=1,
`raise_time`=1, `lower_time`=1, `response_time`=1, `RIImax`=6.

But two constants `wait_time` and `RImax` remain parameterized.

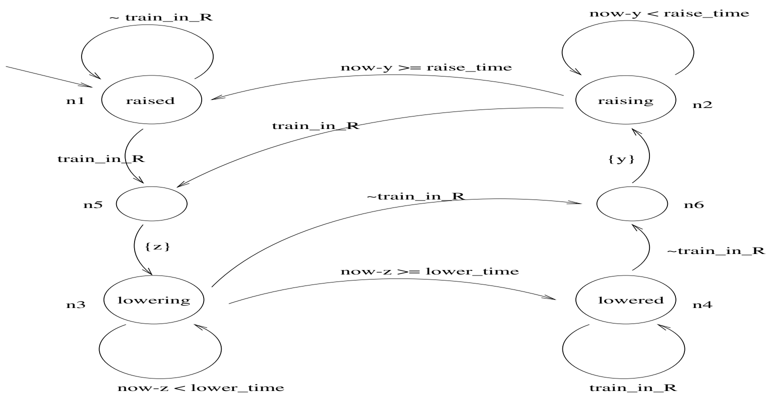

**Fig. 1.** The transition system of a `Gate` instance represented as a timed automaton

The transition system of the `Gate` process can be represented as the timed automaton shown in Figure 1. The local variable `position` in `Gate` has four possible values. They are `raised`, `raising`, `lowering` and `lowered`, which are represented by nodes $n_1, n_2, n_3$ and $n_4$ in the figure, respectively. There are two dummy nodes $n_5$ and $n_6$ in the graph, which will be made clear in a moment. The initial node is $n_1$. That is, the initial position of the gate is `raised`.

The transitions `lower, down, raise` and `up` in the `Gate` process are represented in the figure as follows. Each transition includes a pair of entry and exit assertions with a nonzero duration associated with each pair. The entry assertion must be satisfied at the time the transition starts, whereas the exit assertion will hold after the time indicated by the duration from when the transition fires. The transition `lower`,

```
TRANSITION lower
ENTRY [ TIME : lower_dur ]
  ~ ( position = lowering | position = lowered )
  & EXISTS s: sensor_id ( s.train_in_R )
EXIT
  position = lowering,
```

corresponds to the edges $\langle n_1, n_5 \rangle$ and $\langle n_5, n_3 \rangle$, or the edges $\langle n_2, n_5 \rangle$ and $\langle n_5, n_3 \rangle$. The clock $z$ is used to indicate the end time $\texttt{End}(\texttt{lower})$ (of transition `lower`) used in transition `down`. Whenever the transition `lower` completes, $z$ jumps to *now*. Thus, a dummy node $n_5$ is introduced such that $z$ jumps on the edge $\langle n_5, n_3 \rangle$ to indicate the end of the transition `lower`. On an edge without clock jumps (such as $\langle n_1, n_5 \rangle$ and $\langle n_2, n_5 \rangle$), *now* progresses by one time unit. Thus, the two edges $\langle n_1, n_5 \rangle$ and $\langle n_2, n_5 \rangle$ indicate the duration `lower_dur` of the transition `lower` (recall the parameterized constant `lower_dur` was set to be 1.).

Similarly, transition `raise` corresponds to the edges $\langle n_3, n_6 \rangle$ and $\langle n_6, n_2 \rangle$, or the edges $\langle n_4, n_6 \rangle$ and $\langle n_6, n_2 \rangle$. The other two transitions `down` and `up` correspond to the edges $\langle n_3, n_4 \rangle$ and $\langle n_2, n_1 \rangle$, respectively. Idle transitions need to be added to indicate the behavior of the process when no transition is enabled and executing. They are represented by self-loops on nodes $n_1, n_2, n_3$ and $n_4$ in the figure.

Besides variable `position`, `Gate` has an imported variable `train_in_R`, which is a local variable of the `Sensor` process, to indicate an arrival of a train. `Gate` has no control over the imported variable. That is, `train_in_R` can be either true or false at any given time, even though we do not explicitly specify this in the figure. But not all the execution sequences of the `Gate` process are intended. For instance, consider the scenario that `train_in_R` has value *true* at $now = 2$ and the value changes to $false$ at $now = 3$. This change is too fast, since the gate `position` at $now = 3$ may be `lowering` when the change happens. At $now = 3$, the train had already crossed the intersection. This is bad, since the gate was not in the fully lowered position `lowered`. Thus, the imported variable clause is needed to place extra requirements on the behaviors of the imported variable. The requirement essentially states that once the sensor reports a train's arrival, it will keep reporting a train at least as long as it takes the fastest train to exit the region. By substituting for the parameterized constants and noticing that there is only one sensor in the system, the imported variable clause in the ASTRAL specification can be written as

$$\begin{array}{l} \quad now \geq 1 \wedge \texttt{past}(\texttt{train\_in\_R}, now - 1) = true \wedge \texttt{train\_in\_R} = false \\ \rightarrow \\ \quad now \geq 5 \wedge \forall t (t \geq now - 5 \wedge t < now \rightarrow \texttt{past}(\texttt{train\_in\_R}, t) = true). \end{array}$$

We use $f$ to denote this clause. It is easy to see that $f$ is a past formula. Figure 1 can be modified by adding $f$ to the enabling condition of each edge. The resulting automaton is denoted by $M$.

It is easy to check that $M$ does rule out the unwanted execution sequences shown above. Now we use clock $x$ to indicate the (last) change time of the imported variable `train_in_R`. A proper modification to $M$ can be made by incorporating clock $x$ into the automaton. The resulting automaton, denoted by $M'$, is a past pushdown timed automaton without the pushdown stack. Recall that the process instance has two parameterized constants `wait_time` and `RImax`. Therefore, $M'$ is augmented with two reversal-bounded counters `wait_time` and `RImax` to indicate the two constants. These two counters remain unchanged during the computations of $M'$ (i.e., 0-reversal-bounded). They are restricted by the axiom clause $g$ of the process:

```
   wait_time >= raise_dur + raise_time + up_dur
 & RImax >= response_time + lower_dur +
       lower_time + down_dur + raise_dur
 & RImax >= response_time + lower_dur +
       lower_time + down_dur + up_dur
```

recalling that all the constants in the clause have concrete values except `wait_time` and `RImax`.

The first conjunction of the schedule clause of the process instance specifies a safety property such that the gate will be down before the fastest train reaches the crossing; i.e., $(\texttt{train\_in\_R} = \mathit{true} \ \wedge \ \mathit{now} - x \geq \texttt{RImax} - 1) \rightarrow \texttt{position} = \texttt{lowered}$. We use $p$ to denote this formula. Notice that $p$ is a non-region property (since `RImax` is a parameterized constant). Verifying this part of the schedule clause is equivalent to solving the Presburger safety analysis problem for $M'$ (augmented with two reversal-bounded counters) with the Presburger safety property $g \rightarrow p$ over the clocks and the reversal-bounded counters. From the result of the previous section, this property can be automatically verified.

**Acknowledgement.** The authors would like to thank P. San Pietro and J. Su for discussions. The ASTRAL specification used in this paper was written by P. Kolano. The work by Dang and Kemmerer was supported by DARPA F30602-97-1-0207. The work by Bultan was supported by NSF CCR-9970976 and NSF CAREER award CCR-9984822. The work by Ibarra was supported by NSF IRI-9700370.

## References

1. R. Alur, C. Courcoubetis, and D. Dill, *"Model-checking in dense real time," Information and Computation*, **104** (1993) 2-34
2. R. Alur and D. Dill, *"A theory of timed automata," TCS*, **126** (1994) 183-236
3. R. Alur, T. Feder, and T. A. Henzinger, *"The benefits of relaxing punctuality," J. ACM*, **43** (1996) 116-146
4. R. Alur, T. A. Henzinger, *"Real-time logics: complexity and expressiveness," Information and Computation*, **104** (1993) 35-77
5. R. Alur, T. A. Henzinger, *"A really temporal logic," J. ACM*, **41** (1994) 181-204
6. A. Bouajjani, J. Esparza, and O. Maler, *"Reachability analysis of pushdown automata: application to model-checking,", CONCUR'97*, LNCS **1243**, pp. 135-150

7. A. Bouajjani, R. Echahed, and R. Robbana, *"On the automatic verification of systems with continuous variables and unbounded discrete data structures," Hybrid System II*, LNCS **999**, 1995, pp. 64-85
8. A. Coen-Porisini, C. Ghezzi and R. Kemmerer, *"Specification of real-time systems using ASTRAL," TSE*, **23** (1997) 572-598
9. H. Comon and Y. Jurski, *"Timed automata and the theory of real numbers," CONCUR'99*, LNCS **1664**, pp. 242-257
10. A. Coen-Porisini, R. Kemmerer and D. Mandrioli, *"A formal framework for ASTRAL intralevel proof obligations," TSE*, **20** (1994) 548-561
11. Z. Dang, *"Verification and debugging of infinite state real-time systems,"* PhD Dissertation, UCSB, August 2000. Available at `http://www.cs.ucsb.edu/~dang`.
12. Z. Dang, *"Binary reachability analysis of timed pushdown automata with dense clocks," CAV'01*, LNCS **2102**, pp. 506-517
13. Z. Dang and R. A. Kemmerer, *"Using the ASTRAL model checker to analyze Mobile IP," ICSE'99*, pp. 132-141
14. Z. Dang and R. A. Kemmerer, *"Using the ASTRAL symbolic model checker as a specification debugger: three approximation techniques," ICSE'00*, pp. 345-354
15. Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer and J. Su, *"Binary reachability analysis of discrete pushdown timed automata," CAV'00*, LNCS **1855**, pp. 69-84
16. A. Finkel, B. Willems and P. Wolper, *"A direct symbolic approach to model checking pushdown systems,"* INFINITY'97
17. C. Heitmeyer and N. Lynch. "The generalized railroad crossing: a case study in formal verification of real-time systems," *RTSS'94*, pp. 120-131
18. T. A. Henzinger, Z. Manna, and A. Pnueli, *"What good are digital clocks?," ICALP'92*, LNCS **623**, pp. 545-558
19. T. A. Henzinger and Pei-Hsin Ho, *"HyTech: the Cornell hybrid technology tool," Hybrid Systems II*, LNCS **999**, 1995, pp. 265-294
20. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, *"Symbolic model checking for real-time systems," Information and Computation*, **111** (1994) 193-244
21. O. H. Ibarra, *"Reversal-bounded multicounter machines and their decision problems," J. ACM*, **25** (1978) 116-133
22. P. Z. Kolano, Z. Dang and R. A. Kemmerer. "The design and analysis of real-time systems using the ASTRAL software development environment," *Annals of Software Engineering*, **7** (1999) 177-210
23. K. G. Larsen, P. Pattersson, and W. Yi, *"UPPAAL in a nutshell," International Journal on Software Tools for Technology Transfer*, **1** (1997) 134-152
24. F. Laroussinie, K. G. Larsen, and C. Weise, *"From timed automata to logic - and back," MFCS'95*, LNCS **969**, pp. 529-539
25. Amir Pnueli, *"The temporal logic of programs," FOCS'77*, pp. 46–57
26. J. Raskin and P. Schobben, *"State clock logic: a decidable real-time logic," HART'97*, LNCS 1201, pp. 33-47
27. H. Straubing, *Finite automata, formal logic, and circuit complexity.* Birkhauser, 1994
28. W. Thomas, *"Automata on infinite objects,"* in *Handbook of Theoretical Computer Science, Volume B* (J. van Leeuwen eds.), Elsevier, 1990
29. T. Wilke, *"Specifying timed state sequences in powerful decidable logics and timed automata,"* LNCS 863, pp. 694-715, 1994
30. S. Yovine, *"A verification tool for real-time systems," International Journal on Software Tools for Technology Transfer*, **1** (1997): 123-133