Past Pushdown Timed Automata and Safety Verification [1]

Zhe Dang, Tevfik Bultan, Oscar H. Ibarra, and Richard A. Kemmerer [2,3,4,5]

Abstract

We consider past pushdown timed automata that are discrete pushdown timed automata 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 counter machine augmented with a pushdown stack (i.e., a reversal-bounded NPCM). By 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. We also investigate the reachability problem for a class of transition systems under some fairness constraints in the form of generalized past formulas. Finally, we present an example ASTRAL specification to demonstrate the usefulness of the results.

Key words: Timed automata, automated verification, temporal logic, past formulas, Presburger arithmetic

[1] 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.

[2] School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164. EMAIL: zdang@eecs.wsu.edu, TEL: (+1) 509-335-7238, FAX: (+1) 509-335-3818.

[3] Department of Computer Science, University of California, Santa Barbara, CA 93106. EMAIL: bultan@cs.ucsb.edu, TEL: (+1) 805-893-3735, FAX: (+1) 805-893-8553.

[4] Department of Computer Science, University of California, Santa Barbara, CA 93106. EMAIL: ibarra@cs.ucsb.edu, TEL: (+1) 805-893-4171, FAX: (+1) 805-893-8553.

[5] Department of Computer Science, University of California, Santa Barbara, CA 93106. EMAIL: kemm@cs.ucsb.edu, TEL: (+1) 805-893-4232, FAX: (+1) 805-893-8553.

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 and for model-checking tools (see [1] for a survey). The technique is useful, but obviously not enough since a lot of interesting properties cannot be expressed as clock regions. There has been some work [7–9] concerning verification of timed automata for non-region reachability.

In this paper, we consider a class of discrete timed systems, called past pushdown timed automata. In a past pushdown timed automaton, clocks take nonnegative integer values (i.e., discrete clocks) and 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). Since all the clocks are initially 0, it is always true that $x_1, x_2 \leq now$. 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 [12], 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 [4], nor by model-checking pushdown systems [5].

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 [6], 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 [14] for a number of interesting real-time systems specified in ASTRAL). Therefore, the results obtained in this paper would be useful in implementing a symbolic model checker for a subset of ASTRAL.

Past formulas are not new. In fact, they can be expressed in TPTL [3], which is obtained by including clock constraints (in the form of clock regions) and freeze quantifiers in the Linear Temporal Logic (LTL) [16]. 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 [18] and Straubing [17] 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.

We also investigate the reachability problem for a class of transition systems under some fairness constraints. Though the systems are traditional (i.e., history-independent), the constraints are history-dependent. For instance, consider a finite automaton M. We use "time" to indicate the number of moves executed so far. Is it possible that state q can reach state q' during which there are times $t_1 < t_2$ such that $3(\#_q(t_1) - \#_{q'}(t_2)) > t_1 - t_2 \wedge 2(\#_q(t_2) - \#_{q'}(t_1)) < t_1 + t_2$? In the question, $\#_q(t_1)$ means the total number of visits to state q at time t_1. Notice that the constraint allows accesses to past values of an unbounded counter (e.g., $\#_q(t_1)$) and Presburger relations between time variables and these past values. We show that the above mentioned question is decidable while, in general, it is undecidable when one replaces "there are" with "for all" in the question statement. We were also able to generalize M into a more powerful class of transition systems than finite automata.

2 Preliminaries

A *nondeterministic multicounter machine (NCM)* is a nondeterministic machine with a finite set of states, and a finite number of counters with integer counter values. Each counter can add 1, subtract 1, or stay unchanged. A counter can also be tested against an integer constant. A *nondeterministic pushdown multicounter machine (NPCM)* M is an NCM augmented with a pushdown stack. In addition to counter operations, an NPCM can pop the top symbol from the stack or push a word on the top of the stack. A counter is *r-reversal-bounded* if it changes mode between nondecreasing and nonincreasing at most r 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. An NCM (or NPCM) M is reversal-bounded if each counter in M is r-reversal-bounded for some r that is independent of the computations. 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 M is

used as a language recognizer we attach a separate one-way read-only input tape to the machine and assign a state as the 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 [12]:

Theorem 2.1 *The emptiness problem for reversal-bounded nondeterministic (pushdown) multicounter machines with a one-way input tape is decidable.*

We use $\mathbf{N}$ to denote nonnegative integers. Let A be a finite set of finite state variables, i.e., their domains are a bounded range in $\mathbf{N}$. We use $a, b \cdots$ to denote them. Without loss of generality, we can assume they are Boolean variables. Let X be a finite set of variables in $\mathbf{N}$ with $now \notin X$ being a variable indicating the current time. *Past formulas* are defined as

$$f ::= a(y) \mid y < c \mid y < z + c \mid \Box_x f \mid f \vee f \mid \neg f,$$

where $a \in A$, y and z are in $X \cup \{now\}$, $x \in X$, and c is an integer. Intuitively, $a(y)$ is the variable a's value at time y (i.e., $\mathtt{Past}(a, y)$ in ASTRAL [6]). *Quantification* $\Box_x f$, 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 $\Box_x f$ 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 $\mathbf{n} \in \mathbf{N}$ represents the value of now, and for each $a \in A$, the mapping $\|a\| : 0..\mathbf{n} \to \{true, false\}$ gives the Boolean value of a at each time point from 0 to $\mathbf{n}$. Let $\mathbf{B} : X \to 0..\mathbf{n}$ be a *valuation* for variables in X. We use $\mathbf{B}(k/x)$ to denote replacing x's value in the valuation with a non-negative integer k. Given a history H and a valuation $\mathbf{B}$, the interpretations of past formulas are as follows, for each $y, z \in X \cup \{now\}$ and each $x \in X$,

$$\|x\|_{H,\mathbf{B}} = \mathbf{B}(x), \qquad \|now\|_{H,\mathbf{B}} = \mathbf{n},$$

$$\|a(y)\|_{H,\mathbf{B}} \iff \|a\|(\|y\|_{H,\mathbf{B}}), \qquad \|y < c\|_{H,\mathbf{B}} \iff \|y\|_{H,\mathbf{B}} < c,$$

$$\|\neg f\|_{H,\mathbf{B}} \iff \text{not } \|f\|_{H,\mathbf{B}}, \qquad \|f_1 \vee f_2\|_{H,\mathbf{B}} \iff \|f_1\|_{H,\mathbf{B}} \text{ or } \|f_2\|_{H,\mathbf{B}},$$

$$\|y < z + c\|_{H,\mathbf{B}} \iff \|y\|_{H,\mathbf{B}} < \|z\|_{H,\mathbf{B}} + c,$$

$$\|\Box_x f\|_{H,\mathbf{B}} \iff \text{for all } k \text{ with } 0 \leq k \leq \mathbf{n}, \|f\|_{H,\mathbf{B}(k/x)},$$

When f is a closed formula, we write $\|f\|_H$ instead of, for all $\mathbf{B}$, $\|f\|_{H,\mathbf{B}}$. $\|f\|_H$ is the truth value of f under history H. 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 $\mathbf{n}+1$. An important question about the past formulas is the following: Is there any way to calculate the truth value of a closed past 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 following theorem shows that this can be done.

A *Boolean function* is a mapping $\mathbf{N} \rightarrow \{true, false\}$. A Boolean predicate is a mapping $\{true, false\}^m \rightarrow \{true, false\}$ 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. 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 $1 \leq i \leq |A|$. Under the given $v_1, \cdots, v_{|A|}$, we use $u, u_1, \cdots, u_k$ to denote the Boolean functions for closed past formulas f, $g_1, \cdots, g_k$, respectively, in the following theorem.

Theorem 2.2 *For any closed past formula f, there are closed past formulas $g_1, \cdots, g_k$, and Boolean predicates $O, O_1, \cdots, O_k$ (for some 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 $\mathbf{N}$, $u(t+1) = O(v_1(t+1), \cdots, v_{|A|}(t+1), u_1(t), \cdots, u_k(t))$ 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 [3]. A tableau technique is proposed in [3] to show that validity checking of TPTL is decidable. A modification of the technique can be used to prove Theorem 2.2.

3 Past Pushdown Timed Automata

A pushdown timed automaton models a timed automaton augmented with a pushdown stack. As in a timed automaton, clocks either progress or reset. All the clocks start from 0. A past pushdown timed automaton further allows past formulas as enabling conditions in a pushdown timed automaton. Formally, a *past pushdown timed automaton* M is a tuple $\langle Q, A, Z, \Gamma, E, now \rangle$, where Q is a finite set of states, A is a finite set of Boolean variables, $Z = \{z_1, \cdots, z_m\}$ is a finite set of discrete clocks, Γ is a finite stack alphabet, and $now \notin Z$ represents the current time. E is a finite set of edges, each of which is in the form of $\langle q, \delta, \tau, (\eta, \eta'), l, q' \rangle$, where

$q, q' \in Q$, $\delta \subseteq Z$ is the set of clock jumps, $\tau : A \to \{true, false\}$ is the assignment of Boolean variables in A, the pair of $\eta \in \Gamma$ and $\eta' \in \Gamma^*$ indicates the stack operation (replacing the top symbol of the stack η by a word η'), and past formula l is the enabling condition (free variables in l are contained in Z). A configuration α of M is a tuple $\langle \alpha_Q, \alpha_H, \alpha_Z, \alpha_w \rangle$ where α_Q is a state, $\alpha_H = \langle \{\|a\|^{\alpha_H}\}_{a \in A}, \mathbf{n}^{\alpha_H} \rangle$ is a history, $\alpha_Z : Z \to \mathbf{N}$ is a valuation of clocks in Z, and $\alpha_w \in \Gamma^*$ indicates the stack content. $\alpha \to^e \beta$ denotes a one-step transition along edge $e = \langle q, \delta, \tau, (\eta, \eta'), l, q' \rangle$ in M satisfying:

- The state q is set to q'; i.e., $\alpha_Q = q, \beta_Q = q'$.
- The enabling condition is satisfied, i.e., $\|l\|_{\alpha_H, \mathbf{B}}$ holds for any $\mathbf{B}$ satisfying $\mathbf{B}(z) = \alpha_Z(z)$ for each $z \in Z$. That is, l is evaluated under the history α_H and replacing each free clock variable $z \in Z$ in l by its value in α.
- Each clock changes as follows.
  - If $\delta = \emptyset$, i.e., there are no clock jumps on the edge, then now 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 $z \in Z$, $\beta_Z(z) = \alpha_Z(z)$.
  - If $\delta \neq \emptyset$, then all the clocks in δ jump to now, and the other clocks do not change. That is, for each $z \in \delta$, $\beta_Z(z) = \mathbf{n}^{\alpha_H}$. In addition, for each $z \notin \delta$, $\beta_Z(z) = \alpha_Z(z)$ and the clock now does not progress, i.e., $\mathbf{n}^{\beta_H} = \mathbf{n}^{\alpha_H}$.
- The history is updated as follows.
  - If $\delta = \emptyset$, then now progresses. In this case, the history α_H is extended to β_H by adding the resulting values (given by the assignment τ) of the Boolean variables after the transition. That is, for all $0 \leq t \leq \mathbf{n}^{\alpha_H}$, history β_H is consistent with history α_H; i.e., for all $a \in A$, $\|a\|^{\beta_H}(t) = \|a\|^{\alpha_H}(t)$. In addition, β_H extends α_H; i.e., for each $a \in A$, $\|a\|^{\beta_H}(\mathbf{n}^{\beta_H}) = \tau(a)$.
  - If $\delta \neq \emptyset$, then now does not progress. In this case, for all $a \in A$, for all $0 \leq t \leq \mathbf{n}^{\alpha_H} - 1$, $\|a\|^{\beta_H}(t) = \|a\|^{\alpha_H}(t)$, and $\|a\|^{\beta_H}(\mathbf{n}^{\beta_H}) = \tau(a)$. Thus, even though now does not progress, the current values of variables $a \in A$ may change according to the assignment τ.
- According to the stack operation (η, η'), the stack word α_w is updated to β_w.

Notice that, in M, we use clock jumps (i.e., $z := now$) instead of clock resets ($z := 0$). The reason is that, in this way, the start time of a transition can be directly modeled as a clock. Obviously, $now - z$ gives a "traditional" clock with resets. We write $\alpha \to \beta$ if α can reach β by a one-step transition. We designate a state $q_0 \in Q$ as the initial state. α_0 is the initial configuration in which the state is q_0, the stack word is empty and all clocks including now are 0. We write $\alpha_0 \rightsquigarrow \beta$ if there are $\alpha_1, \cdots, \alpha_k = \beta$ for some k with $\alpha_i \to \alpha_{i+1}$ for each $0 \leq i < k$. A tuple $(q, w, v_0, v_1, \cdots, v_m)$ of a state, a stack word, and clock values for $now, z_1, \cdots, z_m$ is *reachable* if there is a configuration β with $\alpha_0 \rightsquigarrow \beta$ such that in β, the state is q, the stack word is the reverse of word w (the reason for using "reverse" will be clear in a moment), and clock values are $v_0, v_1, \cdots, v_m$ for $now, z_1, \cdots, z_m$, respectively. Let the reachability set R be the set of all the tuples that are reachable. Each tuple in R can be encoded as a string composed of the unary representation of the state,

the stack word, and the unary representation for each clock value, separated by a delimiter. In this way, R is treated as a set of strings, i.e., a language. The rest of this section shows that R can be accepted by a reversal-bounded NPCM.

We first show that each enabling condition in M can be replaced by a closed past formula. To see this, without loss of generality, we may assume that each clock in Z initially jumps (since all the clocks including now start from 0 in M). In this way, during an execution of M, a clock value of z intuitively stands for the most recent time when a jump "$z := now$" happened. We introduce a new Boolean variable $b_i \notin A$ for each $z_i \in Z$. Let B denote the set of all such b_i's. We then construct a machine M' that is exactly the same as M except:

- M' has Boolean variables B in addition to A. For each edge e in M, if e has some jump $z_i := now$, then add $b_i := true$ as an assignment on this edge. For other z_i that do not jump on this edge, b_i is unchanged. If e does not have any jumps, then $b_i = false$ for all i.
- Initially, $b_i = true$ for all i.
- Denote J_i as $b_i(z_i) \wedge \Box_y(z_i < y \leq now \rightarrow \neg b_i(y))$, which means z_i is the last jump time of clock z_i. The enabling condition l on an edge is replaced by a closed past formula $\Box_{z_1} \cdots \Box_{z_m}(J_1 \wedge \cdots \wedge J_m \rightarrow l)$.

Next, we will show that each enabling condition in M' can be replaced by a Boolean variable. In M', we use l_e (a closed past formula) and τ_e to denote the enabling condition and the assignment of all Boolean variables in $\hat{A} = A \cup B$ for an edge e, respectively. From Theorem 2.2, each l_e is associated with a number of Boolean functions $O^{l_e}, O^{l_e}_1, \cdots, O^{l_e}_{k_e}$ and a number of Boolean variables $u^{l_e}_1, \cdots, u^{l_e}_{k_e}$ (updated while now progresses). l_e itself can be considered as a Boolean variable u^{l_e}. We use a primed form to indicate the previous value of a variable – here, a variable changes when time progresses. Thus, from Theorem 2.2, [6] these variables are updated as, $u^{l_e} := O^{l_e}(\hat{A}, (u^{l_e}_1)', \cdots, (u^{l_e}_{k_e})')$ and for all i, $u^{l_e}_i := O^{l_e}_i(\hat{A}, (u^{l_e}_1)', \cdots, (u^{l_e}_{k_e})')$. These updates work only for edges that do not have a jump (since now progresses by one time unit on such edges). If an edge e has a jump (i.e., now does not progress), we need to precompute the updates at the previous transition. To do these precomputations, we need multiple copies of extra Boolean variables $u^{l_e,\tau}, u^{l_e,\tau}_1, \cdots, u^{l_e,\tau}_{k_e}, a^\tau$ for each $a \in \hat{A}$ and for each possible assignment τ for Boolean variables $\hat{A}$. Note that there are $2^{|\hat{A}|}$ choices of τ. As shown below, an edge e_0 in M' is modified to incorporate these updates.

- Change the values of Boolean variables $a \in \hat{A}$ according to the assignment τ_{e_0} given on the edge e_0.
- If e_0 does not have a jump,
  · $now := now + 1$,

[6] We simply use $\hat{A}$ to indicate the current values of each variable in $\hat{A}$ with the assumption that the "current time" can be figured out from the context.

- for each τ and for each $a \in \hat{A}$, $a^\tau := \tau(a)$. Denote all a^τ as A^τ.
- For **all** edges e, and for each τ and for each $1 \leq i \leq k_e$, $u_i^{l_e,\tau} := O_i^{l_e}(A^\tau, (u_1^{l_e})', \cdots, (u_{k_e}^{l_e})')$ and $u^{l_e,\tau} := O^{l_e}(A^\tau, (u_1^{l_e})', \cdots, (u_{k_e}^{l_e})')$.
- For **all** edges e, u^{l_e} is assigned the precomputed value according to the assignment τ_{e_0}, i.e., $u^{l_e} := u^{l_e,\tau_{e_0}}$. Also, $u_i^{l_e} := u_i^{l_e,\tau_{e_0}}$ for $1 \leq i \leq k_e$.

- If e_0 has at least one jump,
  - execute all the jumps $z_i := now$ on e_0. But now does not progress.
  - For **all** edges e, u^{l_e} is assigned as the precomputed value according to the assignment τ_{e_0}; i.e., $u^{l_e} := (u^{l_e,\tau_{e_0}})'$, and $u_i^{l_e} := (u_i^{l_e,\tau_{e_0}})'$ for $1 \leq i \leq k_e$. All other Boolean variables are unchanged.

The initial values of Boolean variables $u^{l_e}, u_j^{l_e}$ and $u^{l_e,\tau}, u_j^{l_e,\tau}$ can be assigned using the initial value of a and $\tau(a)$. Each enabling condition l_e in M' is replaced by the Boolean variable u^{l_e}.

Now, we will show that the reachability set R of M' (and hence M) can be accepted by an NPCM. Let $\hat{M}$ be an NPCM that is exactly the same as M' except for the following. Given (the string encoding) of a tuple $(q, w, v_0, v_1, \cdots, v_m)$ on the input tape, $\hat{M}$ simulates M''s computation from the initial configuration α_0 (by properly calculating the initial values of the Boolean variables $u^{l_e}, u_j^{l_e}, u^{l_e,\tau}, u_j^{l_e,\tau}, a \in \hat{A}$). Updates (shown above) to these Boolean variables are implemented by the finite control of $\hat{M}$. The stack operations of M' are faithfully simulated by $\hat{M}$ using its own stack. $\hat{M}$ uses counters $x_0, x_1, \cdots, x_m$ to simulate clocks $now, z_1, \cdots, z_m$, respectively, in M'. Initially, each counter is 0. However, whenever M' executes $now := now + 1$, $\hat{M}$ increases all the counters by 1, i.e., $x_i := x_i + 1$ for each $0 \leq i \leq m$. When M' executes a jump $z_i := now$, $\hat{M}$ does nothing to the counters. For each x_i with $i \neq 0$, at some point, either initially or at the moment $z_i := now$ is being executed by M', $\hat{M}$ guesses (only once for each i) that x_i has already reached the value v_i of z_i given on the input tape. After such a guess for i, an execution of $now := now + 1$ will not cause $x_i := x_i + 1$ as indicated above (i.e., x_i will no longer be incremented). However, after such a guess for i, a later execution of a jump $z_i := now$ in M' will cause $\hat{M}$ to abort abnormally (without accepting the input). At some point after all x_i, $1 \leq i \leq m$, have been guessed, $\hat{M}$ guesses that now in M' has reached the value v_0 on the input tape. Then, $\hat{M}$ compares its current state, stack word, and clock values with the ones on the input tape. Comparing the stack word requires $\hat{M}$ to pop its stack while reading the w on the input tape (recalling that the stack word is reversed in R). Comparing an x_i with v_i on the input tape requires $\hat{M}$ to decrement x_i to 0 while reading the encoding of v_i. $\hat{M}$ accepts iff the comparisons succeed. Clearly, $\hat{M}$ accepts R and $x_0, x_1, \cdots, x_k$ are reversal-bounded. Hence,

Theorem 3.1 *The reachability 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. Let the stack alphabet $\Gamma = \{\gamma_1, \cdots, \gamma_k\}$. For a stack word w, $\#_\gamma(w)$ is used to indicate the number of occurrences of a stack symbol $\gamma \in \Gamma$. Let P be a Presburger formula over $k + m + 2$ variables. We say that a configuration α of M satisfies P if $P(q, \#_{\gamma_1}(w), \cdots, \#_{\gamma_k}(w), v_0, v_1, \cdots, v_m)$ holds, where in α, the state is q (understood as a value taken from a bounded range of integers), the stack word is w, the clock values are $v_0, v_1, \cdots, v_m$ for $now, z_1, \cdots, z_m$, respectively. The Presburger safety analysis problem for M is to decide whether, given a Presburger formula P, there is a configuration β that is reachable from the initial configuration α_0 and that satisfies P. In practice, P is used to specify an unsafe property for a system design M. Therefore, the existence of a witness β implies an error in the design. For instance, can a given past pushdown timed automaton reach an undesired configuration satisfying $z_1 - z_2 + 2z_3 > \#_{\gamma_1}(w) - 4\#_{\gamma_2}(w)$? The following theorem states that detecting the errors is decidable.

Theorem 3.2 *The Presburger safety analysis problem for past pushdown timed automata is decidable.*

Proof. Given an instance of the problem for M and P, consider the set R' of integer tuples $(q, w, v_0, v_1, \cdots, v_m, n_1, \cdots, n_k)$ satisfying the following three conditions: (a) $(q, w, v_0, v_1, \cdots, v_m) \in R$, (b) $P(q, n_1, \cdots, n_k, v_0, v_1, \cdots, v_m)$, and (c) $\#_{\gamma_i}(w) = n_i, 1 \leq i \leq k$. From Theorem 3.1, (a) can be verified by a reversal-bounded NPCM. (b) can be verified by a deterministic reversal-bounded NCM [12]. Obviously, (c) can also be verified by a deterministic reversal-bounded NCM. Hence, R' can be accepted by a reversal-bounded NPCM obtained by intersecting the three machines. Clearly, R' is not empty iff the instance of the Presburger safety analysis problem is true. The result follows from Theorem 2.1. □

Theorem 3.1 and Theorem 3.2 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. This is because, in the proof of Theorem 3.1, clocks in a past pushdown timed automaton are simulated by reversal-bounded counters. 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. 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 Section 5.

In a past formula, besides quantification, only comparisons of one integer variable or the difference of two integer variables against an integer constant are allowed.

Though there are practical needs to augment past formulas with more complex constructs (such as allowing "+" operations, e.g., "$y_1 + ... + y_n < z_1 + ... + z_k + c$" in place of "$y < z + c$" in the definition of past formulas), the "Turing computing" power of such augmented automata prevents automatic verification of simple properties such as reachability [2]. As a result of the generalization, Theorem 3.2 no longer holds, even when the past formulas do not contain any quantifications.

However, the purpose of embedding past formulas in pushdown timed automata is to make it possible to specify some history-dependent systems. For many traditional history-independent systems (such as an arithmetic program), history-dependency can be specified through a fairness constraint to restrict a reachability path that leads one configuration to another. Would it be possible to allow more general past formulas as the constraints such that the reachability is decidable? We will elaborate this problem in the next section.

4 The $\Box$ and $\Diamond$ Presburger Safety Verification

We first consider a simple computation model M that is a finite automaton augmented with a number of monotonic counters $z_1, \cdots, z_m$. Each move causes a state transition and at most one counter is incremented by 1. M is nondeterministic. A configuration is a tuple of a state and counter values. For a given path $\alpha_0 \cdots \alpha_n$ of length n for some n that leads from $\alpha = \alpha_0$ to $\beta = \alpha_n$ in M, we may, for notational convenience, simply use $q(t)$ and $z_i(t)$ to denote the value of the state and the counter z_i in configuration α_t, for all $1 \leq t \leq n$. Though M is an untimed model, we may interpret $z_i(t)$ as the value of z_i at "time" t (thus each move takes one time unit). In the following, we introduce a notion of reachability from α to β by restricting the intermediate configurations on the path.

Let $k > 0$ and P be a Presburger formula over $k(m+2)$ variables. We say that β is $\Diamond P$-reachable from α, written $\alpha \rightsquigarrow^{\Diamond P} \beta$, if there is a number n such that α reaches β through a path of length n on which $\exists 0 \leq t_1, \cdots, t_{k-1} \leq t_k = n$ satisfying $P(q(t_1), z_1(t_1), \cdots, z_m(t_1), t_1, \cdots, q(t_k), z_1(t_k), \cdots, z_m(t_k), t_k)$. Similarly, we may define $\alpha \rightsquigarrow^{\Box P} \beta$ by replacing $\exists$ with $\forall$ in the above definition. Intuitively, $\Diamond P$ and $\Box P$ can be understood as a form of generalized past formulas. For instance, the following reachability from α to β can be defined as $\alpha \rightsquigarrow^{\Box P} \beta$ for some P: if on a path from α to β, for any times t_1 and t_2, it is always true that $z_1(t_1) + z_2(t_2) > z_1(t_2) + z_2(t_1) + t_2$. In the language of past formulas, $\Box P$ corresponds to the following formula: $\Box_{t_1} \Box_{t_2} (z_1(t_1) + z_2(t_2) > z_1(t_2) + z_2(t_1) + t_2)$ interpreted on the history constructed from a path from α to β. However, comparing to past formulas, $\Box P$ allows (a) access to past values of counters (e.g., $z_1(t_1)$) instead of Boolean variables and (b) linear constraints over the past values and time variables. The $\Box$ (resp. $\Diamond$) Presburger safety analysis problem is to decide, given M and P, whether $\alpha \rightsquigarrow^{\Box P} \beta$ (resp. $\alpha \rightsquigarrow^{\Diamond P} \beta$) for some α and β.

Theorem 4.1 *The $\Box$ Presburger safety analysis problem is undecidable for finite automata augmented with monotonic counters.*

Proof. From [13], the halting problem for the following M is undecidable. M is a deterministic finite automaton augmented with 3 monotonic counters, C_1, C_2, and T which are initially zero. M's enabling conditions involve tests $T = C_1$? or $T = C_2$?. From this model of M, one can easily construct a finite automaton augmented with 3 monotonic counters M' and a Presburger formula P such that $\alpha \rightsquigarrow^{\Box P} \beta$ for some α and β of M' iff M halts. $\Box$

Theorem 4.2 *The $\Diamond$ Presburger safety analysis problem is decidable for finite automata augmented with monotonic counters.*

Proof. Given M with monotonic counters $z_1, \cdots, z_m$ and Presburger formula P over $k(m+2)$ variable as an instance of the problem, we will construct a reversal-bounded NCM M' as follows. M' is equipped with k sets of counters and a one-way input tape. The j-th set contains $m+1$ monotonic counters $C_1^j, \cdots, C_m^j, D^j$. Every counter is initially zero. We use $\mathbf{incrC}(j,i)$ (resp. $\mathbf{incrD}(j)$) to denote the subroutine that increments every C_i^h (resp. D^h) by one, $j \leq h \leq k$, and leaves all the other counters unchanged. M' works as follows. For each $1 \leq i \leq m$, M' repeatedly performs $\mathbf{incrC}(1,i)$ for some times (nondeterministically chosen for each i). Now, M' guesses a state q_1 and starts simulating M from q_1. Let $j := 1$. In the simulation, whenever M performs an increment on a move, say $z_i := z_i + 1$, M' performs $\mathbf{incrC}(j,i)$. Each move of M also causes M' to run $\mathbf{incrD}(j)$. M' continues the simulation during which, nondeterministically, M' increments j by 1 (at the moment M' remembers the state q_j of M). At the time when j reaches k, M' shuts down the simulation. Now, M' checks that

$$P(q_1, C_1^1, \cdots, C_m^1, D^1, \cdots, q_k, C_1^k, \cdots, C_m^k, D^k).$$

The checking requires a bounded number of counter reversals (with the help of additional reversal-bounded counters). M' accepts if the checking succeeds. In the simulation, $q_j, C_1^j, \cdots, C_m^j, D^j$ in M' are used to denote $q(t_j), z_1(t_j), \cdots, z_m(t_j), t_j$ in M, respectively. Clearly, M' accepts a nonempty language iff $\alpha \rightsquigarrow^{\Diamond P} \beta$ in M for some α and β. The result follows from Theorem 2.1. $\Box$

Theorem 4.2 can be generalized. Let M now be further augmented with a pushdown stack (the stack alphabet is Γ). One may similarly define $\alpha \rightsquigarrow^{\Diamond P} \beta$ for M; the differences are that (1) configurations α and β also include the stack content, and (2) in P, one may have additional variables, for each $\gamma \in \Gamma$, $\#_\gamma(t_1), \cdots, \#_\gamma(t_k)$ to indicate the number of symbol γ's in the stack at times $t_1, \cdots, t_k$, respectively. After properly modifying M' into a reversal-bounded NPCM (instead of a reversal-bounded NCM) in the proof of Theorem 4.2, one may conclude that the $\Diamond$ Presburger safety analysis problem is decidable for M. (In modifying M', one introduces two monotonic counters $+_\gamma^j$ and $-_\gamma^j$, for each symbol $\gamma \in \Gamma$ and each

$1 \leq j \leq k$. $+^j_\gamma$ (resp. $-^j_\gamma$) is used to record the pushes (resp. pops) of γ's in M'. Obviously, at time t_j, the actual number of symbol γ's in the stack equals $+^j_\gamma$ minus $-^j_\gamma$.) In fact, if one generalizes the monotonic counters in M to reversal-bounded counters (so M is a reversal-bounded NPCM), the conclusion still holds. This is because a reversal-bounded counter can be simulated by finitely many monotonic counters. For instance, for a 2-reversal-bounded counter C, we may use C^+ to record the increments made towards C and use C^- to record the decrements. Both C^+ and C^- are monotonic with $C = C^+ - C^-$. Hence,

Theorem 4.3 *The $\diamond$ Presburger safety analysis problem is decidable for reversal-bounded NPCMs.*

Theorem 4.3 can be used to verify fairly complex properties. For instance, consider a pushdown automaton M with stack symbols γ_1 and γ_2. Given two states q and q', are there stack words w and w' such that M moves from (q, w) to (q', w') during which the following conditions are satisfied: (1) the total number of visits to state q is no more than $\#_{\gamma_1}(w)+\#_{\gamma_1}(w')$, and (2) the stack is once higher than $\#_{\gamma_1}(w')$ but $|w|$ time units later it is lower than $\#_{\gamma_2}(w')$? From Theorem 4.3, one can show (by adding additional monotonic counters to M) that the problem can be automatically verified.

5 An Example

This section considers an ASTRAL specification [15] 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 [11]. The system consists of a set of railroad tracks that intersect a street where cars may 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 [15] and at `http://www.cs.ucsb.edu/~dang`. The ASTRAL specification was proved to be correct by using the PVS-based ASTRAL theorem prover [15] and was tested by a bounded-depth symbolic search technique [10].

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 [6,15]. 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 Section 3, 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., there is only one `Sensor` process instance.) and assigning concrete values to some parameterized constants in order to have the enabling conditions in the process in the form of past formulas. 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`.

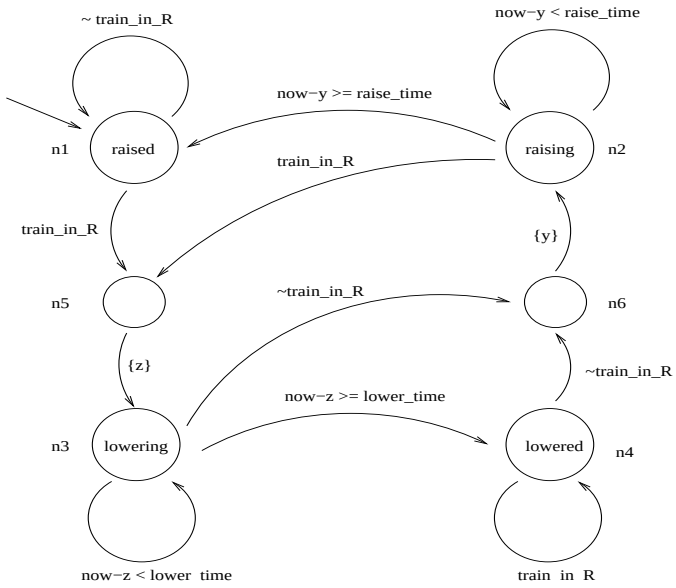

Fig. 1. The transition system of a `Gate` instance represented as a timed automaton

There are four transitions `lower, down, raise` and `up` in the `Gate` process. 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

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

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 rules 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. We assign concrete values to all the parameterized constants except for `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, which is a linear constraint over all the constants including `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., $(\texttt{train\_in\_R} = true \wedge now - x \geq \texttt{RImax} - 1) \rightarrow \texttt{position} = \texttt{lowered}$. 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 results of Section 3, this property can be automatically verified.

6 Conclusions

In this paper, we showed that the reachability set of a past pushdown timed automaton can be accepted by an NPCM. From this result, model-checking past pushdown timed automata against Presburger safety properties on discrete clocks and stack word counts is decidable. We also studied the reachability problem for a class of transition systems under some fairness constraints in the form of generalized past formulas. An example ASTRAL specification was presented to demonstrate the usefulness of the results. In the future, we will generalize some of the results presented in this paper to dense clocks, along the recent work of [8].

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.

References

[1] R. Alur. Timed automata. In *Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer, Berlin, 1999.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

[3] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, January 1994.

[4] A. Bouajjani, R. Echahed, and R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems*

II, volume 999 of *Lecture Notes in Computer Science*, pages 64–85. Springer, Berlin, 1995.

[5] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model-checking. In *Concurrency (CONCUR'97)*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, Berlin, 1997.

[6] A. Coen-Porisini, C. Ghezzi, and R. A. Kemmerer. Specification of realtime systems using ASTRAL. *IEEE Transactions on Software Engineering*, 23(9):572–598, September 1997.

[7] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Concurrency (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257. Springer, Berlin, 1999.

[8] Zhe Dang. Binary reachability analysis of pushdown timed automata with dense clocks. In *Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 506–517. Springer, Berlin, 2001.

[9] Zhe Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer, and J. Su. Binary reachability analysis of discrete pushdown timed automata. In *Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 69–84. Springer, Berlin, 2000.

[10] Zhe Dang and R. A. Kemmerer. Three approximation techniques for astral symbolic model checking of infinite state real-time systems. In *Proceedings of the 22nd international conference on Software engineering*, pages 345–354. ACM Press, New York, 2000.

[11] C. Heitmeyer and N. A. Lynch. The generalized railroad crossing — a case study in formal verification of real-time systems. In *Proceedings 15th IEEE Real-Time Systems Symposium*, pages 120–131. IEEE Computer Society Press, Los Alamitos, 1994.

[12] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, January 1978.

[13] O. H. Ibarra, J. Su, Zhe Dang, T. Bultan, and R. A. Kemmerer. Counter machines and verification problems. *Theoretical Computer Science*, 289(1):165–189, October 2002.

[14] P. Z. Kolano. Phd. dissertation. *Department of Computer Science, University of California at Santa Barbara*, 1999.

[15] P. Z. Kolano, Zhe Dang, and R. A. Kemmerer. The design and analysis of real-time systems using the ASTRAL Software Development Environment. *Annals of Software Engineering*, 7:177–210, 1999.

[16] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, Los Alamitos, 1977.

[17] H. Straubing. *Finite automata, formal logic, and circuit complexity*. Birkhauser, Boston, 1994.

[18] W. Thomas. Auotmata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B*, pages 133–191. Elsevier Science Publishers B. V., Amsterdam, 1990.