WHAT'S DECIDABLE ABOUT HYBRID AUTOMATA

by

Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and
Pravin Varaiya

Memorandum No. UCB/ERL M98/22

15 April 1998

WHAT'S DECIDABLE ABOUT HYBRID AUTOMATA

by

Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and
Pravin Varaiya

Memorandum No. UCB/ERL M98/22

15 April 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

What's Decidable About Hybrid Automata?[*][†]

Thomas A. Henzinger[‡] Peter W. Kopke[§] Anuj Puri[†] Pravin Varaiya[†]

Abstract. Hybrid automata model systems with both digital and analog components, such as embedded control programs. Many verification tasks for such programs can be expressed as reachability problems for hybrid automata. By improving on previous decidability and undecidability results, we identify a precise boundary between decidability and undecidability for the reachability problem of hybrid automata.

On the positive side, we give an (optimal) PSPACE reachability algorithm for the case of initialized rectangular automata, where all analog variables follow independent trajectories within piecewise-linear envelopes and are reinitialized whenever the envelope changes. Our algorithm is based on the construction of a timed automaton that contains all reachability information about a given initialized rectangular automaton. The translation has practical significance for verification, because it guarantees the termination of symbolic procedures for the reachability analysis of initialized rectangular automata. The translation also preserves the ω-languages of initialized rectangular automata with bounded nondeterminism.

On the negative side, we show that several slight generalizations of initialized rectangular automata lead to an undecidable reachability problem. In particular, we prove that the reachability problem is undecidable for timed automata augmented with a single stopwatch.

1 Introduction

A hybrid automaton [ACHH93, NOSY93] combines the discrete dynamics of a finite automaton with the continuous dynamics of a dynamical system. Hybrid automata thus provide a mathematical model for digital computer systems that interact with an analog environment in real time. Case studies indicate that the model of hybrid automata is useful for the analysis of embedded software and hardware, including distributed processes with drifting clocks, real-time schedulers, and protocols for the control of manufacturing plants, vehicles, and robots (see, for example, [HRP94, ACH$^+$95, HHWT95, HW95, NS95, AHH96, Cor96, HWT96, SMF97]). Two problems

[*] A preliminary version of this paper appeared in the *Proceedings of the 27th Annual ACM Symposium on Theory of Computing* (STOC 1995), pp. 373–382.

[†] This research was supported in part by the Office of Naval Research Young Investigator award N00014-95-1-0520, by the National Science Foundation CAREER award CCR-9501708, by the National Science Foundation grant CCR-9504469, by the Air Force Office of Scientific Research contract F49620-93-1-0056, by the Army Research Office MURI grant DAAH-04-96-1-0341, by the Army Research Office contract DAAH04-94-G-0026, by the Advanced Research Projects Agency grant NAG2-892, and by the California PATH program.

[‡] Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

[§] Computer Science Department, Cornell University, Ithaca, NY 14853.

that are central to the analysis of hybrid automata are the reachability problem and the more general ω-language emptiness problem. The solution of the reachability problem for a given hybrid automaton allows us to check if the trajectories of the automaton meet a given safety requirement; the solution to the ω-language emptiness problem allows us to check if the trajectories of the automaton meet a liveness requirement [VW86]. While a scattering of previous results show that both problems are decidable in certain special cases, and undecidable in certain general cases, this paper attempts a systematic characterization of the boundary between decidability and undecidability.

Hybrid automata generalize timed automata. Timed automata [AD94] equip finite automata with clocks, which are real-valued variables that follow continuous trajectories with constant slope 1. Hybrid automata equip finite automata with real-valued variables whose trajectories follow more general dynamical laws. For each class of dynamical laws, we obtain a class of hybrid automata. A particularly interesting class of dynamical laws confines the set of possible trajectories to piecewise-linear envelopes. Suppose, for example, that the variable x represents the water level in a tank. Depending on the position of a control valve (i.e., the state of a finite control automaton), the water level either falls nondeterministically at any rate between 2 and 4 cm s^{-1}, or rises at any rate between 1 and 3 cm s^{-1}. We model these two situations by the dynamical laws $\dot{x} \in [-4,-2]$ and $\dot{x} \in [1,3]$—so-called rectangular flow constraints [PV94]—which enforce piecewise-linear envelopes on the water-level trajectories. Rectangular-flow automata are interesting from a practical point of view, as they permit the modeling of clocks with bounded drift and the conservative approximation of arbitrary trajectory sets [OSY94, HH95, PBV96], and from a theoretical point of view, as they lie at the boundary of decidability.

Our results are threefold. First, we give an (optimal) PSPACE algorithm for the reachability problem of rectangular-flow automata with two restrictions: (1) the values of two variables with different flow constraints are never compared; (2) whenever the flow constraint of a variable changes, the value of the variable is reinitialized. Second, under the additional assumption of bounded nondeterminism (which requires that the successor of a bounded region be bounded), we obtain a PSPACE algorithm for checking ω-language emptiness of rectangular-flow automata. Third, we prove that the reachability problem becomes undecidable if either one of the restrictions (1) and (2) is relaxed, or if more general, triangular flow constraints are admitted.

The first two results are proven by translating rectangular-flow automata of dimension n into timed automata of dimension $2n+1$, where the dimension is the number of real-valued variables. The translation preserves finite languages, and in the case of bounded nondeterminism, also ω-languages. In addition, the translation implies that, when applied to rectangular-flow automata that meet restrictions (1) and (2), existing semidecision procedures for the reachability problem of hybrid automata terminate. Such procedures have been implemented in the HyTech verification tool [AHH96, HHWT97].

The third result is proven by reduction from the halting problem for two-counter machines. In an attempt to characterize the undecidability frontier, we sharpen the reduction as much as possible. First, we prove that any relaxation of restriction (1) leads to the undecidability of the reachability problem for timed automata augmented with a single constant-slope variable whose slope is different from 1. Second, we prove that any relaxation of restriction (2) leads to the undecidability of the reachability problem for timed automata augmented with a single uninitialized two-slope variable, such as a stopwatch, which is a variable whose slope is always either 0 or 1.

Previous work. Over the past few years, there have been many decidability and undecidability results about hybrid systems; we list only those that led to the present work. The first decidability result for hybrid automata was obtained for timed automata, whose reachability and ω-language emptiness problems are PSPACE-complete [AD94]. Under restrictions (1) and (2), that result was later generalized to automata with variables that run at any constant positive slopes [ACH$^+$95], and to the reachability problem for automata with nonstrict rectangular flow constraints [PV94]. In [BES93, KPSY93, BER94, MV94, BR95, ACH97], it was shown that, under various strong side conditions, reachability is decidable for timed automata with one stopwatch, but the general problem of one-stopwatch automata was left open. As far as undecidability results are concerned, in [Cer92] it was shown that reachability is undecidable for timed automata with three stopwatches, as well as for timed automata with one memory cell (a variable of constant slope 0) and assignments between variables. It was also known that reachability is undecidable for timed automata with six memory cells and no assignments [AHV93], for timed automata with two three-slope variables and restriction (1) [KPSY93], and for timed automata with two nonclock constant-slope variables [ACH$^+$95].

2 Rectangular Automata

A hybrid automaton of dimension n is an infinite-state machine whose state has a discrete part, which ranges over the vertices of a graph, and a continuous part, which ranges over the n-dimensional euclidean space $\mathbb{R}^n$ [ACH$^+$95]. A run of a hybrid automaton is a sequence of edge steps and time steps. During an edge step (also called *jump*), the discrete and continuous states are updated according to a guarded command. During a time step (also called *flow*), the discrete state remains unchanged, and the continuous state changes according to a dynamical law, say, a differential equation. In this paper, we are concerned with decidability questions about hybrid automata, and therefore consider restricted classes of guarded commands and dynamical laws. This leads us to the definition of rectangular automata.

Notation. We use the symbol $\mathbb{R}_{\geq 0}$ to denote the set $\{x \in \mathbb{R} \mid x \geq 0\}$ of the nonnegative reals. We use the boldface characters $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ for vectors in $\mathbb{R}^n$, and subscripts on italic characters such as x_i, y_j, and z_k for components of vectors.

Rectangular regions

Given a positive integer $n > 0$, a subset of $\mathbb{R}^n$ is called a *region.* A closed and bounded region is *compact.* A region $R \subset \mathbb{R}^n$ is *rectangular* if it is a cartesian product of (possibly unbounded) intervals, all of whose finite endpoints are rational. We write R_i for the projection of R on the i-th coordinate, so that $R = \prod_{i=1}^{n} R_i$. The set of all rectangular regions in $\mathbb{R}^n$ is denoted $\mathcal{R}^n$.

Definition of rectangular automata

An *n-dimensional rectangular automaton* A consists of a directed multigraph (V, E), a finite *observation alphabet* Σ, three vertex labeling functions $init\colon V \to \mathcal{R}^n$, $inv\colon V \to \mathcal{R}^n$, and $flow\colon V \to \mathcal{R}^n$, and four edge labeling functions $pre : E \to \mathcal{R}^n$, $post : E \to \mathcal{R}^n$, $jump : E \to 2^{\{1,\ldots,n\}}$, and $obs\colon E \to \Sigma$. An *n-dimensional rectangular automaton with ε moves* differs in that the function *obs* maps E into Σ^ε, where $\Sigma^\varepsilon = \Sigma \cup \{\varepsilon\}$ augments the observation alphabet with the *null observation $\varepsilon \notin \Sigma$*. When we discuss more than one automaton, we use the subscript A to identify the components of A. For example, the vertex set of A may be denoted V_A.

The *initial function init* specifies a set of initial automaton states. When the discrete state begins at vertex v, the continuous state must begin in the initial region $init(v)$. The *preguard function pre*, the *postguard function post*, and the *jump function jump* constrain the behavior of the automaton state during edge steps. The edge $e = (v, w)$ may be traversed only if the discrete state resides at vertex v and the continuous state lies in the preguard region $pre(e)$. For each i in the jump set $jump(e)$, the i-th coordinate of the continuous state is nondeterministically assigned a new value in the postguard interval $post(e)_i$. For each $i \notin jump(e)$, the i-th coordinate of the continuous state is not changed and must lie in $post(e)_i$. The *observation function obs* identifies every edge traversal with an observation from Σ or Σ^{ε}. The *invariant function inv* and the *flow function flow* constrain the behavior of the automaton state during time steps. While the discrete state resides at vertex v, the continuous state nondeterministically follows a smooth (C^{∞}) trajectory within the invariant region $inv(v)$, whose first time derivative remains within the flow region $flow(v)$. A rectangular automaton with ε moves may traverse ε edges during time steps.

Note that if we replace rectangular regions with arbitrary linear regions in the definition of rectangular automata, we obtain the linear hybrid automata of [AHH96]. Thus rectangular automata are the subclass of linear hybrid automata in which all defining regions are rectangular.

Initialization and bounded nondeterminism

The rectangular automaton A is *initialized* if for every edge $e = (v, w)$ of A, and every coordinate $i \in \{1, \ldots, n\}$ with $flow(v)_i \neq flow(w)_i$, we have $i \in jump(e)$. It follows that whenever the i-th continuous coordinate of an initialized automaton changes its dynamics, as given by the flow function, then its value is nondeterministically reinitialized according to the postguard function.

The rectangular automaton A has *bounded nondeterminism* if (1) for every vertex $v \in V$, the regions $init(v)$ and $flow(v)$ are bounded, and (2) for every edge $e \in E$, and every coordinate $i \in \{1, \ldots, n\}$ with $i \in jump(e)$, the interval $post(e)_i$ is bounded. Note that bounded nondeterminism does not imply finite branching. It ensures that the edge and time successors of a bounded region are bounded.

The labeled transition system of a rectangular automaton

The rectangular automaton A, possibly with ε moves, defines a labeled transition system with an infinite state space Q, the infinite set $\Sigma \cup \mathbb{R}_{\geq 0}$ of labels, and the binary transition relations $\xrightarrow{\tau}$ on Q, one for each label $\tau \in \Sigma \cup \mathbb{R}_{\geq 0}$. Each transition with label $\sigma \in \Sigma$ corresponds to an edge step whose observation is σ. Each transition with label $t \in \mathbb{R}_{\geq 0}$ corresponds to a time step of duration t. The states and the transitions of A are defined formally as follows.

States. A *state* $(v, \mathbf{x})$ of A consists of a discrete part $v \in V$ and a continuous part $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{x} \in inv(v)$. The *state space* $Q \subset V \times \mathbb{R}^n$ of A is the set of all states of A. A subset of Q is called a *zone* of A. Each zone $Z \subset Q$ can be uniquely decomposed into a collection $\bigcup_{v \in V} \{v\} \times [Z]^v$ of regions $[Z]^v \subset \mathbb{R}^n$, one for each vertex $v \in V$. The zone Z is *rectangular* (resp. *bounded*; *compact*), if each region $[Z]^v$ is rectangular (resp. bounded; compact). The state $(v, \mathbf{x})$ is an *initial state* of A if $\mathbf{x} \in init(v)$. The *initial zone* of A, denoted *Init*, is the set of all initial states of A. Notice that both the state space Q and the initial zone *Init* are rectangular.

Jump transitions. For each edge $e = (v, w)$ of A, we define the binary relation $\xrightarrow{e} \subset Q^2$ by $(v, \mathbf{x}) \xrightarrow{e} (w, \mathbf{y})$ iff $\mathbf{x} \in pre(e)$, and $\mathbf{y} \in post(e)$, and for every coordinate $i \in \{1, \ldots, n\}$ with $i \notin jump(e)$, we have $x_i = y_i$. Hence $\mathbf{x}$ and $\mathbf{y}$ differ only at coordinates in the jump set *jump*(*e*). For each observation $\sigma \in \Sigma^\varepsilon$, we define the *edge-step relation* $\xrightarrow{\sigma} \subset Q^2$ by $q \xrightarrow{\sigma} r$ iff $q \xrightarrow{e} r$ for some edge $e \in E$ with $obs(e) = \sigma$.

Flow transitions. For each nonnegative real $t \in \mathbb{R}_{\geq 0}$, we define the binary relation $\xrightarrow{[t]} \subset Q^2$ by $(v,\mathbf{x}) \xrightarrow{[t]} (w,\mathbf{y})$ iff (1) $v = w$ and (2) either $t = 0$ and $\mathbf{x} = \mathbf{y}$, or $t > 0$ and $(\mathbf{y}-\mathbf{x})/t \in \mathit{flow}(v)$. Notice that due to the convexity of rectangular regions, $(v,\mathbf{x}) \xrightarrow{[t]} (w,\mathbf{y})$ iff there is a smooth function $f\colon [0,t] \rightarrow \mathit{inv}(v)$, with first derivative f', such that $f(0) = \mathbf{x}$, and $f(t) = \mathbf{y}$, and for all reals $s \in (0,t)$, we have $f'(s) \in \mathit{flow}(v)$. Hence the continuous state may evolve from $\mathbf{x}$ to $\mathbf{y}$ via any smooth trajectory satisfying the constraints imposed by $\mathit{inv}(v)$ and $\mathit{flow}(v)$. If A does not have ε moves, then we define the *time-step relation* $\xrightarrow{t}$ to be $\xrightarrow{[t]}$. If A has ε moves, then the time-step relation $\xrightarrow{t} \subset Q^2$ is defined by $q \xrightarrow{t} r$ iff there exists an integer $m \geq 1$, nonnegative reals $t_1,\ldots,t_m$, and states $q_1,\ldots,q_{2m-2}$ such that $q \xrightarrow{[t_1]} q_1 \xrightarrow{\varepsilon} q_2 \xrightarrow{[t_2]} q_3 \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} q_{2m-2} \xrightarrow{[t_m]} r$ and $t = \sum_{i=1}^{m} t_i$.

We write $\Pi = \Sigma^\varepsilon \cup \mathbb{R}_{\geq 0} \cup E \cup [\mathbb{R}_{\geq 0}]$ for the set of labels that arise in connection with the automaton A, where $[\mathbb{R}_{\geq 0}] = \{[t] \mid t \in \mathbb{R}_{\geq 0}\}$. Let Z be a zone of A, and let π be a label from Π. We define $Post^\pi(Z) = \{q \in Q \mid \exists r \in Z.\, r \xrightarrow{\pi} q\}$ to be the zone of states that are reachable in one π step from Z, and we define $Post(Z) = \bigcup_{\pi \in \Sigma \cup \mathbb{R}_{\geq 0}} Post^\pi(Z)$ to be the zone of states that are reachable in one edge or time step from Z. Similarly, we define $Pre^\pi(Z) = \{q \in Q \mid \exists r \in Z.\, q \xrightarrow{\pi} r\}$ to be the zone of states from which Z is reachable in one π step, and we define $Pre(Z) = \bigcup_{\pi \in \Sigma \cup \mathbb{R}_{\geq 0}} Pre^\pi(Z)$ to be the zone of states from which Z is reachable in one edge or time step. Notice that $Post(Z) \supset Z$ and $Pre(Z) \supset Z$ because of time steps of the form $\xrightarrow{0}$.

The reverse automaton

For an n-dimensional rectangular automaton A, the *reverse automaton* $-A$ is an n-dimensional rectangular automaton that defines the same state space as A, but with the transition relations reversed. The vertex set, observation alphabet, initial and invariant functions of $-A$ are the same as for A. For each vertex v, the flow region of $-A$ is defined by $\mathit{flow}_{-A}(v) = \{\mathbf{x} \in \mathbb{R}^n \mid -\mathbf{x} \in \mathit{flow}_A(v)\}$. For each edge $e = (v,w)$ of A, the reverse automaton $-A$ has the edge $-e = (w,v)$ with $\mathit{pre}_{-A}(-e) = \mathit{post}_A(e)$, $\mathit{jump}_{-A}(-e) = \mathit{jump}_A(e)$, and $\mathit{post}_{-A}(-e) = \mathit{pre}_A(e)$. From these definitions, Proposition 2.1 follows immediately.

Proposition 2.1 *For all states q and r of a rectangular automaton A, and every label $\pi \in \Pi$, we have $q \xrightarrow{\pi}_A r$ iff $r \xrightarrow{\pi}_{-A} q$.*

It follows that for every zone Z of A, and every label $\pi \in \Pi$, $Post^\pi_A(Z) = Pre^\pi_{-A}(Z)$ and $Pre^\pi_A(Z) = Post^\pi_{-A}(Z)$.

Multirectangular zones

A zone Z is *multirectangular* if Z is a finite union of rectangular zones. Multirectangularity is preserved by edge and time steps.

Proposition 2.2 *For every multirectangular zone Z of a rectangular automaton A, and every label $\pi \in \Pi$, the zones $Post^\pi(Z)$ and $Pre^\pi(Z)$ are multirectangular.*

Proof. We give the proof for *Post*; the result for *Pre* then follows from Proposition 2.1. Since each relation $\xrightarrow{\sigma}$ with $\sigma \in \Sigma^\varepsilon$ is a finite union of relations $\xrightarrow{e}$ with $e \in E$, and each relation $\xrightarrow{t}$

with $t \in \mathbb{R}_{\geq 0}$ is a finite composition of relations $\xrightarrow{\varepsilon}$ and $\xrightarrow{[s]}$ with $s \in \mathbb{R}_{\geq 0}$, it suffices to prove the proposition for $\pi \in E \cup [\mathbb{R}_{\geq 0}]$. Call a zone *elementary* if it is of the form $\{v\} \times R$, where R is a rectangular region. Then a zone is multirectangular iff it is a finite union of elementary zones. We show that for every elementary zone $Z = \{v\} \times R$, the successor zone $Post^{\pi}(Z)$ is elementary. If $\pi = (v, w)$ is an edge of A, then $Post^{\pi}(Z) = \{w\} \times S$, where

$$S_i = \begin{cases} R_i \cap pre(\pi)_i \cap post(\pi)_i \cap inv(w)_i, & \text{if } i \notin jump(\pi)_i, \\ post(\pi)_i \cap inv(w)_i, & \text{if } i \in jump(\pi)_i \text{ and } R_i \cap pre(\pi)_i \neq \emptyset, \\ \emptyset, & \text{if } R_i \cap pre(\pi)_i = \emptyset. \end{cases}$$

If $\pi = [0]$, then $Post^{\pi}(Z) = Z$. If $\pi \in [\mathbb{R}_{\geq 0}]$ and $\pi > 0$, then $Post^{\pi}(Z) = \{v\} \times S$, where

$$S_i = inv(v)_i \cap \{^1 \inf(R_i) + \pi \cdot \inf(flow(v)_i), \infty) \cap (-\infty, \sup(R_i) + \pi \cdot \sup(flow(v)_i)\}^2.$$

Here, $\{^1$ stands for [if R_i and $flow(v)_i$ are left-closed, $\{^1$ stands for (if R_i or $flow(v)_i$ are left-open, $\}^2$ stands for] if R_i and $flow(v)_i$ are right-closed, and $\}^2$ stands for) if R_i or $flow(v)_i$ are right-open. ■

Given a zone Z of the rectangular automaton A, and a finite sequence $\pi_0\pi_1 \cdots \pi_m \in \Pi^*$ of labels, we define $Post^{\pi_0 \cdot \pi_1}(Z) = Post^{\pi_1}(Post^{\pi_0}(Z))$, and we define $Post^{\pi_0\pi_1 \cdots \pi_m}(Z)$ inductively in the usual way. Also, we define $Post^*(Z) = \bigcup_{i \in \mathbb{N}} Post^i(Z)$ to be the zone of states that are reachable from Z in a finite number of edge and time steps. Analogous definitions are made for *Pre*. A state q is a *reachable state* of A if $q \in Post^*(Init)$. The *reachable zone* of A, denoted $Reach(A)$, is the set $Post^*(Init)$ of all reachable states of A. The reachable zone $Reach(A)$ is an infinite union of rectangular zones, and may not be multirectangular.

The ω-language of a rectangular automaton

Let A be a rectangular automaton, possibly with ε moves. A *timed word* for A is a finite or infinite sequence $\overline{\tau} = \tau_0\tau_1\tau_2 \cdots$ of letters from $\Sigma \cup \mathbb{R}_{\geq 0}$; that is, each τ_i is either an observation of A, or a nonnegative real that denotes a duration of time between observations. The timed word $\overline{\tau}$ is *divergent* if $\overline{\tau}$ is infinite and $\sum\{\tau_i \mid i \in \mathbb{N} \text{ and } \tau_i \in \mathbb{R}_{\geq 0}\} = \infty$. A *run* ρ of A is a finite or infinite sequence of the form $q_0 \xrightarrow{\tau_0} q_1 \xrightarrow{\tau_1} q_2 \xrightarrow{\tau_2} \cdots$, where $q_0 \in Init$, and for all $i \geq 0$, we have $q_i \in Q$ and $\tau_i \in \Sigma \cup \mathbb{R}_{\geq 0}$. The run ρ *accepts* the timed word $\overline{\tau} = \tau_0\tau_1\tau_2 \cdots$, and ρ is called *divergent* if $\overline{\tau}$ is divergent. The *ω-language* of A, denoted $Lang(A)$, is the set of all divergent timed words that are accepted by runs of A.

Example

It is often convenient to refer to each coordinate of the continuous state as a *variable*. We use letters from the beginning of the alphabet, such as a, b, c, d, for variables. If the variable a corresponds to the i-th coordinate of the continuous state, we write $flow(v)(a)$ for $flow(v)_i$, etc. In pictorial descriptions of rectangular automata, we annotate each vertex with its flow region, and sometimes with its invariant region. For example, if $flow(v)(a) = [3, 5]$, $flow(v)(b) = [4, 4]$, $inv(v)(a) = (1, 7]$, and $inv(v)(b) = (-\infty, 0]$, we write "$\dot{a} \in [3, 5]$", "$\dot{b} = 4$", "$1 < a \leq 7$", and "$b \leq 0$" inside of vertex v. Often however, we give the invariant function in the text and omit it from the figure. Edges are annotated with observations and guarded commands. A guarded command ϕ defines two regions $pre(\phi)$ and $post(\phi)$, and a jump set $jump(\phi)$, in a natural manner. For example, if ϕ is the (nondeterministic) guarded command

$$a \leq 5 \wedge b = 4 \rightarrow b := 7;\ c :\in [0, 5]$$

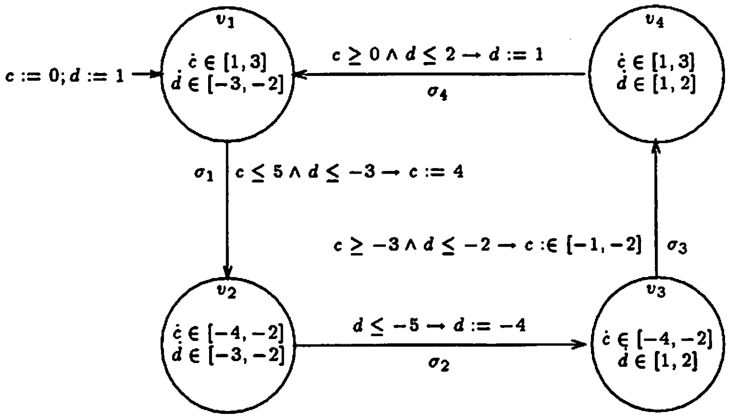

Figure 1: The initialized rectangular automaton $\hat{A}$

then $pre(\phi)(a) = (-\infty, 5]$, $pre(\phi)(b) = [4,4]$, $pre(\phi)(c) = (-\infty, \infty)$, $jump(\phi) = \{b, c\}$, $post(\phi)(a) = (-\infty, \infty)$, $post(\phi)(b) = [7,7]$, and $post(\phi)(c) = [0,5]$. As usual, when writing guarded commands, the guard *true* is omitted, and so is the empty list of assignments.

Consider, for instance, the 2D rectangular automaton $\hat{A}$ of Figure 1. The observation alphabet of $\hat{A}$ is $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, and the invariant function of $\hat{A}$ is the constant function $\lambda v. [-20, 20]^2$ (not shown in the figure). The automaton $\hat{A}$ is initialized, as the values of the two variables c and d are reinitialized whenever their flow regions change. Figure 2 shows a sample trajectory of $\hat{A}$ from the initial zone $Init_{\hat{A}} = \{(v_1, (0,1))\}$. Each arc is labeled with a vertex giving the discrete state, while the continuous state follows the arc. The discontinuities between the arcs labeled v_2 and v_3 correspond to the jump of variable d from -5 to -4 upon traversal of the edge from v_2 to v_3. The divergent timed word $(4\sigma_1 1\sigma_2 1\sigma_3 1\sigma_4)^\omega$ is an example of a timed word that is accepted by $\hat{A}$. This timed word is accepted by a run with the state sequence

$$((v_1, (0,1))(v_1, (5,-10))(v_2, (4,-10))(v_2, (0,-12.5))(v_3, (0,-4))(v_3, (-3,-2))(v_4, (-1,-2))(v_4, (0,0)))^\omega.$$

CNF edge families

We sometimes annotate edges of rectangular automata with positive boolean combinations of guarded commands. Consider the two guarded commands ϕ_1 and ϕ_2. First, the edge annotation $\phi_1 \wedge \phi_2$ stands for a guarded command ϕ_3 with $pre(\phi_3) = pre(\phi_1) \cap pre(\phi_2)$, $post(\phi_3) = post(\phi_1) \cap post(\phi_2)$, and $jump(\phi_3) = jump(\phi_1) \cup jump(\phi_2)$. Second, an edge with the annotation $\phi_1 \vee \phi_2$ stands for two edges that share source vertex, target vertex, and observation; one annotated with ϕ_1 and the other with ϕ_2. These conventions generalize to DNF expressions of guarded commands. An edge annotated with a CNF expression of guarded commands is interpreted by first converting the expression into DNF. A *CNF edge family* $((v,w), \sigma, \psi)$, then, consists of a pair (v,w) of vertices, an observation σ, and a CNF expression ψ of guarded commands. Consider, for example, the CNF edge family with the vertex pair (v,w), the observation σ, and the CNF expression

$$((a < 2 \ \rightarrow \ a := 2) \vee (2 \leq a \leq 5)) \ \wedge \ ((b > 7 \ \rightarrow \ b := 7) \vee (4 \leq b \leq 7)).$$

This edge family corresponds to four edges from v to w, each annotated with the observation σ and one of the following guarded commands:

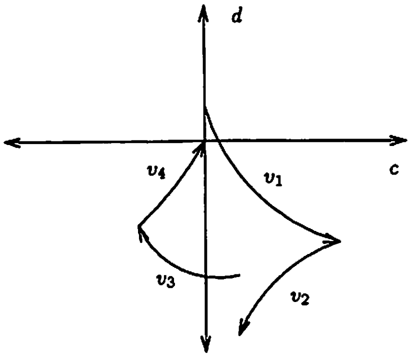

Figure 2: A sample trajectory of $\hat{A}$

1. $a < 2 \wedge b > 7 \rightarrow a := 2;\ b := 7$,
2. $a < 2 \wedge 4 \leq b \leq 7 \rightarrow a := 2$,
3. $2 \leq a \leq 5 \wedge b > 7 \rightarrow b := 7$,
4. $2 \leq a \leq 5 \wedge 4 \leq b \leq 7$.

In this way, an n-dimensional rectangular automaton may be specified by a set of vertices, an observation alphabet, initial, invariant, and flow functions, and a set of CNF edge families. If Z is a zone of the rectangular automaton A, and Ψ is a CNF edge family, we define $Post^{\Psi}(Z)$ to be $\bigcup_e Post^e(Z)$, where the union is taken over all edges e of A that correspond to the edge family Ψ.

Two problems concerning rectangular automata

We study the following two problems about rectangular automata.

Reachability. Given a rectangular automaton A, and a rectangular zone Z_f of A, is $Reach(A) \cap Z_f$ nonempty? That is, does Z_f contain a reachable state of A? If so, we say that the zone Z_f is *reachable* for A. A solution to this problem permits the verification of safety requirements for systems that are modeled as rectangular automata. If we equip rectangular automata with rectangular final zones, then the reachability problem is equivalent to the finitary language emptiness problem.

ω-language emptiness. Given a rectangular automaton A, is $Lang(A)$ nonempty? That is, does A have a divergent run? This problem is more general than the reachability problem, and a solution permits the verification of safety and liveness requirements for systems that are modeled as rectangular automata.

For initialized rectangular automata, we provide a PSPACE decision procedure for the reachability problem. For initialized rectangular automata with bounded nondeterminism, we give a PSPACE decision procedure for the ω-language emptiness problem. We then show that the reachability problem (and therefore ω-language emptiness) is undecidable for very restricted classes of uninitialized rectangular automata, and also for initialized automata with slightly generalized invariant, flow, preguard, postguard, or jump functions.

3 Decidability

We translate a given initialized rectangular automaton A into a timed automaton [AD94] that contains all reachability information about A. The translation proceeds in two steps: from initialized rectangular automata to initialized singular automata (Section 3.2), and from initialized singular automata to timed automata (Section 3.1). For the subclass of automata with bounded nondeterminism, the translation also preserves ω-languages (Section 3.3), and therefore reduces the ω-language emptiness problem for these automata to the corresponding problem for timed automata. In Section 3.4, we explain our translations in terms of simulations and bisimulations of the underlying labeled transition systems. In Section 3.5, we supply a practical implication of our translations, showing that symbolic execution [ACH+95] terminates on initialized rectangular automata after a linear preprocessing step.

3.1 From Initialized Singular Automata To Timed Automata

We begin by defining several special cases of rectangular automata for which, using known results about timed automata, the reachability and ω-language emptiness problems can be solved easily.

Finite-slope variables

Consider a rectangular automaton A. The variable c is a *one-slope variable* of A if there exists a rational number k such that for each vertex v of A, we have $\mathit{flow}(v)(c) = [k,k]$. A one-slope variable with slope $k = 0$ is called a *memory cell.* A one-slope variable with slope $k = 1$ is called a *clock.* A one-slope variable with any other slope is called a *skewed clock.* Notice that, if every variable of A is a one-slope variable, then A is initialized. The variable c is a *two-slope variable* of A if there exist two rational numbers k_1 and k_2, with $k_1 \neq k_2$, such that for each vertex v of A, either $\mathit{flow}(v)(c) = [k_1,k_1]$ or $\mathit{flow}(v)(c) = [k_2,k_2]$. A *stopwatch* is a two-slope variable with $k_1 = 1$ and $k_2 = 0$. The variable c is a *finite-slope variable* of A if for each vertex v of A, the interval $\mathit{flow}(v)(c)$ is a singleton.

A n-dimensional rectangular automaton A has *deterministic jumps* if (1) for every vertex v of A, the region $\mathit{init}(v)$ is either empty or a singleton, and (2) for every edge e of A, and every coordinate $i \in \{1,\ldots,n\}$ with $i \in \mathit{jump}(e)$, the interval $\mathit{post}(e)_i$ is a singleton. The first requirement ensures that the number of initial states is finite. The second requirement says that along each edge step, each variable either remains unchanged or is deterministically assigned a new value. Notice that, if A has deterministic jumps and every variable of A is a finite-slope variable, then A has bounded nondeterminism.

Timed automata

A *timed automaton* D is a rectangular automaton with deterministic jumps such that every variable of D is a clock.

Theorem 3.1 [AD94] *The reachability and ω-language emptiness problems for timed automata (with or without ε moves) are complete for PSPACE.*

More precisely, the ω-language emptiness problem for an n-dimensional timed automaton D with ε moves can be solved in space $O(\log(|V| \cdot n! \cdot K^n)) = O(\log|V| + n \cdot (\log n + \log K))$, where the integer constant K is determined by the rational numbers that are used in the definition of D, as finite endpoints of initial, invariant, preguard, and postguard intervals. If the definition of D uses only nonnegative integer constants, then K is the largest of these constants. If the definition of D uses only nonnegative rational constants, let $\mathcal{K}$ be the set of these constants, and let l be their least common denominator. Then $K = \max\{k \cdot l \mid k \in \mathcal{K}\}$. If the definition of D uses negative rational constants, then subtract the constant with the least value from each of the other constants, and compute K from the resulting set of nonnegative numbers as in the nonnegative case. We assume that, in the definition of D, all constants are written in binary notation. It follows that the value of K is at most singly exponential in the size of the definition of D, and hence the above space requirement is polynomial. The reachability problem for a timed automaton D and a rectangular zone Z can be solved in the same amount of space, only that the constant K must take into account also the finite endpoints of all intervals in the definition of Z.

We consider generalizations of timed automata. Therefore, all of our PSPACE hardness results follow from the corresponding hardness results for timed automata.

Stopwatch automata

A *stopwatch automaton* C is a rectangular automaton with deterministic jumps such that every variable of C is a stopwatch. Unlike timed automata, not every stopwatch automaton is initialized. We will see that for nonintialized stopwatch automata, the reachability problem is undecidable (Section 4.1). Initialized stopwatch automata, however, can be polynomially encoded by timed automata.

Let C be an n-dimensional initialized stopwatch automaton with ε moves, let $\mathcal{K}_C$ be the set of rational constants used in the definition of C, and let $\mathcal{K}_\perp = \mathcal{K}_C \cup \{\perp\}$. We define an n-dimensional timed automaton D_C with the set $V_{D_C} = V_C \times \mathcal{K}_\perp^{\{1,\ldots,n\}}$ of vertices. Thus, each vertex (v, f) of D_C consists of a vertex v from C and a function f from $\{1,\ldots,n\}$ to $\mathcal{K}_\perp$. Each state $q = ((v, f), \mathbf{x})$ of D_C is intended to represent the state $\alpha(q) = (v, \mathbf{y})$ of C, where $y_i = x_i$ if $f(i) = \perp$, and $y_i = f(i)$ if $f(i) \neq \perp$. Intuitively, if the i-th stopwatch of C is running (slope 1), then its value is tracked by the value of the i-th clock of D_C; if the i-th stopwatch is halted (slope 0), at value $k \in \mathcal{K}_C$, then this value is remembered by the vertices of D_C. Note that $\alpha\colon Q_{D_C} \to Q_C$ is an onto function from the states of D_C to the states of C. It is extended to zones $Z \subset Q_{D_C}$ in the natural way, by $\alpha(Z) = \bigcup_{q\in Z} \alpha(q)$.

It is useful to define also an "inverse" of the function α. We define $-\alpha : Q_C \to Q_{D_C}$ so that $-\alpha(v, \mathbf{y}) = ((v, f), \mathbf{y})$, where for every coordinate $i \in \{1,\ldots,n\}$, if $\mathit{flow}_C(v)_i = [1,1]$ then $f(i) = \perp$, and if $\mathit{flow}_C(v)_i = [0,0]$ then $f(i) = y_i$. The function $-\alpha$ is a one-to-one function from the states of C to the states of D_C, so that for each state r of C, we have $\alpha(-\alpha(r)) = r$. The map $-\alpha$ is extended to zones $Z \subset Q_C$ in the natural way. Then, for each zone Z of C, we have $\alpha(-\alpha(Z)) = Z$.

The remaining components of the timed automaton D_C are defined as follows. The observation alphabet of D_C is the same as for C. The initial zone of D_C is $-\alpha(\mathit{Init}_C)$. For each vertex (v, f) of D_C, $\mathit{inv}_{D_C}(v, f) = \mathit{inv}_C(v)$. For each edge $e = (v, w)$ of C, the timed automaton D_C has all edges of the form $e' = ((v, f), (w, g))$, where for every coordinate $i \in \{1,\ldots,n\}$, either $\mathit{flow}_C(w)_i = [1,1]$ and $g(i) = \perp$, or $\mathit{flow}_C(v)_i = [1,1]$ and $\mathit{flow}_C(w)_i = [0,0]$ and $g(i) = \mathit{post}_C(e)_i$, or $\mathit{flow}_C(v)_i = [0,0]$ and $\mathit{flow}_C(w)_i = [0,0]$ and $g(i) = f(i)$. Furthermore, e and e' have the same preguard and postguard regions, jump sets, and observations. From these definitions, Lemma 3.2 follows immediately.

Lemma 3.2 *Let C be an initialized stopwatch automaton with ε moves. First, $\mathit{Init}_C = \alpha(\mathit{Init}_{D_C})$. Second, for all states q and r of D_C, and every label $\tau \in \Sigma^\varepsilon \cup [\mathbb{R}_{\geq 0}]$, if $q \xrightarrow{\tau}_{D_C} r$, then $\alpha(q) \xrightarrow{\tau}_C \alpha(r)$. Third, for all states q of D_C, all states r' of C, and every label $\tau \in \Sigma^\varepsilon \cup [\mathbb{R}_{\geq 0}]$, if $\alpha(q) \xrightarrow{\tau}_{D_C} r'$, then there is a state r of D_C such that $q \xrightarrow{\tau}_{D_C} r$ and $\alpha(r) = r'$; similarly, if $r' \xrightarrow{\tau}_{D_C} \alpha(q)$, then there is a state r of D_C such that $r \xrightarrow{\tau}_{D_C} q$ and $\alpha(r) = r'$.*

$$\begin{tikzcd}
Init_{D_C} \arrow[d, shift left, "\alpha"] & & \cdots & \bullet \arrow[d, "\alpha"] & \bullet \arrow[l, "Pre_{D_C}"'] \arrow[d, "\alpha"] & \bullet \arrow[l, "Pre_{D_C}"'] \arrow[r, "Post_{D_C}"] \arrow[d, shift left, "\alpha"] & \bullet \arrow[r, "Post_{D_C}"] \arrow[d, "\alpha"] & \bullet \arrow[d, "\alpha"] & \cdots \\
Init_C \arrow[u, shift left, "-\alpha"] & & \cdots & \bullet & \bullet \arrow[l, "Pre_C"] & Z \arrow[l, "Pre_C"] \arrow[r, "Post_C"'] \arrow[u, shift left, "-\alpha"] & \bullet \arrow[r, "Post_C"'] & \bullet & \cdots
\end{tikzcd}$$

Figure 3: Commuting diagram for zones Z of an initialized stopwatch automaton C

From the second and third claims of Lemma 3.2, it follows that for every zone Z of D_C, and every label $\tau \in \Sigma \cup \mathbb{R}_{\geq 0}$, $Post_C^\tau(\alpha(Z)) = \alpha(Post_{D_C}^\tau(Z))$ and $Pre_C^\tau(\alpha(Z)) = \alpha(Pre_{D_C}^\tau(Z))$. These relationships are illustrated in the commuting diagram of Figure 3. From the first claim of Lemma 3.2 and the fact that *Post* commutes with α, we conclude that $Reach(C) = \alpha(Reach(D_C))$. From Lemma 3.2 it also follows that $Lang(C) = Lang(D_C)$.

Theorem 3.3 *The reachability and ω-language emptiness problems for initialized stopwatch automata (with or without ε moves) are complete for PSPACE.*

Proof. For containment in PSPACE, notice that $|V_{D_C}| \leq |V_C| \cdot (K_C + 1)^n$, and that the definition of D_C uses the same constants as the definition of C. The space requirement $O(\log |V_{D_C}| + n \cdot (\log n + \log K_{D_C})) = O(\log |V_C| + n \cdot (\log n + \log K_C))$, where K_C is defined as for timed automata, is polynomial in the size of C. ■

Singular automata

A *singular automaton* B is a rectangular automaton with deterministic jumps such that every variable of B is a finite-slope variable. Initialized singular automata can be rescaled to initialized stopwatch automata.

Let B be an n-dimensional initialized singular automaton with ε moves. We define an n-dimensional initialized stopwatch automaton C_B with the same vertex set, edge set, and observation alphabet as B. Each state $q = (v, \mathbf{x})$ of C_B is intended to represent the state $\beta(q) = (v, \beta_v(\mathbf{x}))$ of B, for the following definition of the bijections $\beta_v : \mathbb{R}^n \to \mathbb{R}^n$. For each vertex v of B, if $flow_B(v) = \prod_{i=1}^n [k_i, k_i]$, then $\beta_v(x_1, \ldots, x_n) = (l_1 \cdot x_1, \ldots, l_n \cdot x_n)$, where $l_i = k_i$ if $k_i \neq 0$, and $l_i = 1$ if $k_i = 0$. Note that $\beta : Q_{C_B} \to Q_B$ is a bijection between the states of C_B and the states of B, and can be viewed as a rescaling of the state space. Similar rescaling techniques for hybrid state spaces can be found in in [ACH+95]. The maps β_v are extended to regions in the natural way, and the map β is extended to zones in the natural way.

The remaining components of the initialized stopwatch automaton C_B are defined as follows. For each vertex v of C_B, $init_{C_B}(v) = \beta_v^{-1}(init_B(v))$, $inv_{C_B}(v) = \beta_v^{-1}(inv_B(v))$, and for every coordinate $i \in \{1, \ldots, n\}$, $flow_{C_B}(v)_i = [0, 0]$ if $flow_B(v)_i = [0, 0]$, and $flow_{C_B}(v)_i = [1, 1]$ if $flow_B(v)_i \neq [0, 0]$. For each edge $e = (v, w)$ of C_B, $pre_{C_B}(e) = \beta_v^{-1}(pre_B(e))$, $post_{C_B}(e) = \beta_w^{-1}(post_B(e))$, $jump_{C_B}(e) = jump_B(e)$, and $obs_{C_B}(e) = obs_B(e)$. From these definitions, Lemma 3.4 follows immediately.

Lemma 3.4 *Let B be an initialized singular automaton with ε moves. First, $Init_B = \beta(Init_{C_B})$. Second, for all states q and r of C_B, and every label $\tau \in \Sigma^\varepsilon \cup [\mathbb{R}_{\geq 0}]$, we have $q \xrightarrow{\tau}_{C_B} r$ iff $\beta(q) \xrightarrow{\tau}_B \beta(r)$.*

From the second claim of Lemma 3.4, it follows that for every zone Z of C_B, and every label $\tau \in \Sigma \cup \mathbb{R}_{\geq 0}$, $Post_B^\tau(\beta(Z)) = \beta(Post_{C_B}^\tau(Z))$ and $Pre_B^\tau(\beta(Z)) = \beta(Pre_{C_B}^\tau(Z))$. These relationships

$$\begin{tikzcd}
\mathit{Init}_{C_B} \arrow[d, leftrightarrow, "\beta"] & \cdots & \bullet \arrow[d, leftrightarrow, "\beta"] & \bullet \arrow[l, "\mathit{Pre}_{C_B}"'] \arrow[d, leftrightarrow, "\beta"] & \bullet \arrow[l, "\mathit{Pre}_{C_B}"'] \arrow[r, "\mathit{Post}_{C_B}"] \arrow[d, leftrightarrow, "\beta"] & \bullet \arrow[r, "\mathit{Post}_{C_B}"] \arrow[d, leftrightarrow, "\beta"] & \bullet \arrow[d, leftrightarrow, "\beta"] & \cdots \\
\mathit{Init}_B & \cdots & \bullet & \bullet \arrow[l, "\mathit{Pre}_B"] & Z \arrow[l, "\mathit{Pre}_B"] \arrow[r, "\mathit{Post}_B"'] & \bullet \arrow[r, "\mathit{Post}_B"'] & \bullet & \cdots
\end{tikzcd}$$

Figure 4: Commuting diagram for zones Z of an initialized singular automaton B

are illustrated in the commuting diagram of Figure 4. From the first claim of Lemma 3.4 and the fact that *Post* commutes with γ^c, we conclude that $\mathit{Reach}(B) = \beta(\mathit{Reach}(C_B))$. From Lemma 3.4 it also follows that $\mathit{Lang}(B) = \mathit{Lang}(C_B)$.

Theorem 3.5 *The reachability and ω-language emptiness problems for initialized singular automata (with or without ε moves) are complete for PSPACE.*

Proof. It remains to show containment in PSPACE. Consider the case that the definition of B uses only nonnegative integer constants; the general case is similar to the analysis of timed automata with rational and negative constants. As for timed automata, define K_B to be the maximum over all finite endpoints of initial, invariant, preguard, and postguard intervals. Define L_B to be the product of K_B and the least common multiple of all finite endpoints of flow intervals. Then the value of L_B is at most singly exponential in the size of the definition of B, and $K_{C_B} \leq L_B$. It follows that the space requirement $O(\log|V_{C_B}| + n \cdot (\log n + \log K_{C_B})) = O(\log|V_B| + n \cdot (\log n + \log L_B))$ is polynomial in the size of B. ■

3.2 From Initialized Rectangular Automata to Initialized Singular Automata

Let A be an n-dimensional initialized rectangular automaton. We translate A into a $(2n+1)$-dimensional initialized singular automaton B_A with ε moves so that B_A contains all reachability information about A. (While we assume, for simplicity, that the given automaton A has no ε moves, the translation and its results apply also to initialized rectangular automata with ε moves.) The translation is similar to the subset construction for determinizing finite automata. We first give a simplified construction for the compact case, and then proceed to the general case. All of the main ideas of the construction are already present in the compact case, but the general case requires a lot of additional bookkeeping.

Preliminary assumptions. Without loss of generality, we assume that for each vertex v of A, $\mathit{init}_A(v) \subset \mathit{inv}_A(v)$, and for each edge $e = (v, w)$ of A, $\mathit{pre}_A(e) \subset \mathit{inv}_A(v)$, $\mathit{post}_A(e) \subset \mathit{inv}_A(w)$, and for every coordinate $i \notin \mathit{jump}_A(e)$, $\mathit{pre}_A(e)_i = \mathit{post}_A(e)_i$. If this is not the case, then first we replace each initial and guard region by its intersection with the appropriate invariant region, and second we replace each guard interval $\mathit{pre}_A(e)_i$ and $\mathit{post}_A(e)_i$ with $i \notin \mathit{jump}_A(e)$ by their intersection $\mathit{pre}_A(e)_i \cap \mathit{post}_A(e)_i$. These replacements do not change the labeled transition system of A.

The compact case

We first restrict our attention to the case where inv_A is the trivial invariant function $\lambda v.\mathbb{R}^n$, and all initial, flow, preguard, and postguard regions of A are compact. In this case, we say that the automaton A is *compact*.

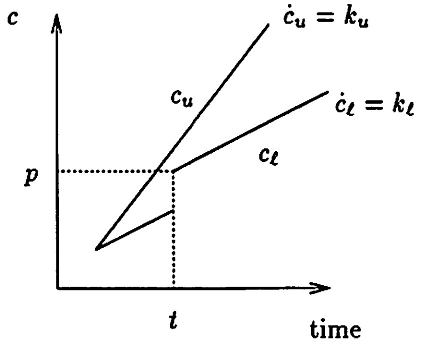

Figure 5: Envelope created by the flow interval $flow(v)(c) = [k_\ell, k_u]$

In the compact case, we translate A into a $2n$-dimensional initialized singular automaton denoted B_A^c. The idea is to replace each variable c of A by two finite-slope variables c_ℓ and c_u such that when $flow_A(v)(c) = [k_\ell, k_u]$, then $flow_{B_A^c}(v)(c_\ell) = [k_\ell, k_\ell]$ and $flow_{B_A^c}(v)(c_u) = [k_u, k_u]$. Intuitively, the variable c_ℓ tracks the least possible value of c, and c_u tracks the greatest possible value of c. Consider Figure 5. With each time step, the flow interval of c creates an envelope, whose boundaries are tracked by c_ℓ and c_u. With each edge step, the values of c_ℓ and c_u are updated so that the interval $[c_\ell, c_u]$ remains precisely the range of possible values of c. In Figure 5, at time t a jump transition is taken along an edge e with $pre_A(e)(c) = [p, \infty)$ and $c \notin jump_A(e)$. Since the value of c_ℓ is below p at time t, the variable c_ℓ is updated to the new value p.

In the following formal definition of the automaton B_A^c, we use $\{a_i \mid 1 \leq i \leq n\}$ for the variables of A, and $\{b_i \mid 1 \leq i \leq 2n\}$ for the variables of B_A^c. The $\ell(i)$-th coordinate $b_{\ell(i)}$ of B_A^c represents the lower bound on the i-th coordinate a_i of A, and the $u(i)$-th coordinate $b_{u(i)}$ of B_A^c represents the upper bound on a_i. For concreteness, put $\ell(i) = 2i - 1$ and $u(i) = 2i$.

The $2n$-dimensional automaton B_A^c has the same vertex set and observation alphabet as A. The initial function $init_{B_A^c}$ is given by $init_{B_A^c}(v)_{\ell(i)} = \min(init_A(v)_i)$ and $init_{B_A^c}(v)_{u(i)} = \max(init_A(v)_i)$. The invariant function $inv_{B_A^c}$ is the trivial invariant function. The flow function $flow_{B_A^c}$ is defined as outlined above: if $flow_A(v)_i = [k_\ell, k_u]$, then $flow_{B_A^c}(v)_{\ell(i)} = [k_\ell, k_\ell]$ and $flow_{B_A^c}(v)_{u(i)} = [k_u, k_u]$. We are left with defining a set of CNF edge families for B_A^c. For each edge $e = (v, w)$ of A, the automaton B_A^c has the CNF edge family $\Psi_e = (v, w, obs_A(e), \psi_e)$, which shares the observation of e. The CNF expression ψ_e is a conjunction $\bigwedge_{i=1}^n \psi_e^i$ of n CNF expressions ψ_e^i. Suppose that $pre_A(e)_i = [p_\ell, p_u]$ and $post_A(e)_i = [p'_\ell, p'_u]$. If $i \in jump_A(e)$, then the CNF expression ψ_e^i is the guarded command

$$b_{\ell(i)} \leq p_u \wedge b_{u(i)} \geq p_\ell \rightarrow b_{\ell(i)} := p'_\ell;\ b_{u(i)} := p'_u.$$

The values of $b_{\ell(i)}$ and $b_{u(i)}$ satisfy the guard iff $[b_{\ell(i)}, b_{u(i)}]$ intersects $pre_A(e)_i$. Since $i \in jump_A(e)$, the range of values of a_i after traversing e is exactly $post_A(e)_i$, and hence $b_{\ell(i)}$ is set to the minimum of this interval, and $b_{u(i)}$ is set to the maximum. If $i \notin jump_A(e)$, then by assumption $[p_\ell, p_u] = [p'_\ell, p'_u]$, and the CNF expression ψ_e^i is

$$((b_{\ell(i)} < p_\ell \rightarrow b_{\ell(i)} := p_\ell) \vee (p_\ell \leq b_{\ell(i)} \leq p_u)) \wedge ((b_{u(i)} > p_u \rightarrow b_{u(i)} := p_u) \vee (p_\ell \leq b_{u(i)} \leq p_u)).$$

The idea is that when the edge e is traversed in A, new information becomes available about the value of a_i, namely, that it lies within the interval $[p_\ell, p_u]$. Therefore, if $b_{\ell(i)} < p_\ell$, it must be updated to p_ℓ, and if $b_{u(i)} > p_u$, it must be updated to p_u, in order to keep $[b_{\ell(i)}, b_{u(i)}]$ the range of possible values of a_i.

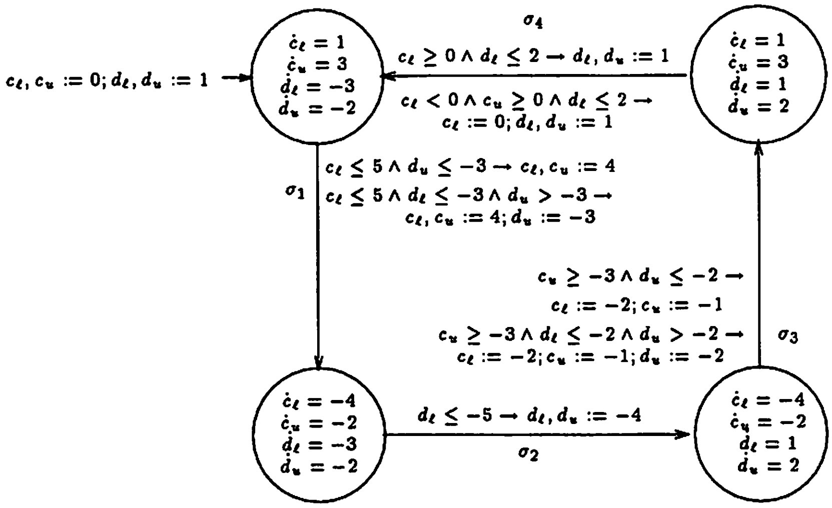
$c_\ell, c_u := 0; d_\ell, d_u := 1$
$\dot{c}_\ell = 1$
$\dot{c}_u = 3$
$\dot{d}_\ell = -3$
$\dot{d}_u = -2$
σ_4
$c_\ell \geq 0 \land d_\ell \leq 2 \to d_\ell, d_u := 1$
$c_\ell < 0 \land c_u \geq 0 \land d_\ell \leq 2 \to$
$c_\ell := 0; d_\ell, d_u := 1$
$\dot{c}_\ell = 1$
$\dot{c}_u = 3$
$\dot{d}_\ell = 1$
$\dot{d}_u = 2$
σ_1
$c_\ell \leq 5 \land d_u \leq -3 \to c_\ell, c_u := 4$
$c_\ell \leq 5 \land d_\ell \leq -3 \land d_u > -3 \to$
$c_\ell, c_u := 4; d_u := -3$
$c_u \geq -3 \land d_u \leq -2 \to$
$c_\ell := -2; c_u := -1$
$c_u \geq -3 \land d_\ell \leq -2 \land d_u > -2 \to$
$c_\ell := -2; c_u := -1; d_u := -2$
σ_3
$\dot{c}_\ell = -4$
$\dot{c}_u = -2$
$\dot{d}_\ell = -3$
$\dot{d}_u = -2$
$d_\ell \leq -5 \to d_\ell, d_u := -4$
σ_2
$\dot{c}_\ell = -4$
$\dot{c}_u = -2$
$\dot{d}_\ell = 1$
$\dot{d}_u = 2$

Figure 6: The initialized singular automaton $B^c_{\hat{A}}$

This completes the definition of the automaton B^c_A. The automaton B^c_A is an initialized singular automaton. Figure 6 shows the initialized singular automaton $B^c_{\hat{A}}$ that corresponds to the initialized rectangular automaton $\hat{A}$ from Figure 1. Figure 7 shows the initialized stopwatch automaton $C_{B^c_{\hat{A}}}$ that corresponds to $\hat{A}$. Notice that $C_{B^c_{\hat{A}}}$ is a timed automaton.

The relationship between A and B^c_A is formalized through the function $\gamma^c : Q_{B^c_A} \to 2^{Q_A}$, which maps each state of B^c_A to a zone of A: define $\gamma^c(v, \mathbf{x}) = \{v\} \times \prod_{i=1}^{n}[x_{\ell(i)}, x_{u(i)}]$. The map γ^c is extended to zones $Z \subset Q_{B^c_A}$ by $\gamma^c(Z) = \bigcup_{q \in Z} \gamma^c(q)$. Not all states of B^c_A are of interest. The state $(v, \mathbf{x}) \in Q_{B^c_A}$ is an *upper-half state* of B^c_A if for every coordinate $i \in \{1, \ldots, n\}$, we have $x_{\ell(i)} \leq x_{u(i)}$. Notice that q is an upper-half state of B^c_A iff $\gamma^c(q) \neq \emptyset$. The *upper-half space* of B^c_A, denoted $U_{B^c_A}$, is the set of all upper-half states of B^c_A. Lemma 3.6 shows that, for reachability purposes, each upper-half state q of B^c_A represents the set $\gamma^c(q) \subset Q_A$ of states of A.

It is useful to define also an "inverse" of the function γ^c. For a compact rectangular region $R \subset \mathbb{R}^n$, define the vector $-\gamma^c(R) \in \mathbb{R}^{2n}$ by $-\gamma^c(R)_{\ell(i)} = \min(R_i)$ and $-\gamma^c(R)_{u(i)} = \max(R_i)$; that is, $-\gamma^c(R)$ is the vector of all boundary points of the intervals R_i. For a compact rectangular zone Z of A, define $-\gamma^c(Z)$ to be the zone $\bigcup_{v \in V_A}\{(v, -\gamma^c([Z]^v))\}$ of B^c_A. Notice that $-\gamma(Z) \subset U_{B^c_A}$, and that $Init_{B^c_A} = -\gamma^c(Init_A)$.

Lemma 3.6 *Let A be a compact initialized rectangular automaton. First, for every compact rectangular zone Z of A, we have $\gamma^c(-\gamma^c(Z)) = Z$; in particular, $Init_A = \gamma^c(Init_{B^c_A})$. Second, for every upper-half state q of B^c_A, and every label $\tau \in \Sigma \cup \mathbb{R}_{\geq 0}$, $Post^\tau_A(\gamma^c(q)) = \gamma^c(Post^\tau_{B^c_A}(q))$.*

We delay a detailed proof of these claims to the general case. From the second claim of Lemma 3.6, it follows that for every upper-half zone $Z \subset U_{B^c_A}$, and every label $\tau \in \Sigma \cup \mathbb{R}_{\geq 0}$, $Post^\tau_A(\gamma^c(Z)) = \gamma^c(Post^\tau_{B^c_A}(Z))$ and, by Proposition 2.1, $Pre^\tau_A(\gamma^c(Z)) = \gamma^c(Pre^\tau_{-B^c_A}(Z))$. These relationships are illustrated in the commuting diagram of Figure 12. From the first claim of Lemma 3.6 and the fact that *Post* commutes with γ^c, we conclude that $Reach(A) = \gamma^c(Reach(B^c_A))$.

Theorem 3.7 *The reachability problem for compact initialized rectangular automata is complete for PSPACE.*

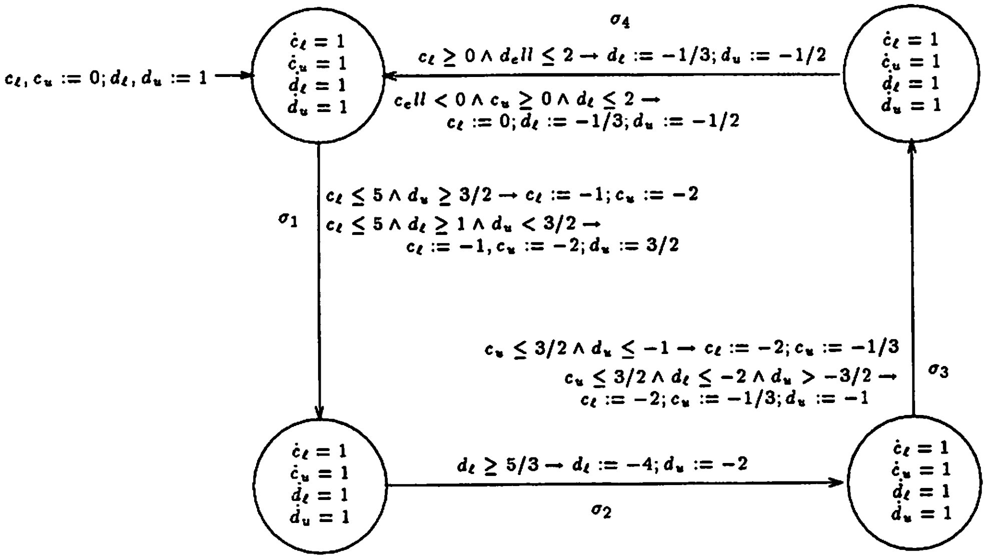
$c_\ell, c_u := 0; d_\ell, d_u := 1$
$\dot{c}_\ell = 1$
$\dot{c}_u = 1$
$\dot{d}_\ell = 1$
$\dot{d}_u = 1$
σ_4
$c_\ell \geq 0 \wedge d_ell \leq 2 \rightarrow d_\ell := -1/3; d_u := -1/2$
$c_ell < 0 \wedge c_u \geq 0 \wedge d_\ell \leq 2 \rightarrow$
$c_\ell := 0; d_\ell := -1/3; d_u := -1/2$
$\dot{c}_\ell = 1$
$\dot{c}_u = 1$
$\dot{d}_\ell = 1$
$\dot{d}_u = 1$
σ_1
$c_\ell \leq 5 \wedge d_u \geq 3/2 \rightarrow c_\ell := -1; c_u := -2$
$c_\ell \leq 5 \wedge d_\ell \geq 1 \wedge d_u < 3/2 \rightarrow$
$c_\ell := -1, c_u := -2; d_u := 3/2$
$c_u \leq 3/2 \wedge d_u \leq -1 \rightarrow c_\ell := -2; c_u := -1/3$
$c_u \leq 3/2 \wedge d_\ell \leq -2 \wedge d_u > -3/2 \rightarrow$
$c_\ell := -2; c_u := -1/3; d_u := -1$
σ_3
$\dot{c}_\ell = 1$
$\dot{c}_u = 1$
$\dot{d}_\ell = 1$
$\dot{d}_u = 1$
$d_\ell \geq 5/3 \rightarrow d_\ell := -4; d_u := -2$
σ_2
$\dot{c}_\ell = 1$
$\dot{c}_u = 1$
$\dot{d}_\ell = 1$
$\dot{d}_u = 1$

Figure 7: The initialized stopwatch automaton (and timed automaton) $C_{B^c_A}$

Indeed, since $Init_A = \gamma^c(Init_{B^c_A})$ and *Post* commutes with γ^c, we also have the following. For every run of A we can, starting from the beginning of the run, inductively construct a run of B^c_A that accepts the same timed word. Therefore $Lang(A) \subset Lang(B^c_A)$. Furthermore, for every finite run of B^c_A we can, starting from the end of the run, inductively construct a run of A that accepts the same timed word. Therefore A and B_A accept the same finite timed words.

The general case

The extension from the compact to the general case is mostly a matter of bookkeeping. In particular, for each lower-bound variable $b_{\ell(i)}$ and upper-bound variable $b_{u(i)}$, one bit is used to distinguish a weak from a strict bound, and a second bit is used to distinguish a finite from an infinite bound. The reader who is uninterested in the details can skip ahead to Theorem 3.18 without loss of continuity (in this case, the reader should know that B_A is the generalization of B^c_A, and γ is the generalization of γ^c).

We encounter the following difficulties when the rectangular automaton A has noncompact components and a nontrivial invariant function. (1) The lower-bound and upper-bound variables $b_{\ell(i)}$ and $b_{u(i)}$ may violate the invariant region of a vertex, and the function γ must be changed to account for this. (2) The lower and upper bounds represented by $b_{\ell(i)}$ and $b_{u(i)}$ may be weak (inclusive) or strict (exclusive). To solve this problem, we introduce a bit called the *weak/strict bit* for each variable of B_A. (3) Lower and upper bounds may be finite or infinite. For this, we introduce a bit called the *finite/infinite bit* for each variable of B_A. (4) Unbounded flow regions cause discontinuous jumps in lower and upper bounds. For example, suppose that the variable a_i is assigned the value 3 by traversal of the edge $e = (v, w)$, where $flow(w)_i = [1, \infty)$. Then in B_A, both $b_{\ell(i)}$ and $b_{u(i)}$ are assigned the value 3 by traversal of e. But after any positive amount of time passes, the upper bound on a_i should be ∞. For this, we introduce an ε edge, which is taken before any positive time step. The ε edge sets the finite/infinite bit for $b_{u(i)}$ to infinite. Since the result of the ε edge presupposes that some positive amount of time has passed, all edge steps inherited from A are disabled in B_A until time passes. Implementing this restriction requires a new clock called

$$\begin{tikzcd}
Init_{B^c_A} \arrow[d, shift left, "\gamma^c"] \\
Init_A \arrow[u, shift left, "-\gamma^c"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdots & \bullet \arrow[d, "\gamma^c"] & \bullet \arrow[l, "Pre_{-B^c_{-A}}"'] \arrow[d, "\gamma^c"] & \bullet \arrow[l, "Pre_{-B^c_{-A}}"'] \arrow[r, "Post_{B^c_A}"] \arrow[d, shift left, "\gamma^c"] & \bullet \arrow[r, "Post_{B^c_A}"] \arrow[d, "\gamma^c"] & \bullet \arrow[d, "\gamma^c"] & \cdots \\
\cdots & \bullet & \bullet \arrow[l, "Pre_A"] & Z \arrow[l, "Pre_A"] \arrow[r, "Post_A"'] \arrow[u, shift left, "-\gamma^c"] & \bullet \arrow[r, "Post_A"'] & \bullet & \cdots
\end{tikzcd}$$

Figure 8: Commuting diagram for compact rectangular zones Z of a compact initialized rectangular automaton A

the *synchronization clock*, and a bit called the *time-passage bit*. (5) Strict flow regions may cause a weak bound to change to strict after any positive amount of time passes. For example, suppose in the above case that $flow(w)_i = [1, 5)$. Then after the edge e is traversed, the upper bound on a_i is a weak bound of 3. But after $t > 0$ time units pass, the upper bound is a strict bound of $3 + 5t$. Once again, we use the ε edge to solve this problem. When the ε edge is traversed, the weak/strict bit for $b_{u(i)}$ is set to strict.

We now proceed formally to define the $(2n+1)$-dimensional initialized singular automaton B_A with ε moves. The observation alphabet of B_A is the same as for A. We define first the continuous state (variables) and discrete state (vertices) of B_A, and then the flow and invariant functions. Next we define the map γ, which relates states of B_A to zones of A. Then come the edges of B_A, which are classified into ε edges and edges inherited from A. Last, we define the initial function. We provide lemmas about B_A as soon as enough definitions have been made to give the proofs.

Variables and vertices of B_A. For each variable a_i of A, the automaton B_A has two variables, $b_{\ell(i)}$ and $b_{u(i)}$. We add a zeroth coordinate b_0 to B_A, which represents the synchronization clock. Hence the dimension of B_A is $2n + 1$. The finite/infinite and weak/strict bitvectors and the time-passage bit are encoded in the vertex set of B_A. Therefore, $V_{B_A} = V_A \times (\{0,1\}^{2n})^2 \times \{0,1\}$. A state $((v, \vec{\mu}, \vec{\nu}, tp), \mathbf{x})$ of B_A, then, consists of a vertex v of A, a vector $\vec{\mu}$ of $2n$ finite/infinite bits ($fin = 0$, $inf = 1$), a vector $\vec{\nu}$ of $2n$ weak/strict bits ($wk = 0$, $str = 1$), a time-passage bit tp, and a vector $\mathbf{x} \in \mathbb{R}^{2n+1}$ of values for the $2n + 1$ variables of B_A.

Notation. For the remainder of Section 3.2, i ranges over $\{1, \ldots, n\}$, the symbol v ranges over V_A, the vectors $\vec{\mu}$ and $\vec{\nu}$ range over $\{0,1\}^{2n}$, and tp ranges over $\{0, 1\}$. So when we implicitly or explicitly quantify these variables, as in "for all $v, \vec{\mu}, \vec{\nu}, tp, i$," the quantification is over the domains just specified. Recall that for a zone Z of A, we have the canonical decomposition $Z = \bigcup_v [Z]^v$. We lift the functions $[\cdot]^v$ to the zones Z' of B_A by defining $[Z']^v = \{\mathbf{x} \in \mathbb{R}^{2n+1} \mid \exists \vec{\mu}, \vec{\nu}, tp.\, ((v, \vec{\mu}, \vec{\nu}, tp), \mathbf{x}) \in Z'\}$. We say that the vector $\mathbf{x} \in \mathbb{R}^{2n+1}$ *satisfies* a predicate φ over the variables $b_0, b_1, \ldots, b_{2n}$ iff φ evaluates to true when each variable b_i is replaced by the i-th component of $\mathbf{x}$. Finally, consider an interval $I \subset \mathbb{R}$ of the real line. We define the *lower strictness* of I by $strict{\downarrow}(I) = wk$ if $\inf(I) \in I$, and $strict{\downarrow}(I) = str$ if $\inf(I) \notin I$. Similarly, we define the *upper strictness* of I by $strict{\uparrow}(I) = wk$ if $\sup(I) \in I$ and $strict{\uparrow}(I) = str$ if $\sup(I) \notin I$.

Flow regions of B_A. The flow function $flow_{B_A}$ is defined by

$$flow_{B_A}(v, \vec{\mu}, \vec{\nu}, tp)_{\ell(i)} = \begin{cases} \{\inf(flow_A(v)_i)\}, & \text{if } \inf(flow_A(v)_i) \neq -\infty, \\ \{0\}, & \text{otherwise;} \end{cases}$$

$$flow_{B_A}(v, \vec{\mu}, \vec{\nu}, tp)_{u(i)} = \begin{cases} \{\sup(flow_A(v)_i)\}, & \text{if } \sup(flow_A(v)_i) \neq \infty, \\ \{0\}, & \text{otherwise.} \end{cases}$$

The slope of $b_{\ell(i)}$ (resp. $b_{u(i)}$) in B_A is the infimum (resp. supremum) of the allowable slopes for a_i in A, unless that infimum (resp. supremum) is infinite. The zeroth coordinate b_0 is a clock: define $\mathit{flow}_{B_A}(v,\vec{\mu},\vec{\nu},tp)_0 = \{1\}$.

Invariant regions of B_A. The lower-bound components of the invariant function inv_{B_A} are defined by

$$\mathit{inv}_{B_A}(v,\vec{\mu},\vec{\nu},tp)_{\ell(i)} = \begin{cases} (-\infty,\infty), & \text{if } \mu_{\ell(i)} = \mathit{inf}, \\ (-\infty,\sup(\mathit{inv}_A(v)_i)], & \text{if } \mu_{\ell(i)} = \mathit{fin} \text{ and } \nu_{\ell(i)} = \mathit{strict}\!\uparrow(\mathit{inv}_A(v)_i) = \mathit{wk}, \\ (-\infty,\sup(\mathit{inv}_A(v)_i)), & \text{otherwise.} \end{cases}$$

If the finite/infinite bit $\mu_{\ell(i)}$ is infinite, then the value of the lower-bound variable $b_{\ell(i)}$ is irrelevant, so we do not constrain it. If the finite/infinite bit is finite, then the interval of allowable values for $b_{\ell(i)}$ is right-open unless both the weak/strict bit $\nu_{\ell(i)}$ is weak and the invariant interval $\mathit{inv}_A(v)_i$ is right-closed. The motivation for this is that if I and J are intervals, and $\inf(I) = \sup(J)$, then $I \cap J \neq \emptyset$ iff $\mathit{strict}\!\downarrow(I) = \mathit{strict}\!\uparrow(J) = \mathit{wk}$. Here I is meant to represent the range of possible values for a_i in A, as determined by the state of B_A, and J is meant to represent the invariant interval $\mathit{inv}_A(v)_i$. The corresponding definition for the upper-bound components of inv_{B_A} is given by

$$\mathit{inv}_{B_A}(v,\vec{\mu},\vec{\nu},tp)_{u(i)} = \begin{cases} (-\infty,\infty), & \text{if } \mu_{u(i)} = \mathit{inf}, \\ [\inf(\mathit{inv}_A(v)_i),\infty), & \text{if } \mu_{u(i)} = \mathit{fin} \text{ and } \nu_{u(i)} = \mathit{strict}\!\downarrow(\mathit{inv}_A(v)_i) = \mathit{wk}, \\ (\inf(\mathit{inv}_A(v)_i),\infty), & \text{otherwise.} \end{cases}$$

The invariant intervals for the synchronization clock b_0 let time pass iff the time-passage bit tp is 1: define $\mathit{inv}_{B_A}(v,\vec{\mu},\vec{\nu},0)_0 = [0,0]$ and $\mathit{inv}_{B_A}(v,\vec{\mu},\vec{\nu},1)_0 = [0,\infty)$.

Relating the state spaces of B_A and A. The function $\gamma' : Q_{B_A} \to 2^{V_A \times \mathbb{R}^n}$ specifies how each state of B_A indicates a range of possible values for each variable of A. Consider a state $q = ((v,\vec{\mu},\vec{\nu},tp),\mathrm{x})$ of B_A. We define $\gamma'(q) = \{v\} \times \gamma'_2(q)$, where the function $\gamma'_2 : Q_{B_A} \to 2^{\mathbb{R}^n}$ maps q to a rectangular region in $\mathbb{R}^n$. For every coordinate $i \in \{1,\ldots,n\}$, the interval $\gamma'_2(q)_i$ is completely specified by its infimum, supremum, and which, if any, of the two it contains.

- If $\mu_{\ell(i)} = \mathit{inf}$, then $\inf(\gamma'_2(q)_i) = -\infty$; if $\mu_{\ell(i)} = \mathit{fin}$, then $\inf(\gamma'_2(q)_i) = x_{\ell(i)}$.
- If $\mu_{u(i)} = \mathit{inf}$, then $\sup(\gamma'_2(q)_i) = \infty$; if $\mu_{u(i)} = \mathit{fin}$, then $\sup(\gamma'_2(q)_i) = x_{u(i)}$.
- If $\mu_{\ell(i)} = \mathit{fin}$, then $\inf(\gamma'_2(q)_i) \in \gamma'_2(q)_i$ iff $\nu_{\ell(i)} = \mathit{wk}$.
- If $\mu_{u(i)} = \mathit{fin}$, then $\sup(\gamma'_2(q)_i) \in \gamma'_2(q)_i$ iff $\nu_{u(i)} = \mathit{wk}$;

Since we have not taken into account the invariant function of A, we may have $\gamma'(q) \not\subset Q_A$ for some states q of B_A. We remedy this deficiency by defining $\gamma : Q_{B_A} \to 2^{Q_A}$ by $\gamma(q) = \gamma'(q) \cap Q_A$ if $tp = 0$ or $x_0 > 0$, and $\gamma(q) = \emptyset$ if $tp = 1$ and $x_0 = 0$. The latter adjustment is necessary because the states of B_A in which the time-passage bit is 1 and the synchronization clock is 0 are visited only transiently between an ε edge and a positive time step, and do not correspond to any states of A. The function $\gamma_2 : Q_{B_A} \to 2^{\mathbb{R}^n}$ strips off the vertex: it is defined by the equation $\gamma(q) = \{v\} \times \gamma_2(q)$. The functions γ, γ', γ_2, and γ'_2 are extended to zones $Z \subset Q_{B^c_A}$ by $\gamma(Z) = \bigcup_{q \in Z} \gamma(q)$, etc. The state q is an *upper-half state* of B_A if $\gamma(q) \neq \emptyset$. The *upper-half space* U_{B_A} is the set of all upper-half states of B_A. The truncation of γ' to γ is justified by the following fact, which follows from the assumption that the preguard region of each edge of A is contained in the invariant region of the source vertex.

Fact 3.8 *For every edge e of an initialized rectangular automaton A, and every upper-half state q of B_A, we have $\gamma_2'(q) \cap pre_A(e) \neq \emptyset$ iff $\gamma_2(q) \cap pre_A(e) \neq \emptyset$.*

Edges of B_A. As in the compact case, the automaton B_A has an edge family for each edge of A. We say that the edges defined by these edge families are *inherited from* A. In addition, B_A has a set of ε edges. The ε edges provide changes to the finite/infinite and weak/strict bitvectors that are caused by the passage of any positive amount of time, however small. Such changes can be carried out only through edge steps.

Epsilon edges. We first define the ε edges of B_A. For all $v, \vec{\mu}, \vec{\nu}$, there is an edge e from $(v, \vec{\mu}, \vec{\nu}, 0)$ to $(v, \vec{\mu}', \vec{\nu}', 1)$ with observation ε. The target vertex components $\vec{\mu}'$ and $\vec{\nu}'$ will be defined presently. The preguard and postguard regions of e are both $[0,0] \times \mathbb{R}^{2n}$, so this edge can be traversed only when both the time-passage bit and the synchronization clock have the value 0. The update set of e is empty. The finite/infinite bitvector must be changed to account for finite bounds that become infinite due to an unbounded flow interval:

$$\mu'_{\ell(i)} = \begin{cases} inf, & \text{if } \inf(flow_A(v)_i) = -\infty, \\ \mu_{\ell(i)}, & \text{otherwise;} \end{cases}$$

$$\mu'_{u(i)} = \begin{cases} inf, & \text{if } \sup(flow_A(v)_i) = \infty, \\ \mu_{u(i)}, & \text{otherwise.} \end{cases}$$

The weak/strict bitvector must be changed to account for weak bounds that become strict due to a strict flow interval:

$$\nu'_{\ell(i)} = \begin{cases} str, & \text{if } strict{\downarrow}(flow_A(v)_i) = str, \\ \nu_{\ell(i)}, & \text{otherwise;} \end{cases}$$

$$\nu'_{u(i)} = \begin{cases} str, & \text{if } strict{\uparrow}(flow_A(v)_i) = str, \\ \nu_{u(i)}, & \text{otherwise.} \end{cases}$$

The ε edges, just defined, play the following role in B_A. Suppose that an edge inherited from A is traversed. Then $tp = 0$ and $b_0 = 0$, and before any time may pass (since no time may pass when $tp = 0$), an ε edge must be traversed, setting tp to 1, and performing whatever bookkeeping is required for the finite/infinite and weak/strict bitvectors. The changes made by the ε edge reflect the situation after some infinitesimal positive amount of time has passed. Therefore no edge inherited from A is allowed after an ε edge but before a positive time step: as long as $tp = 1$ and $b_0 = 0$, no inherited edges are enabled. After the next positive time step, $tp = 1$ and $b_0 > 0$, and another inherited edge may be traversed, resetting both tp and b_0 to 0. Then the situation repeats.

Inherited edges. We now define the edges of B_A that are inherited from A. For this purpose, it is convenient to extend the definition of CNF edge families to allow multiple target vertices. For example, in a guarded command, we may write $\mu_{\ell(i)} := inf$ to change the $\ell(i)$-th component of the finite/infinite bitvector $\vec{\nu}$ to inf. In this way, a disjunction of guarded commands can refer to several target vertices. An *extended CNF edge family for B_A* is completely specified by a source vertex, an observation, the first component of the target vertex (an element of V_A), the time passage bit of the target vertex, and a CNF expression that includes assignments to the bitvectors $\vec{\mu}$ and $\vec{\nu}$. The translation of such an extended CNF edge family into a set of edges for B_A is a straightforward extension of the translation for standard CNF edge families, and will not be detailed.

For each edge $e = (v, w)$ of A, all bitvectors $\vec{\mu}$ and $\vec{\nu}$, and each bit tp, the automaton B_A has the extended CNF edge family $\Psi_{e,\vec{\mu},\vec{\nu},tp} = ((v, \vec{\mu}, \vec{\nu}, tp), obs_A(e), w, 0, \psi_{e,\vec{\mu},\vec{\nu},tp})$. Every edge derived from

strict$\downarrow(J)$	*strict*$\uparrow(I)$	φ_ℓ
wk	*wk*	$\inf(J) \leq \sup(I)$
wk	*str*	$\inf(J) < \sup(I)$
str	*wk*	$\inf(J) < \sup(I)$
str	*str*	$\inf(J) < \sup(I)$

strict$\uparrow(J)$	*strict*$\downarrow(I)$	φ_u
wk	*wk*	$\sup(J) \geq \inf(I)$
wk	*str*	$\sup(J) > \inf(I)$
str	*wk*	$\sup(J) > \inf(I)$
str	*str*	$\sup(J) < \inf(I)$

Table 1: $I \cap J \neq \emptyset$ iff $\varphi_\ell \wedge \varphi_u$

the family $\Psi_{e,\vec{\mu},\vec{\nu},tp}$ shares the observation label of e (which, by assumption, is different from ε), and leads to a target vertex of the form $(w, \vec{\mu}', \vec{\nu}', 0)$. The CNF expression $\psi_{e,\vec{\mu},\vec{\nu},tp}$ is a conjunction $\phi_{tp} \wedge \bigwedge_{i=1}^{n} \psi^i_{e,\vec{\mu},\vec{\nu}}$ of $n+1$ CNF expressions. The guarded command ϕ_0 is $b_0 = 0$, and the guarded command ϕ_1 is $b_0 > 0 \rightarrow b_0 := 0$. Hence an inherited edge from a state with the time-passage bit $tp = 1$ can be taken only if the synchronization clock has a positive value. Upon traversal of the edge, both the time-passage bit and the synchronization clock are reset to 0.

It remains to define the CNF expressions $\psi^i_{e,\vec{\mu},\vec{\nu}}$. We define their guards so that an edge of B_A derived from the edge family $\Psi_{e,\vec{\mu},\vec{\nu},tp}$ can be taken from an upper-half state q of B_A iff the range of possible values for each variable a_i intersects the interval $pre_A(e)_i$, that is, iff $\gamma_2(q) \cap pre_A(e) \neq \emptyset$. Recall that by Fact 3.8, we have $\gamma_2(q) \cap pre_A(e) \neq \emptyset$ iff $\gamma_2'(q) \cap pre_A(e) \neq \emptyset$. If all values are finite and all bounds are weak, then this intersection is nonempty iff both $b_{\ell(i)} \leq \max(pre_A(e)_i)$ and $b_{u(i)} \geq \min(pre_A(e)_i)$. These were the lower-bound and upper-bound guards given in the construction of B^c_A for compact A. Taking strictness and infinite bounds into account, we obtain the more complicated guards $\ell guard(i, pre_A(e)_i)$ and $uguard(i, pre_A(e)_i)$, defined as follows. For an interval $I \subset \mathbb{R}$, and $i \in \{1, \ldots, n\}$, define

$$\ell guard(i, I) = \begin{cases} true, & \text{if } \mu_{\ell(i)} = inf, \\ b_{\ell(i)} \leq \sup(I), & \text{if } \mu_{\ell(i)} = fin \text{ and } \nu_{\ell(i)} = strict\!\uparrow(I) = wk, \\ b_{\ell(i)} < \sup(I), & \text{otherwise;} \end{cases}$$

$$uguard(i, I) = \begin{cases} true, & \text{if } \mu_{u(i)} = inf, \\ b_{u(i)} \geq \inf(I), & \text{if } \mu_{u(i)} = fin \text{ and } \nu_{u(i)} = strict\!\downarrow(I) = wk, \\ b_{u(i)} > \inf(I), & \text{otherwise.} \end{cases}$$

These expressions are guarded commands with no assignments. To understand the definition, consider the conditions under which an interval J intersects the interval I.

Fact 3.9 *Let I and J be two nonempty intervals of the real line, and let φ_ℓ and φ_u be defined as in Table 3.2. Then $I \cap J \neq \emptyset$ iff φ_ℓ and φ_u are both true.*

The guard $\ell guard(i, I)$ on the lower-bound variable $b_{\ell(i)}$ corresponds to the condition φ_ℓ, and the guard $uguard(i, I)$ on the upper-bound variable $b_{u(i)}$ corresponds to φ_u. Notice that φ_ℓ is always true if $\inf(J) = -\infty$. Hence the first line of the definition of $\ell guard(i, I)$. If *strict*$\downarrow(J)$ = *strict*$\uparrow(I)$ = *wk*, then φ_ℓ is $\inf(J) \leq \sup(I)$. Hence the second line of the definition of $\ell guard(i, I)$. Finally, if either *strict*$\downarrow(J)$ = *str* or *strict*$\uparrow(I)$ = *str*, then φ_ℓ is $\inf(J) < \sup(I)$. Hence the third line of the definition of $\ell guard(i, I)$. Symmetrical remarks apply to $uguard(i, I)$ and φ_u. Taking $I = pre_A(e)_i$ and $J = \gamma_2'(q)_i$, Lemma 3.10 follows.

Lemma 3.10 *Let e be an edge of the n-dimensional initialized rectangular automaton A, and let ψ_e be the predicate $\bigwedge_{i=1}^{n}(\ell guard(i, pre_A(e)_i) \wedge uguard(i, pre_A(e)_i))$. For every upper-half state $q = ((v, \vec{\mu}, \vec{\nu}, tp), \mathbf{x})$ of B_A, $Post^e_A(\gamma(q)) \neq \emptyset$ iff the vector $\mathbf{x} \in \mathbb{R}^{2n+1}$ satisfies the predicate ψ_e.*

The reader may recall from the compact case that the construction for coordinates $i \in jump_A(e)$ differs from the construction for coordinates $i \notin jump_A(e)$. We first consider the general construction for the case $i \in jump_A(e)$. In this case, the lower-bound and upper-bound variables $b_{\ell(i)}$ and $b_{u(i)}$ are assigned to the infimum and supremum, respectively, of the interval $post_A(e)_i$, and the finite/infinite and weak/strict bitvectors are updated appropriately. This is done by the lists $\ell assign(i, post_A(e)_i)$ and $uassign(i, post_A(e)_i)$ of assignments, defined as follows. For an interval $I \subset \mathbb{R}$, and $i \in \{1, \ldots, n\}$, define

$$\ell assign(i, I) = \begin{cases} b_{\ell(i)} := \inf(I);\ \mu_{\ell(i)} := \mathit{fin};\ \nu_{\ell(i)} := \mathit{strict}{\downarrow}(post_A(e)_i) & \text{if } \inf(I) \neq -\infty, \\ b_{\ell(i)} := 0;\ \mu_{\ell(i)} := \mathit{inf};\ \nu_{\ell(i)} := \mathit{str} & \text{otherwise;} \end{cases}$$

$$uassign(i, I) = \begin{cases} b_{u(i)} := \sup(I);\ \mu_{u(i)} := \mathit{fin};\ \nu_{u(i)} := \mathit{strict}{\uparrow}(post_A(e)_i) & \text{if } \sup(I) \neq \infty, \\ b_{u(i)} := 0;\ \mu_{u(i)} := \mathit{inf};\ \nu_{u(i)} := \mathit{str} & \text{otherwise.} \end{cases}$$

These expressions are guarded commands with the guards *true*. The assignments to 0 are required for B_A to be initialized. After such an assignment, the value of the variable is ignored due to the finite/infinite bit being *inf*. Now, for $i \in jump_A(e)$, the extended CNF expression $\psi^i_{e,\vec{\mu},\vec{\nu}}$ is defined by

$$\ell guard(i, pre_A(e)_i) \wedge uguard(i, pre_A(e)_i) \wedge \ell assign(i, post_A(e)_i) \wedge uassign(i, post_A(e)_i).$$

From these definitions and Lemma 3.10, we obtain Lemma 3.11.

Lemma 3.11 *Let $e = (v, w)$ be an edge of the initialized rectangular automaton A. For every upper-half state $q = ((v, \vec{\mu}, \vec{\nu}, tp), \mathbf{x})$ of B_A, and every $i \in jump_A(e)$, we have $[Post^e_A(\gamma(q))]^w_i = [\gamma(Post^{\Psi_{e,\vec{\mu},\vec{\nu},tp}}_{B_A}(q))]^w_i$.*

The case $i \notin jump(e)$ is more complicated, because the lower-bound (resp. upper-bound) variable is reset only if its value is too small (resp. too large). Strictness considerations also contribute some complications. Our definitions follow once again from Fact 3.9. The necessary adjustments to $b_{\ell(i)}$ and $b_{u(i)}$, and to the finite/infinite and weak/strict bitvectors, are given by the extended CNF expressions $\ell adjust(i, pre_A(e)_i)$ and $uadjust(i, pre_A(e)_i)$, defined as follows. For an interval $I \subset \mathbb{R}$, and $i \in \{1, \ldots, n\}$, define

$$\ell adjust(i, I) = \begin{cases} \mathit{true} & \text{if } \inf(I) = -\infty, \\ (\mu_{\ell(i)} = \mathit{inf} \ \rightarrow\ \mu_{\ell(i)} := \mathit{fin};\ b_{\ell(i)} := \inf(I);\ \nu_{\ell(i)} := \mathit{strict}{\downarrow}(I)) & \\ \vee (\mu_{\ell(i)} = \mathit{fin} \wedge b_{\ell(i)} < \inf(I) \ \rightarrow\ b_{\ell(i)} := \inf(I);\ \nu_{\ell(i)} := \mathit{strict}{\downarrow}(I)) & \\ \vee (\mu_{\ell(i)} = \mathit{fin} \wedge b_{\ell(i)} = \inf(I) \wedge \nu_{\ell(i)} = \mathit{wk} \ \rightarrow\ \nu_{\ell(i)} := \mathit{strict}{\downarrow}(I)) & \\ \vee (\mu_{\ell(i)} = \mathit{fin} \wedge b_{\ell(i)} = \inf(I) \wedge \nu_{\ell(i)} = \mathit{str}) & \\ \vee (\mu_{\ell(i)} = \mathit{fin} \wedge b_{\ell(i)} > \inf(I)), & \text{otherwise;} \end{cases}$$

$$uadjust(i, I) = \begin{cases} \mathit{true} & \text{if } \sup(I) = \infty, \\ (\mu_{u(i)} = \mathit{inf} \ \rightarrow\ \mu_{u(i)} := \mathit{fin};\ b_{u(i)} := \sup(I);\ \nu_{u(i)} := \mathit{strict}{\uparrow}(I)) & \\ \vee (\mu_{u(i)} = \mathit{fin} \wedge b_{u(i)} > \sup(I) \ \rightarrow\ b_{u(i)} := \sup(I);\ \nu_{u(i)} := \mathit{strict}{\uparrow}(I)) & \\ \vee (\mu_{u(i)} = \mathit{fin} \wedge b_{u(i)} = \sup(I) \wedge \nu_{u(i)} = \mathit{wk} \ \rightarrow\ \nu_{u(i)} := \mathit{strict}{\uparrow}(I)) & \\ \vee (\mu_{u(i)} = \mathit{fin} \wedge b_{u(i)} = \sup(I) \wedge \nu_{u(i)} = \mathit{str}) & \\ \vee (\mu_{u(i)} = \mathit{fin} \wedge b_{u(i)} < \sup(I)), & \text{otherwise.} \end{cases}$$

Then, for $i \notin jump_A(e)_i$, the extended CNF expression $\psi^i_{e,\vec{\mu},\vec{\nu}}$ is defined by

$$\ell guard(i, pre_A(e)_i) \wedge uguard(i, pre_A(e)_i) \wedge \ell adjust(i, pre_A(e)_i) \wedge uadjust(i, pre_A(e)_i).$$

Let us examine the definition of $\ell adjust(i, pre_A(e)_i)$. When the edge e is traversed in A, then new information about the value of a_i is obtained, namely, that it lies within the interval $pre_A(e)_i$ (which, by assumption, is equal to $post_A(e)_i$). Let $p = \inf(pre_A(e)_i)$. If $p = -\infty$, then there is no new lower-bound information, and so $\ell adjust(i, pre_A(e)_i) = true$; hence the first line of the definition. If $p \neq -\infty$, then we distinguish several cases. If $\mu_{\ell(i)} = inf$, then the present lower bound is infinite, and so $\mu_{\ell(i)}$ must be set to fin, and $b_{\ell(i)}$ must be reassigned to p, and the weak/strict bit $\nu_{\ell(i)}$ must be assigned to the lower strictness of $pre_A(e)_i$ (line two). Now suppose that $\mu_{\ell(i)} = fin$. If $b_{\ell(i)} < p$, then again $b_{\ell(i)}$ and $\nu_{\ell(i)}$ must be reset to p and its strictness (line three). If $b_{\ell(i)} = p$ and $\nu_{\ell(i)} = wk$, then information is gained if the lower strictness of $pre_A(e)_i$ is str. So in this case (line four) we perform the assignment $\nu_{\ell(i)} := strict\!\downarrow\!(pre_A(e)_i)$. But if $b_\ell = p$ and $\nu_{\ell(i)} = str$, then no information is gained, and so no assignment is performed (line five). Finally, if $b_{\ell(i)} > p$, then there is no new lower-bound information, and so there is no assignment (line 6). The definition of $uadjust(i, pre_A(e)_i)$ is symmetric. Using Lemma 3.10, we conclude Lemma 3.12.

Lemma 3.12 *Let $e = (v, w)$ be an edge of the initialized rectangular automaton A. For every upper-half state $q = ((v, \vec{\mu}, \vec{\nu}, tp), \mathbf{x})$ of B_A, and every $i \notin jump_A(e)$, we have $[Post^e_A(\gamma(q))]^w_i = [\gamma(Post^{\Psi(e,\vec{\mu},\vec{\nu},tp)}_{B_A}(q))]^w_i$.*

Putting together Lemmas 3.11 and 3.12, we obtain the following.

Lemma 3.13 *Let A be an initialized rectangular automaton. For every upper-half state $q = ((v, \vec{\mu}, \vec{\nu}, tp), \mathbf{x})$ of B_A, and every edge e of A, $Post^e_A(\gamma(q)) = \gamma(Post^{\Psi_{e,\vec{\mu},\vec{\nu},tp}}_{B_A}(q))$. Moreover, the zone $Post^{\Psi_{e,\vec{\mu},\vec{\nu},tp}}_{B_A}(q)$ of B_A is either empty or a singleton.*

Proof. To see that $|Post^{\Psi_{e,\vec{\mu},\vec{\nu},tp}}_{B_A}(q)| \leq 1$, notice that all assignments made by the guarded commands comprising $\Psi_{e,\vec{\mu},\vec{\nu},tp}$ are deterministic, and the disjuncts of $\ell adjust(i, pre_A(e)_i)$ are mutually exclusive, as are the disjuncts of $uadjust(i, pre_A(e)_i)$. So from each state of B_A, at most one of the disjuncts of each of these CNF expressions can be executed. ■

Initial regions of B_A. For a rectangular region $R \subset \mathbb{R}^n$, we define the bitvectors $\vec{\mu}^R, \vec{\nu}^R \in \{0,1\}^{2n}$ and the vector $\mathbf{x}^R \in \mathbb{R}^{2n+1}$ as follows. If $\inf(R_i) = -\infty$, then $\mu^R_{\ell(i)} = inf$, and $\nu^R_{\ell(i)} = str$, and $x^R_{\ell(i)} = 0$. If $\inf(R_i) \neq -\infty$, then $\mu^R_{\ell(i)} = fin$, and $\nu^R_{\ell(i)} = strict\!\downarrow\!(R_i)$, and $x^R_{\ell(i)} = \inf(R_i)$. Similarly, if $\sup(R_i) = \infty$, then $\mu^R_{u(i)} = inf$, and $\nu^R_{u(i)} = str$, and $x^R_{u(i)} = 0$. If $\sup(R_i) \neq \infty$, then $\mu^R_{u(i)} = fin$, and $\nu^R_{u(i)} = strict\!\uparrow\!(R_i)$, and $x^R_{u(i)} = \sup(R_i)$. Finally, let $x^R_0 = 0$. For a rectangular zone Z of A, define $-\gamma(Z)$ to be the zone $\bigcup_{v \in V_A} \{((v, \vec{\mu}^{[Z]^v}, \vec{\nu}^{[Z]^v}, 0), \mathbf{x}^{[Z]^v})\}$ of B_A. Notice that $-\gamma(Z) \subset U_{B_A}$ and that $-\gamma(Z)$ contains at most one state for each vertex of B_A.

We put $Init_{B_A} = -\gamma(Init_A)$. It is a straightforward matter to give the actual initial function of B_A from this. Lemma 3.14 follows immediately from the definitions.

Lemma 3.14 *Let A be an initialized rectangular automaton. For every rectangular zone Z of A, we have $\gamma(-\gamma(Z)) = Z$. In particular, $Init_A = \gamma(Init_{B_A})$.*

This completes the definition of the automaton B_A. Notice that B_A is initialized and singular. The automaton B_A has exponentially more vertices and edges than A. However, as in the translation from initialized stopwatch automata to timed automata, this exponential blowup does not adversely affect the space complexity of reachability analysis. Before we establish this, we first prove the analog of Lemma 3.13 for time steps.

Analysis of time steps. Lemma 3.13 relates the edge steps of A to edge steps of B_A. We must develop a similar correspondence for time steps. For this purpose, the following two facts are useful about the reachable states of B_A are useful. Fact 3.15 says that every reachable state of B_A that is the target of a time step has its finite/infinite and weak/strict bits set correctly. This is because reachability ensures that every sequence of consecutive time steps must have been preceded by an ε step.

Fact 3.15 *Let $((v,\vec{\mu},\vec{\nu},tp),\mathbf{x})$ be a reachable state of B_A with $tp = 1$ and $x_0 > 0$, and let $i \in \{1,\dots,n\}$. First, if $\inf(\mathit{flow}_A(v)_i) = -\infty$, then $\mu_{\ell(i)} = \mathit{inf}$, and if $\sup(\mathit{flow}_A(v)_i) = \infty$, then $\mu_{u(i)} = \mathit{inf}$. Second, if $\mathit{strict}{\downarrow}(\mathit{flow}_A(v)_i) = \mathit{str}$, then $\nu_{\ell(i)} = \mathit{str}$, and if $\mathit{strict}{\uparrow}(\mathit{flow}_A(v)_i) = \mathit{str}$, then $\nu_{u(i)} = \mathit{str}$.*

We say that a state q of B_A *satisfies the invariant* of A if $\gamma(q) \subset Q_A$. Fact 3.16 says that every reachable state of B_A that does not satisfy the invariant of A cannot move, by ε and time steps alone, to a state that does satisfy the invariant. This is because all initial states of B_A and all target states of inherited edge steps satisfy the invariant of A, and because the direction of a flow of B_A can change only after an inherited edge step. The latter follows from B_A being initialized. The former follows from the assumption that the initial region of each vertex of A is contained in the invariant region, and postguard region of each edge is contained in the invariant region of the target vertex.

Fact 3.16 *Let $((v,\vec{\mu},\vec{\nu},tp),\mathbf{x})$ be a reachable state of B_A, and let $i \in \{1,\dots,n\}$. If $x_{\ell(i)} \notin \mathit{inv}_A(v)_i$, then $\inf(\mathit{flow}_A(v)_i) < 0$, and if $x_{u(i)} \notin \mathit{inv}_A(v)_i$, then $\sup(\mathit{flow}_A(v)_i) > 0$.*

Lemma 3.17 relates the time steps of A to time steps of B_A.

Lemma 3.17 *Let A be an initialized rectangular automaton. For every reachable upper-half state q of B_A, and every duration $t \in \mathbb{R}_{\geq 0}$, $\mathit{Post}^t_A(\gamma(q)) = \gamma(\mathit{Post}^t_{B_A}(q))$. Moreover, the zone $\mathit{Post}^t_{B_A}(q) \cap U_{B_A}$ of B_A is either empty or a singleton.*

Proof. Consider a reachable state $q = ((v,\vec{\mu},\vec{\nu},tp),\mathbf{x})$ of B_A, not necessarily upper-half. From the construction of B_A we observe the following. If $tp = 0$, then there is a unique state q' such that $q \xrightarrow{\varepsilon}_{B_A} q'$; furthermore, $\gamma(q') = \emptyset$, and the time-passage bit of q' is 1. If $tp = 1$, then there is no state q' such that $q \xrightarrow{\varepsilon}_{B_A} q'$. If $tp = 0$, then there is no state r such that $q \xrightarrow{[t]}_{B_A} r$, for any $t > 0$. If $tp = 1$, since B_A is a singular automaton, for each $t > 0$, there is a at most one state r such that $q \xrightarrow{[t]}_{B_A} r$; furthermore, the time-passage bit of r is still 1. This shows the second claim of the lemma, that $|\mathit{Post}^t_{B_A}(q) \cap U_{B_A}| \leq 1$. Moreover, since A has no ε moves, the first claim of the lemma is reduced to the following proof obligation: for every reachable upper-half state q of B_A, and every positive duration $t > 0$, show that $\mathit{Post}^{[t]}_A(\gamma(q)) = \gamma(\mathit{Post}^{\pi}_{B_A}(q))$, where $\pi = \varepsilon \cdot [t]$ if the time-passage bit of q is 0, and $\pi = [t]$ if the time-passage bit of q is 1.

So suppose that $q = ((v,\vec{\mu},\vec{\nu},tp),\mathbf{x})$ is a reachable upper-half state of B_A, and consider a duration $t > 0$. Then $\gamma(q)$ has the form $\{v\} \times R$, for a rectangular region $R \subset \mathbb{R}^n$. Recall from the proof of Proposition 2.2 that $\mathit{Post}^{[t]}_A(\gamma(q))$ has the form $\{v\} \times S$, where

$$\inf(S_i) \;=\; \max\{\inf(\mathit{inv}_A(v)_i),\, \inf(R_i) + t \cdot \inf(\mathit{flow}_A(v)_i)\}, \tag{1}$$

$$\sup(S_i) \;=\; \min\{\sup(\mathit{inv}_A(v)_i), \sup(R_i) + t \cdot \sup(\mathit{flow}_A(v)_i)\}. \tag{2}$$

The strictness of the infimum of S_i is given as follows. Put $\mathit{Inv} = \mathit{inv}_A(v)_i$, $\mathit{Flow} = \mathit{flow}_A(v)_i$, and $\mathit{Try} = \inf(R_i) + t \cdot \inf(\mathit{Flow})$. If $\inf(\mathit{Inv}) > \mathit{Try}$, then $\mathit{strict}{\downarrow}(S_i) = \mathit{strict}{\downarrow}(\mathit{Inv})$. If $\inf(\mathit{Inv}) = \mathit{Try}$, then $\mathit{strict}{\downarrow}(S_i) = \mathit{wk}$ iff $\mathit{strict}{\downarrow}(\mathit{Inv}) = \mathit{strict}{\downarrow}(R_i) = \mathit{strict}{\downarrow}(\mathit{Flow}) = \mathit{wk}$. If $\inf(\mathit{Inv}) < \mathit{Try}$, then $\mathit{strict}{\downarrow}(S_i) = \mathit{wk}$ iff $\mathit{strict}{\downarrow}(R_i) = \mathit{strict}{\downarrow}(\mathit{Flow}) = \mathit{wk}$. The strictness of the supremum of S_i is given symmetrically. We show that $\gamma(\mathit{Post}^{\pi}_{B_A}(q)) = \{v\} \times S$, for $\pi = \varepsilon \cdot [t]$ if $\mathit{tp} = 0$, and $\pi = [t]$ if $\mathit{tp} = 1$.

Case $\mathit{Post}^{\pi}_{B_A}(q) \neq \emptyset$**.** In this case, $\mathit{Post}^{\pi}_{B_A}(q)$ has the form $\{r\}$ for a reachable upper-half state $r = ((v, \vec{\mu}', \vec{\nu}', 1), \mathbf{y})$ of B_A with $y_0 > 0$. Hence Fact 3.15 applies to r. The set $\gamma(r)$ has the form $\{v\} \times T$, for a rectangular region $T \subset \mathbb{R}^n$. We show that $T = S$. For this purpose, we show that for each coordinate i, the right endpoints of the intervals T_i and S_i coincide in value and in strictness. The argument for the left endpoints is symmetric. In the following, put $\mathit{Isup} = \sup(\mathit{inv}(v)_i)$, $I{\uparrow} = \mathit{strict}{\uparrow}(\mathit{inv}(v)_i)$, $\mathit{Fsup} = \sup(\mathit{flow}(v)_i)$, and $F{\uparrow} = \mathit{strict}{\uparrow}(\mathit{flow}(v)_i)$.

Subcase $\mu'_{u(i)} = \mathit{fin}$**.** By Fact 3.15, the flow interval $\mathit{flow}(v)_i$ is bounded from above. So the upper-bound variable $b_{u(i)}$ moves at the supremum of the allowable slopes for a_i. Hence, $y_{u(i)} = x_{u(i)} + t \cdot \mathit{Fsup}$ and $\sup(T_i) = \min\{y_{u(i)}, \mathit{Isup}\} = \min\{x_{u(i)} + t \cdot \mathit{Fsup}, \mathit{Isup}\}$. From the subcase assumption and $q \xrightarrow{\pi}_{B_A} r$, we know that $\mu_{u(i)} = \mathit{fin}$, and therefore $\sup(R_i) = \min\{x_{u(i)}, \mathit{Isup}\}$. By Fact 3.16, if $x_{u(i)} > \mathit{Isup}$, then $\mathit{Fsup} > 0$. Thus equation (2) implies $\sup(S_i) = \min\{\mathit{Isup}, x_{u(i)} + t \cdot \mathit{Fsup}\}$, which is the same as $\sup(T_i)$. The question of strictness remains. If $y_{u(i)} > \mathit{Isup}$, then $\mathit{strict}{\uparrow}(T_i) = I{\uparrow} = \mathit{strict}{\uparrow}(S_i)$. If $y_{u(i)} \leq \mathit{Isup}$ and $\nu'_{u(i)} = \mathit{str}$, then from $q \xrightarrow{\pi}_{B_A} r$ we know that either $F{\uparrow} = \mathit{str}$ or $\nu_{u(i)} = \mathit{str}$, and in both cases $\mathit{strict}{\uparrow}(T_i) = \mathit{str} = \mathit{strict}{\uparrow}(S_i)$. If $y_{u(i)} \leq \mathit{Isup}$ and $\nu'_{u(i)} = \mathit{wk}$, then from $q \xrightarrow{\pi}_{B_A} r$ we know that $\nu_{u(i)} = \mathit{wk}$, and Fact 3.15 implies $F{\uparrow} = \mathit{wk}$. Therefore, $\mathit{strict}{\uparrow}(T_i) = \mathit{wk}$ iff either $y_{u(i)} < \mathit{Isup}$ or $I{\uparrow} = \mathit{wk}$, and the same is true for $\mathit{strict}{\uparrow}(S_i)$.

Subcase $\mu'_{u(i)} = \mathit{inf}$**.** Then $\sup(T_i) = \mathit{Isup}$ and $\mathit{strict}{\uparrow}(T_i) = I{\uparrow}$. From the subcase assumption and $q \xrightarrow{\pi}_{B_A} r$, we know that either $\mathit{Fsup} = \mathit{inf}$, or $\mu_{u(i)} = \mathit{inf}$ and therefore $\sup(R_i) = \mathit{Isup}$ and $\mathit{strict}{\uparrow}(R_i) = I{\uparrow}$. Either way, equation (2) implies $\sup(S_i) = \mathit{Isup}$ and $\mathit{strict}{\uparrow}(S_i) = I{\uparrow}$.

Case $\mathit{Post}^{\pi}_{B_A}(q) = \emptyset$**.** This means that for some coordinate i, the lower-bound variable $b_{\ell(i)}$ rises above the upper boundary of $\mathit{inv}_A(v)_i$ within time t, or the upper-bound variable $b_{u(i)}$ drops below the lower boundary of $\mathit{inv}_A(v)_i$. In the first case, either $x_{\ell(i)} + t \cdot \inf(\mathit{flow}_A(v)_i) > \sup(\mathit{inv}_A(v)_i)$, or these two expressions are equal and one of $\nu_{\ell(i)}$ or $\mathit{strict}{\downarrow}(\mathit{flow}_A(v)_i)$ or $\mathit{strict}{\uparrow}(\mathit{inv}_A(v)_i)$ is str. The second case is symmetric and will not be detailed. If $x_{\ell(i)} + t \cdot \inf(\mathit{flow}_A(v)_i) > \sup(\mathit{inv}_A(v)_i)$, then equations (1) and (2) together imply $\inf(S_i) > \sup(S_i)$. If $x_{\ell(i)} + t \cdot \inf(\mathit{flow}_A(v)_i) = \sup(\mathit{inv}_A(v)_i)$ and one of $\nu_{\ell(i)}$ or $\mathit{strict}{\downarrow}(\mathit{flow}_A(v)_i)$ or $\mathit{strict}{\uparrow}(\mathit{inv}_A(v)_i)$ is str, then $\inf(S_i) \geq \sup(S_i)$ and either $\mathit{strict}{\uparrow}(S_i) = \mathit{str}$ or $\mathit{strict}{\downarrow}(S_i) = \mathit{str}$. In all cases, $S_i = \emptyset$ and therefore $\{v\} \times S = \emptyset$. But from the assumption $\mathit{Post}^{\pi}_{B_A}(q) = \emptyset$ it follows that also $\gamma(\mathit{Post}^{\pi}_{B_A}(q)) = \emptyset$, which concludes the proof. ■

From Lemmas 3.13 and 3.17 it follows that for every upper-half zone $Z \subset U_{B_A}$, and every label $\tau \in \Sigma \cup \mathbb{R}_{\geq 0}$, $\mathit{Post}^{\tau}_{A}(\gamma(Z)) = \gamma(\mathit{Post}^{\tau}_{B_A}(Z))$ and, by Proposition 2.1, $\mathit{Pre}^{\tau}_{A}(\gamma(q)) = \gamma(\mathit{Pre}^{\tau}_{-B_{-A}}(q))$. Hence we again have the commuting diagram of Figure 8, with all superscripts c and the restriction that Z be compact removed. From Lemma 3.14 and the fact that Post commutes with γ, we conclude, as in the compact case, that $\mathit{Reach}(A) = \gamma(\mathit{Reach}(B_A))$, that $\mathit{Lang}(A) \subset \mathit{Lang}(B_A)$, and that A and B_A accept the same finite timed words.

Theorem 3.18 *The reachability problem for initialized rectangular automata is complete for PSPACE.*

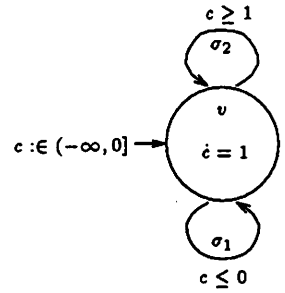

Figure 9: The initialized rectangular automaton $\tilde{A}$ with $Lang(\tilde{A}) \subsetneq Lang(B_{\tilde{A}})$

Proof. For containment in PSPACE, notice that $|V_{B_A}| = |V_A| \cdot 2^{4n+1}$, and that the definition of B_A uses the same constants as the definition of A. The space requirement $O(\log|V_{B_A}| + (2n+1) \cdot (\log(2n+1) + \log L_{B_A})) = O(\log|V_A| + n \cdot (\log n + \log L_A))$, where L_A is defined as for singular automata, is polynomial in the size of A. ■

3.3 ω-Language Emptiness

While the initialized rectangular automaton A and the initialized singular automaton B_A accept the same finite timed words, the automaton B_A may accept infinite timed words that are not accepted by A. To see this, consider the 1D initialized rectangular automaton $\tilde{A}$ from Figure 9. The only coordinate of $\tilde{A}$ is a clock, the only vertex of $\tilde{A}$ is v, and the invariant region of v is $\mathbb{R}$. Note that $\tilde{A}$ is not a timed automaton, because the initial region $init_{\tilde{A}}(v) = (-\infty, 0]$ is unbounded from below. While every finite timed word of the form $(1\sigma_1)^m$ is accepted by $\tilde{A}$, the divergent timed word $(1\sigma_1)^\omega$ is not accepted by $\tilde{A}$. This is because, no matter what the initial value of the clock c, in a divergent run it will eventually be positive. However, in $B_{\tilde{A}}$ the finite/infinite bit for the lower bound on c is initially *inf*, and remains *inf* during time steps and σ_1 steps. Therefore $(1\sigma_1)^\omega$ is accepted by $B_{\tilde{A}}$. A similar phenomenon is exhibited with unbounded postguard intervals. The definition of bounded nondeterminism precludes both.

Closure uder divergent limits

A set $\mathcal{L}$ of infinite timed words is *limit-closed* if for all infinite timed words $\overline{\tau}$, if every finite prefix of $\overline{\tau}$ is a prefix of some word in $\mathcal{L}$, then $\overline{\tau}$ itself is in $\mathcal{L}$. Since we are interested only in infinite timed words that diverge, we relax the requirement of limit closure as follows [HNSY94]. The set $\mathcal{L}$ is *closed under divergent limits* if for all divergent timed words $\overline{\tau}$, if every finite prefix of $\overline{\tau}$ is a prefix of some word in $\mathcal{L}$, then $\overline{\tau}$ itself is in $\mathcal{L}$. A set of divergent timed words that is closed under divergent limits is completely determined by its finite prefixes. So if we have two rectangular automata A_1 and A_2 such that (1) every finite timed word accepted by A_1 is accepted also by A_2, and (2) the ω-language $Lang(A_2)$ is closed under divergent limits, then $Lang(A_1) \subset Lang(A_2)$. Specifically, if A is an initialized rectangular automaton whose ω-language $Lang(A)$ is closed under divergent limits, then $Lang(A) = Lang(B_A)$.

As we have seen, the ω-language of the initialized rectangular automaton $\tilde{A}$ is not closed under divergent limits. The unbounded initial region of $\tilde{A}$ is to blame. We will prove that for every initialized rectangular automaton A with bounded nondeterminism, the ω-language $Lang(A)$ is closed under divergent limits. Consequently, for initialized rectangular automata A with bounded nondeterminism, the translation from A to B_A can be used to solve not only reachability but also ω-language emptiness.

For a rectangular automaton A, the *ω-language $Lang^c(A)$ with convergent words* is the set of all infinite timed words, divergent or not, that are accepted by runs of A. One way of showing that $Lang(A)$ is closed under divergent limits is to prove that $Lang^c(A)$ is limit-closed. While it is not true for all initialized rectangular automata A with bounded nondeterminism that $Lang^c(A)$ is limit-closed, this is true in the special case that A has compact nondeterminism. So we first consider the special case of compact nondeterminism, and then proceed to the more general case of bounded nondeterminism.

Preliminary definitions. Let A be an n-dimensional rectangular automaton (without ε moves). In this section, it is convenient to consider timed words over the alphabet $E \cup \mathbb{R}_{\geq 0}$, where the edge set replaces the observation alphabet. Formally, a *timed edge word* for A is a finite or infinite sequence $\overline{\pi} = \pi_0\pi_1\pi_2\cdots$ of letters from $E \cup \mathbb{R}_{\geq 0}$. An *edge run* of A that accepts $\overline{\pi}$ is a sequence of the form $q_0 \xrightarrow{\pi_0} q_1 \xrightarrow{\pi_1} q_2 \xrightarrow{\pi_2} \cdots$ with $q_0 \in Init$, and $q_i \in Q$ for all $i \geq 0$. Divergence for timed edge words and edge runs is defined as for timed words and runs. The *edge ω-language* of A (*with convergent words*), denoted $Lang_E(A)$ (resp. $Lang_E^c(A)$), is the set of all divergent (resp. all infinite) timed edge words that are accepted by runs of A. Limit closure and closure under divergent limits for edge ω-languages is defined as for ω-languages. The following observation is due to the fact that the edge set of A is finite.

Proposition 3.19 *Let A be a rectangular automaton. First, if the edge ω-language $Lang_E(A)$ is closed under divergent limits, then so is the ω-language $Lang(A)$. Second, if the edge ω-language $Lang_E^c(A)$ with convergent words is limit-closed, then so is $Lang^c(A)$.*

Proof. We say that a timed edge word $\pi_0\pi_1\pi_2\cdots$ *matches* the timed word $\tau_0\tau_1\tau_2\cdots$ if for all $i \geq 0$, if $\pi_i \in E$ then $\tau_i = obs(\pi_i)$, and if $\pi_i \in \mathbb{R}_{\geq 0}$ then $\tau_i = \pi_i$. Assume that $Lang_E^c(A)$ is limit-closed, and consider all finite prefixes of an infinite timed word $\overline{\tau} \in Lang^c(A)$. The finite timed edge words that match these prefixes and are accepted by A form the nodes of a finitely branching tree: the root of the tree is the empty timed edge word, and the successors of a node w are the (matching and accepted) timed edge words of the form $w \cdot \pi$, for $\pi \in E \cup \mathbb{R}_{\geq 0}$. By König's lemma, the tree has an infinite rooted path, and the nodes along this path are the finite prefixes of an infinite timed edge word $\overline{\pi}$. Since $Lang_E^c(A)$ is limit-closed, $\overline{\pi} \in Lang_E^c(A)$. Since $\overline{\pi}$ matches $\overline{\tau}$, we conclude that $\overline{\tau} \in Lang^c(A)$. This proves the second claim of the lemma. The first claim follows from the fact that if $\overline{\tau}$ is divergent, then $\overline{\pi}$ is also divergent. ■

Notice that for every singular automaton A, the edge ω-language $Lang_E^c(A)$ with convergent words (and therefore also $Lang^c(A)$) is limit-closed. This is because a singular automaton A has a finite number of initial states, and all edge steps as well as time steps are deterministic; that is, for all states q of A, and every label $\pi \in E \cup \mathbb{R}_{\geq 0}$, the zone $Post^{\pi}(q)$ is either empty or a singleton. Next, we consider a class of nonsingular, and therefore nondeterministic, automata whose ω-languages with convergent words are limit-closed.

The case of compact nondeterminism

The rectangular automaton A has *compact nondeterminism* if it has bounded nondeterminism, and all rectangular regions that appear in the definition of A are closed. More precisely, A has *compact nondeterminism* if (1) for every vertex v of A, the regions $init(v)$ and $flow(v)$ are compact, and *inv*(v) is closed, and (2) for every edge e of A, the regions *pre*(e) and *post*(e) are closed, and for and every coordinate $i \in \{1, \ldots, n\}$ with $i \in$ *jump*(e), the interval *post*(e) is bounded. We show that if A has compact nondeterminism, then the ω-language $Lang^c(A)$ with convergent words is limit-closed. This result is a consequence of the following basic property of compact zones, which is inherited from $\mathbb{R}^n$.

Proposition 3.20 *Let A be a rectangular automaton, and let $(Z_i)_{i\in\mathbb{N}}$ be an infinite decreasing sequence of nonempty compact zones of A; that is, $Z_i \supset Z_{i+1}$ for all $i \geq 0$. Then the intersection $\bigcap_{i\in\mathbb{N}} Z_i$ is nonempty.*

Proof. This follows from the corresponding statement for regions (subsets of $\mathbb{R}^n$), and the fact that the vertex set of A is finite. ■

The next fact points out the compactness of all zones that will appear in the proof of the main theorem. It follows immediately from the proof of Proposition 2.2.

Fact 3.21 *Let A be a rectangular automaton with compact nondeterminism, and let Z and Z' be two compact rectangular zones of A. For every label $\pi \in E \cup \mathbb{R}_{\geq 0}$, the zones $Post^{\pi}(Z)$ and $Pre^{\pi}(Z') \cap Z$ are compact and rectangular.*

Note the asymmetry in Fact 3.21: the intersection of $Pre^{\pi}(Z')$ with the compact zone Z is required for compactness, because the preguard regions of automata with compact nondeterminism are not necessarily bounded. The next lemma gives the heart of the limit-closure argument, showing that if all finite prefixes of an infinite timed edge word can be generated from a given zone Z, then there is a single state in Z from which each prefix can be generated.

Lemma 3.22 *Let A be a rectangular automaton with compact nondeterminism, and let Z be a compact rectangular zone of A. Suppose that $\overline{\pi} \in (E \cup \mathbb{R}_{\geq 0})^{\omega}$ is a timed edge word such that for every $m \in \mathbb{N}$, $Post^{\pi_0\pi_1\cdots\pi_m}(Z) \neq \emptyset$. Then there is a state $q \in Z$ such that for every $m \in \mathbb{N}$, $Post^{\pi_0\pi_1\cdots\pi_m}(q) \neq \emptyset$.*

Proof. For each $m \in \mathbb{N}$, define $Z_m = \{q \in Z \mid Post^{\pi_0\pi_1\cdots\pi_m}(q) \neq \emptyset\}$. Since each $Post^{\pi_0\pi_1\cdots\pi_m}(Z)$ is nonempty, each Z_m is nonempty. Also, $Z_m \supset Z_{m+1}$ for all $m \geq 0$. We show that each zone Z_m is compact; then the lemma follows from Proposition 3.20. By Fact 3.21, for each $m \in \mathbb{N}$, the zone $Post^{\pi_0\pi_1\cdots\pi_m}(Z)$ is compact and rectangular. Therefore, again by Fact 3.21, the zone $Z_m = Z \cap Pre^{\pi_0\pi_1\cdots\pi_m}(Post^{\pi_0\pi_1\cdots\pi_m}(Z))$ is compact as well. ■

Now we are ready to prove the main theorem.

Theorem 3.23 *If A is a rectangular automaton with compact nondeterminism, then the ω-language $Lang^c(A)$ with convergent words is limit-closed.*

Proof. Let A be a rectangular automaton with compact nondeterminism. By Proposition 3.19, it suffices to show that the edge ω-language $Lang^c_E(A)$ with convergent words is limit-closed. Suppose that $\overline{\pi} \in (E \cup \mathbb{R}_{\geq 0})^{\omega}$ is an infinite timed edge word such that for every $m \in \mathbb{N}$, the prefix $\pi_0\pi_1\cdots\pi_m$ is a finite prefix of a timed edge word in $Lang^c_E(A)$; that is, for each $m \geq 0$, $Post^{\pi_0\pi_1\cdots\pi_m}(Init) \neq \emptyset$. We need to show that $\overline{\pi} \in Lang^c_E(A)$.

We define an infinite sequence of zones $(Z_i)_{i\in\mathbb{N}}$ of A. Let $Z_0 = Init$. Since Z_0 is compact and rectangular, by Lemma 3.22, there is a state $q_0 \in Z_0$ such that for each $m \geq 0$, $Post^{\pi_0\pi_1\cdots\pi_m}(q_0) \neq \emptyset$.

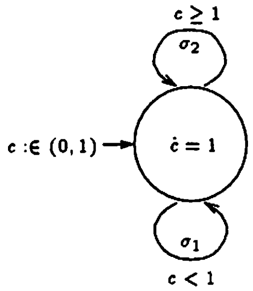

Figure 10: The initialized rectangular automaton $\bar{A}$ for which $Lang^c(\bar{A})$ not limit-closed

Let $Z_1 = Post^{\pi_0}(q_0)$. Then Z_1 is compact and rectangular, and for each $m \geq 1$, $Post^{\pi_1\pi_2\cdots\pi_m}(Z_1) \neq \emptyset$. So by Lemma 3.22, there is a state $q_1 \in Z_1$ such that for each $k \geq 1$, $Post^{\pi_1\pi_2\cdots\pi_m}(q_1) \neq \emptyset$. Proceed inductively in this manner, with $Z_{i+1} = Post^{\pi_i}(q_i)$ compact and rectangular, and $q_{i+1} \in Z_{i+1}$ given by Lemma 3.22 such that for each $m \geq i+1$, $Post^{\pi_{i+1}\pi_{i+2}\cdots\pi_m}(q_{i+1}) \neq \emptyset$. Then $q_0 \xrightarrow{\pi_0} q_1 \xrightarrow{\pi_1} q_2 \xrightarrow{\pi_2} \cdots$ is an edge run of A that accepts $\overline{\pi}$. ■

Corollary 3.24 *The ω-language emptiness problem for initialized rectangular automata with compact nondeterminism is complete for PSPACE.*

The case of bounded nondeterminism

For rectangular automata A with bounded, noncompact nondeterminism, the ω-language $Lang^c(A)$ with convergent words may not be limit-closed. To see this, consider the 1D initialized rectangular automaton $\bar{A}$ from Figure 10 (with the trivial invariant function $\lambda.\mathbb{R}$). While every finite prefix of the infinite timed word $\overline{\tau} = \frac{1}{2}\sigma_1\frac{1}{4}\sigma_1\frac{1}{8}\sigma_1\cdots$ is accepted by $\bar{A}$, the convergent word $\overline{\tau}$ is not in $Lang^c(\bar{A})$. However, we show that the ω-language $Lang(A)$ is still closed under *divergent* limits for all rectangular automata A that have bounded nondeterminism and are initialized, like the sample automaton $\bar{A}$. Bounded regions have no analogue to Proposition 3.20, and this greatly complicates the proof, which now relies on a detailed case analysis of the flow function. The following fact points out the boundedness of all zones that will appear in the proof of the main theorem.

Fact 3.25 *Let A be a rectangular automaton with bounded nondeterminism, and let Z be a bounded rectangular zones of A. For every label $\pi \in E \cup \mathbb{R}_{\geq 0}$, the zone $Post^{\pi}(Z)$ is bounded and rectangular.*

The next lemma gives the heart of the argument. It shows that for 1D rectangular automata A with constant, bounded flow functions, $Lang(A)$ is closed under those divergent limits that are accepted by runs without discontinuous jumps in the continuous state.

Lemma 3.26 *Let $\mathcal{I}$ be a finite set of intervals, and let flow be a bounded interval. Let $(t_i)_{i\in\mathbb{N}}$ be an infinite sequence of positive reals with $\sum_{i=0}^{\infty} t_i = \infty$, and let $(I_i)_{i\in\mathbb{N}}$ be an infinite sequence of intervals such that I_0 is bounded and each I_i is the intersection of one or more members of $\mathcal{I}$. Suppose that for each $m \in \mathbb{N}$, there is a finite sequence $x_0, x_1, \ldots, x_m$ of reals such that for all $i \in \{0, \ldots, m\}$, we have $x_i \in I_i$, and for all $i \in \{0, \ldots, m-1\}$, we have $(x_{i+1} - x_i)/t_i \in flow$. Then there is an infinite sequence $(x_i)_{i\in\mathbb{N}}$ of reals such that for all $i \geq 0$, we have $x_i \in I_i$ and $(x_{i+1} - x_i)/t_i \in flow$.*

Proof. We call a finite sequence $x_0, x_1, \ldots, x_m$ *m-admissible* if for all $i \in \{0, \ldots, m\}$, we have $x_i \in I_i$, and for all $i \in \{0, \ldots, m-1\}$, we have $(x_{i+1} - x_i)/t_i \in \mathit{flow}$. We call an infinite sequence $(x_i)_{i \in \mathbb{N}}$ *admissible* if for all $i \geq 0$, we have $x_i \in I_i$ and $(x_{i+1} - x_i)/t_i \in \mathit{flow}$. Think of these sequences as values of a variable a, with $\dot{a} \in \mathit{flow}$, in a run with time steps of durations t_i and without discontinuous jumps. Let $\mathcal{J}$ be the set $\{J \subset \mathbb{R} \mid J = I_i \text{ for infinitely many } i\}$ of intervals.

Case $0 \notin$ closure(*flow*). Suppose that $\mathit{flow} \subset (\delta, \infty)$ for some $\delta > 0$. The case of $\mathit{flow} \subset (-\infty, \delta)$ is handled symmetrically. Let $k \in \mathbb{R}$ be larger than all of the finite endpoints of the intervals in $\mathcal{I}$. The point here is that the slope of a is bounded from below by δ, so that once $(k - \inf(I_0))/\delta$ time units have passed, no matter what the initial value of a, the value of a will be greater than all of the finite endpoints of intervals from $\mathcal{I}$ (note that $\inf(I_0)$ is finite, because I_0 is bounded). Let j be large enough so that $\sum_{i=0}^{j-1} t_i > (k - \inf(I_0))/\delta$; such a j exists because the sum $\sum_{i=0}^{\infty} t_i$ diverges. Then, for every m-admissible sequence $x_0, x_1, \ldots, x_m$, we have $x_i > k$ for all $i \in \{j+1, \ldots, m\}$. Since each I_i is an intersection of intervals from $\mathcal{I}$, it follows that for every $i > j$, we have $I_i \supset (k, \infty)$. Consider a j-admissible sequence $x_0, x_1, \ldots, x_j$ and choose $\delta' \in \mathit{flow}$. Then $x_0, x_1, \ldots, x_j, x_j + \delta' \cdot t_j, x_j + \delta' \cdot (t_j + t_{j+1}), \ldots$ is an admissible sequence.

Case $0 = \inf(\mathit{flow})$ **or** $0 = \sup(\mathit{flow})$. We suppose that $0 = \inf(\mathit{flow})$; the case $0 = \sup(\mathit{flow})$ is handled symmetrically. Among the intervals I_i are only finitely many distinct intervals. Therefore $\bigcap \mathcal{J} \neq \emptyset$, because $\mathit{flow} \cap (-\infty, 0) = \emptyset$, so a can never descend from an interval I_i to an interval I_j all of whose elements are less than those of I_i. Let j_1 be large enough so that (1) for every $i \in \mathbb{N}$, we have $I_{j_1+i} \in \mathcal{J}$, and (2) for every $J \in \mathcal{J}$, there is an $i < j_1$ with $I_i = J$; that is, j_1 is large enough so that all elements of $\mathcal{J}$ have been met in the past, and only elements of $\mathcal{J}$ will be met in the future. Let $j_2 > j_1$ be large enough so that all elements of $\mathcal{J}$ are represented among $I_{j_1+1}, \ldots, I_{j_2-1}$. Let $x_0, x_1, \ldots, x_{j_2}$ be a j_2-admissible sequence. Then $x_{j_1} \in \bigcap \mathcal{J}$, because a cannot decrease, each interval in $\mathcal{J}$ contains at least one x_i with $i < j_1$, and each interval in $\mathcal{J}$ contains at least one x_i with $j_1 < i < j_2$. If $0 \in \mathit{flow}$, then $x_0, x_1, \ldots, x_{j_1}^{\omega}$ is an admissible sequence. If $0 \notin \mathit{flow}$, then $x_{j_1} < \sup(\bigcap \mathcal{J})$. Let $\delta = \sup(\bigcap \mathcal{J}) - x_{j_1}$. For each $i > j_1$, choose δ_i so that $0 < \delta_i < \delta/(2^i \cdot t_i)$. Then $x_0, x_1, \ldots, x_{j_1}, x_{j_1} + \delta_{j_1+1} \cdot t_{j_1+1}, x_{j_1} + \delta_{j_1+1} \cdot t_{j_1+1} + \delta_{j_1+2} \cdot t_{j_1+2}, \ldots$ is admissible.

Case $0 \in$ interior(*flow*) **and** $\bigcap \mathcal{J} \neq \emptyset$. Since 0 is in the interior of *flow*, every m-admissible sequence can be slowed down to give another m-admissible sequence. Let j_1 be as in the previous case. Since $0 \in \mathit{flow}$, whenever a $(j_1 + i)$-admissible sequence, $i \geq 0$, terminates in $\bigcap \mathcal{J}$, then it can be extended to an admissible sequence by repeating the last value ad infinitum. Such a $(j_1 + i)$-admissible sequence terminating in $\bigcap \mathcal{J}$ exists, because there exist two intervals $J_1, J_2 \in \mathcal{J}$ such that $\inf(J_1) = \inf(\bigcap \mathcal{J})$ with identical strictness, and $\sup(J_2) = \sup(\bigcap \mathcal{J})$ with identical strictness. Let j_2 be large enough so that both J_1 and J_2 each appear twice in $I_{j_1+1}, \ldots, I_{j_2-1}$. Let $x_0, x_1, \ldots, x_{j_2}$ be a j_2-admissible sequence. There must be two positions i_1 and i_2 with $j_1 < i_1 < i_2 < j_2$ such that $x_{i_1} \in J_1$ and $x_{i_2} \in J_2$. By slowing down the x_i sequence, $\bigcap \mathcal{J}$ can be reached: if $x_{i_1} \geq \sup(\bigcap \mathcal{J})$ and $x_{i_2} \leq \inf(\bigcap \mathcal{J})$, then for some j with $i_1 \leq j < i_2$, we have $x_j \geq \sup(\bigcap \mathcal{J})$ and $x_{j+1} < \sup(\bigcap \mathcal{J})$. Choose $y \in \bigcap \mathcal{J}$ so that $y > x_{j+1}$. Then $x_0, x_1, \ldots, x_j, y^{\omega}$ is an admissible sequence.

Case $0 \in$ interior(*flow*) **and** $\bigcap \mathcal{J} = \emptyset$. Let $J_1, J_2 \in \mathcal{J}$ be such that every element of J_1 is greater than every element of J_2. Let j_1 be as in the previous two cases. Let $j_1 < p_1 < q_1 < p_2$ be so that $I_{p_1} = I_{p_2} = J_1$ and $I_{q_1} = J_2$. Let $x_0, x_1, \ldots, x_{p_2}$ be a p_2-admissible sequence. We first show that for every $m \in \mathbb{N}$, there is an m-admissible sequence starting from x_0. This is obvious for $m \leq p_2$, so suppose that $m > p_2$. Let $y_0, y_1, \ldots, y_m$ be an m-admissible sequence. We have three subcases.

Subcase $x_{p_1} = y_{p_1}$. In this case $x_0, x_1, \ldots, x_{p_1}, y_{p_1+1}, y_{p_1+2}, \ldots, y_m$ is m-admissible.

Subcase $x_{p_1} < y_{p_1}$. If $x_{q_1} < y_{q_1}$, then by slowing down, the x_i sequence can meet up with the y_i sequence somewhere along the descent from J_1 to J_2. If $x_{q_1} > y_{q_1}$, then for some $j \in \{p_1+1, \dots, q_1\}$, we have $x_j < y_j$ and $x_{j+1} \geq y_{j+1}$. Since $(y_{j+1} - y_j)/t_j \in \mathit{flow}$ and $(x_{j+1} - x_j)/t_j \in \mathit{flow}$, and $y_{j+1} - y_j < y_{j+1} - x_j < x_{j+1} - x_j$, it must be that $(y_{j+1} - x_j)/t_j \in \mathit{flow}$. Hence the sequence $x_0, x_1, \dots, x_j, y_{j+1}, y_{j+2}, \dots, y_m$ is m-admissible.

Subcase $x_{p_1} > y_{p_1}$. If $x_{q_1} < y_{q_1}$, then by slowing down, the x_i sequence can meet up with the y_i sequence somewhere along the descent from J_1 to J_2. So suppose that $x_{q_1} > y_{q_1}$. Now if $x_{p_2} > y_{p_2}$, then by slowing down, the x_i sequence can meet up with the y_i sequence somewhere along the ascent from J_2 to J_1. If $x_{p_2} < y_{p_2}$, then the y_i sequence must cross the x_i sequence as above, and the $x_0, x_1, \dots, x_j, y_{j+1}, y_{j+2}, \dots, y_m$ construction from the previous subcase provides an m-admissible sequence. This completes the case analysis.

It remains to construct an infinite admissible sequence. Let $x_0 \in I_0$ be such that for every $m \in \mathbb{N}$, there is an m-admissible sequence starting from x_0. Let R be the set of t_0-successors of x_0; that is, $R = \{y \in \mathbb{R} \mid (y - x_0)/t_0 \in \mathit{flow}\}$. Since *flow* is bounded, R is bounded. Let $(t'_i)_{i \in \mathbb{N}}$ be the sequence of reals such that $t'_i = t_{i+1}$ for all $i \geq 0$. Let $(I'_i)_{i \in \mathbb{N}}$ be the sequence of intervals such that $I'_0 = I_1 \cap R$, and $I'_i = I_{i+1}$ for all $i \geq 1$. If we apply what we have already proven to the duration sequence $(t'_i)_{i \in \mathbb{N}}$ and the interval sequence $(I'_i)_{i \in \mathbb{N}}$, we conclude that there is an $x_1 \in I'_0$ such that for every $m \in \mathbb{N}$, there is an m-admissible (with respect to $(t'_i)_{i \in \mathbb{N}}$ and $(I'_i)_{i \in \mathbb{N}}$) sequence starting from x_1. Continuing inductively, we form an admissible sequence starting from x_0. ■

Now the proof of the main theorem consists of reduction to one dimension, eliminating discontinuous jumps, and applying Lemma 3.26.

Theorem 3.27 *If A is an initialized rectangular automaton with bounded nondeterminism, then the ω-language $Lang(A)$ closed under divergent limits.*

Proof. Let A be an n-dimensional initialized rectangular automaton with bounded nondeterminism. By Proposition 3.19, it suffices to show that the edge ω-language $Lang_E(A)$ is closed under divergent limits. Suppose that $\overline{\pi} \in (E \cup \mathbb{R}_{\geq 0})^\omega$ is a divergent timed edge word such that for every $m \in \mathbb{N}$, there is a finite edge run of A of the form $q_0^m \xrightarrow{\pi_0} q_1^m \xrightarrow{\pi_1} \cdots \xrightarrow{\pi_m} q_{m+1}^m$. We need to show that there is an infinite edge run ρ of A of the form $q_0 \xrightarrow{\pi_0} q_1 \xrightarrow{\pi_1} q_2 \xrightarrow{\pi_2} \cdots$. First, observe that each of the n coordinates of a rectangular automaton is independent of the other coordinates. Hence, A has an edge run ρ iff each of the n 1D rectangular automata defined by projecting the continuous state of A to one of the coordinates has the corresponding projection of ρ as an edge run. Therefore we assume that $n = 1$.

Let $\omega = \{i \in \mathbb{N} \mid 1 \in \mathit{jump}(\pi_i)\}$ be the set of positions with discontinuous jumps in the first (and only) coordinate. If $i \in \omega$, then the value of the continuous state after the i-th step is independent of its previous value. So if ω is infinite, then we can string together ρ from infinitely many segments that lead from one discontinuous jump to the next: for all $j \geq 0$, choose $q_j = q_j^{f(j)}$ for $f(j) = \min\{i \in \omega \mid i \geq j\}$. If ω is finite, then we can string together ρ from $|\omega|$ many segments that lead, as in the previous case, from one discontinuous jump to the next, followed by an application of Lemma 3.26. In Lemma 3.26, let $\mathcal{I}$ be the set of all invariant, preguard, and postguard intervals of A; let *flow* be the flow interval of the vertex associated with the state q_j^k for any $k \geq j > \max(\omega)$; and let I_0 be the interval associated with the zone $Post^{\pi_0\pi_1\cdots\pi_{\max(\omega)}}(Init)$. The initialization of A ensures that after the first $\max(\omega)$ steps, the flow interval *flow* remains constant. The bounded nondeterminism of A ensures, by Fact 3.25, that the zone $Post^{\pi_0\pi_1\cdots\pi_{\max(\omega)}}(Init)$ is bounded and rectangular. ■

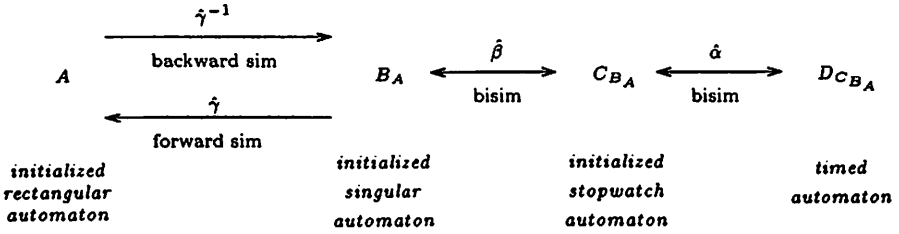

Figure 11: Chain of timed simulations from A to $D_{C_{B_A}}$

Corollary 3.28 *The ω-language emptiness problem for initialized rectangular automata with bounded nondeterminism is complete for PSPACE.*

3.4 Simulation Relations

We introduced several mappings between the state spaces of rectangular automata. We were interested only that the mappings preserve reachability and ω-languages. Now we study the mappings in greater detail and show that they are timed (bi)simulations [LV96] on the underlying labeled transition systems. In particular, the map α from Section 3.1 specifies a timed bisimulation between an initialized stopwatch automaton C and the timed automaton D_C, the map β from Section 3.1 specifies a timed bisimulation between an initialized singular automaton B and the initialized stopwatch automaton C_B, and the map γ from Section 3.2 specifies a timed forward simulation of an initialized rectangular automaton A by the initialized singular automaton B_A, as well as a timed backward simulation of B_A by A.

Let A_1 and A_2 be two rectangular automata with ε moves and the same observation alphabet. A binary relation $\chi \subset \mathit{Reach}(A_1) \times \mathit{Reach}(A_2)$ is a *timed forward simulation of A_2 by A_1* if the following two conditions are met.

1. For every initial state r of A_2, there is an initial state q of A_1 such that $(q, r) \in \chi$.
2. For all states $r, r' \in \mathit{Reach}(A_2)$, every state $q \in \mathit{Reach}(A_1)$ with $(q, r) \in \chi$, and every label $\tau \in \Sigma \cup \mathbb{R}_{\geq 0}$, if $r \xrightarrow{\tau}_{A_2} r'$, then there exists a state $q' \in \mathit{Reach}(A_1)$ such that $(q', r') \in \chi$ and $q \xrightarrow{\tau}_{A_1} q'$.

The relation χ is a *timed backward simulation of A_2 by A_1* if the following three conditions are met.

1. For every state $r \in \mathit{Reach}(A_2)$, there exists a state $q \in \mathit{Reach}(A_1)$ with $(q, r) \in \chi$.
2. For every initial state r of A_2, and every state $q \in \mathit{Reach}(A_1)$, if $(q, r) \in \chi$, then q is an initial state of A_1.
3. For all states $r, r' \in \mathit{Reach}(A_2)$, every state $q' \in \mathit{Reach}(A_1)$ with $(q', r') \in \chi$, and every label $\tau \in \Sigma \cup \mathbb{R}_{\geq 0}$, if $r \xrightarrow{\tau}_{A_2} r'$, then there exists a state $q \in \mathit{Reach}(A_1)$ such that $(q, r) \in \chi$ and $q \xrightarrow{\tau}_{A_1} q'$.

If χ is a timed forward simulation of A_2 by A_1, and χ^{-1} is a timed forward simulation of A_1 by A_2, then χ is a *timed bisimulation* between A_1 and A_2. Notice that if there is a timed forward simulation of A_2 by A_1, then every timed word accepted by A_2 is accepted also by A_1; if there is a timed backward simulation of A_2 by A_1, then every finite timed word accepted by A_2 is accepted also by A_1.

$$\begin{tikzcd}
Init_{D_{C_{B_A}}} \arrow[d, shift left, "\alpha"] \arrow[d, leftarrow, shift right, "-\alpha"'] & \cdots & \bullet \arrow[d, "\alpha"] & \bullet \arrow[l, "Pre_{D_{C_{-B_{-A}}}}"'] \arrow[d, "\alpha"] & \bullet \arrow[l, "Pre_{D_{C_{-B_{-A}}}}"'] \arrow[r, "Post_{D_{C_{B_A}}}"] \arrow[d, shift left, "\alpha"] \arrow[d, leftarrow, shift right, "-\alpha"'] & \bullet \arrow[r, "Post_{D_{C_{B_A}}}"] \arrow[d, "\alpha"] & \bullet \arrow[d, "\alpha"] & \cdots & \text{timed automaton} \\
Init_{C_{B_A}} \arrow[d, leftrightarrow, "\beta"] & \cdots & \bullet \arrow[d, leftrightarrow, "\beta"] & \bullet \arrow[l, "Pre_{C_{-B_{-A}}}"'] \arrow[d, leftrightarrow, "\beta"] & \bullet \arrow[l, "Pre_{C_{-B_{-A}}}"'] \arrow[r, "Post_{C_{B_A}}"] \arrow[d, leftrightarrow, "\beta"] & \bullet \arrow[r, "Post_{C_{B_A}}"] \arrow[d, leftrightarrow, "\beta"] & \bullet \arrow[d, leftrightarrow, "\beta"] & \cdots & \text{initialized stopwatch automaton} \\
Init_{B_A} \arrow[d, shift left, "\gamma"] \arrow[d, leftarrow, shift right, "-\gamma"'] & \cdots & \bullet \arrow[d, "\gamma"] & \bullet \arrow[l, "Pre_{-B_{-A}}"'] \arrow[d, "\gamma"] & \bullet \arrow[l, "Pre_{-B_{-A}}"'] \arrow[r, "Post_{B_A}"] \arrow[d, shift left, "\gamma"] \arrow[d, leftarrow, shift right, "-\gamma"'] & \bullet \arrow[r, "Post_{B_A}"] \arrow[d, "\gamma"] & \bullet \arrow[d, "\gamma"] & \cdots & \text{initialized singular automaton} \\
Init_A & \cdots & \bullet & \bullet \arrow[l, "Pre_A"] & Z \arrow[l, "Pre_A"] \arrow[r, "Post_A"'] & \bullet \arrow[r, "Post_A"'] & \bullet & \cdots & \text{initialized rectangular automaton}
\end{tikzcd}$$

Figure 12: Putting together the commuting diagrams

For a map $\kappa : Q_{A_1} \rightarrow Q_{A_2}$ from the states of A_1 to the states of A_2, define the relation $\hat{\kappa} \subset Reach(A_1) \times Reach(A_2)$ such that $(q, r) \in \hat{\kappa}$ iff $r = \kappa(q)$. In this way, we obtain the relations $\hat{\alpha}$ and $\hat{\beta}$. For a map $\kappa : Q_{A_1} \rightarrow 2^{Q_{A_2}}$ from the states of A_1 to zones of A_2, define the relation $\hat{\kappa} \subset Reach(A_1) \times Reach(A_2)$ such that $(q, r) \in \hat{\kappa}$ iff $r \in \kappa(q)$. In this way, we obtain the relation $\hat{\gamma}$. The next proposition follows from Lemma 3.2 in the case of $\hat{\alpha}$, from Lemma 3.4 in the case of $\hat{\beta}$, and from Lemmas 3.13, 3.14, and 3.17 in the case of $\hat{\gamma}$.

Proposition 3.29 *For every initialized stopwatch automaton C with ε moves, the relation $\hat{\alpha}$ is a timed bisimulation between C and the timed automaton D_C. For every initialized singular automaton B with ε moves, the relation $\hat{\beta}$ is a timed bisimulation between B and the initialized stopwatch automaton D_C. For every initialized rectangular automaton A, the relation $\hat{\gamma}$ is a timed forward simulation of A by B_A, and $\hat{\gamma}^{-1}$ is a timed backward simulation of B_A, restricted to its upper-half space, by A.*

Simulation relations compose, as summarized in Figures 11 and Figure 12. In particular, the timed automaton $D_{C_{B_A}}$ timed forward simulates the initialized rectangular automaton A via the relation $\hat{\alpha} \circ \hat{\beta} \circ \hat{\gamma}$, and A timed backward simulates $D_{C_{B_A}}$ via $\hat{\gamma}^{-1} \circ \hat{\beta} \circ \hat{\alpha}$. It is not difficult to check that A does not timed forward simulate B_A, and B_A does not timed backward simulate A [Kop96].

3.5 Symbolic Reachability Analysis

Consider an n-dimensional rectangular automaton A, and a rectangular zone Z_f of A. To solve the reachability problem for A and Z_f symbolically, by computing with multirectangular zones, we may attempt to compute the sequence *Init*, *Post*(*Init*), *Post*^{2}(*Init*), ... of zones, until either the intersection with Z_f is nonempty, or a fixpint of *Post* is reached within a finite number of steps (that fixpoint, then, is *Reach*(A)). This procedure, which we call the *symbolic execution* of A [ACH+95], will terminate if the zone Z_f is reachable or if there is a natural number $i \in \mathbb{N}$ such that $Reach(A) = Post^i(Init)$, but it will not terminate if Z_f is not reachable and no such i exists. Symbolic execution therefore constitutes a semidecision procedure for the reachability problem of rectangular automata. The procedure has been implemented in the automatic verification tool

HyTech [HHWT97], and successfully applied to examples of practical interest [HHWT95, HW95, NS95, Cor96, HWT96, SMF97].

While the reachability problem is decidable for all initialized rectangular automata, even for timed automata symbolic execution does not always terminate. To see this, consider the 2D timed automaton $\hat{B}$ from Figure 13, with first coordinate c and second coordinate d. The only vertex of $\hat{B}$ has the invariant region $[0, \infty) \times [0, 1]$. For each $i \geq 1$, $Post_{\hat{B}}^{2i-1}(Init_{\hat{B}}) = \{(v, (x, y)) \mid 0 \leq x \leq i \text{ and } y = x - \lfloor x \rfloor\}$. Hence the fixpoint computation does not converge. To blame is the unbounded invariant region. This is because symbolic execution is known to terminate for every timed automaton with bounded invariant regions [HNSY94] (where A has bounded invariant regions if for every vertex v of A, $inv(v)$ is bounded). It follows that symbolic execution terminates also for initialized rectangular automata with bounded invariant regions. This is the content of the next proposition.

Proposition 3.30 *For every initialized rectangular automaton A with ε moves and bounded invariant regions, there is a natural number $i \in \mathbb{N}$ such that $Reach(A) = Post^i(Init)$.*

Proof. Consider an initialized rectangular automaton A with ε moves and bounded invariant regions. The construction of the singular automaton B_A from Section 3.2 can be modified in two respects. First, it is a simple matter to accommodate inherited ε edges from A. Second, B_A is equipped with bounded invariant regions as follows, by introducing additional ε edges. Whenever a lower-bound variable $b_{\ell(i)}$ falls to 1 below the lower boundary of the bounded invariant interval for a_i, the slope of $b_{\ell(i)}$ is changed to 0 via an ε edge. Upper-bound variables are treated symmetrically. These modifications do not affect the mapping γ and its properties. Now the proposition follows immediately from the relationships shown in Figure 12 and the statement of the proposition for timed automata. ∎

In this section, we remedy the problems arising from unbounded invariant regions by preprocessing the given automaton A. For initialized A, we construct in linear time an initialized rectangular automaton A_{bd} with ε moves such that (1) $Reach(A) \cap Z_f \neq \emptyset$ iff $Reach(A_{bd}) \cap Z_f \neq \emptyset$, and (2) there is a natural number $i \in \mathbb{N}$ such that $Reach(A_{bd}) = Post^i_{A_{bd}}(Init_{A_{bd}})$. The second condition implies that the symbolic execution of A_{bd} terminates, and the first condition implies that it gives the correct answer to the reachability problem for A. Consequently, the reachability problem for A can be solved, rather than by translating A into a timed automaton, by direct symbolic execution of A_{bd}. While translation doubles the dimension, the dimension of A_{bd} remains n, which alleviates a major practical bottleneck in the verification of hybrid systems [HHWT97].

To facilitate the proof of condition (2), we first introduce a third automaton A'_{bd}, which satifies both (1) and (2) but is exponentially larger than A. The automaton A'_{bd} will have bounded invariant regions, and therefore satisfy condition (2) by Proposition 3.30. For the remainder of this section, assume that the given automaton A is initialized, and that its variables are $a_1, \ldots, a_n$.

An exponential preprocessing step

We define an n-dimensional initialized rectangular automaton A'_{bd} with ε moves and bounded invariant regions, and a rectangular zone Z'_f of A'_{bd}, such that $Reach(A) \cap Z_f \neq \emptyset$ iff $Reach(A'_{bd}) \cap Z'_f \neq \emptyset$. Let k be 1 more than the largest rational constant that appears in the definitions of A and Z_f as a finite endpoint of an interval. Let g be 1 less than the smallest such constant. The idea is to truncate all invariant, preguard, and postguard regions of A by intersection with $[g, h]^n$. When a variable reaches the upper or lower boundary of the rectangular region $[g, h]^n$, we change

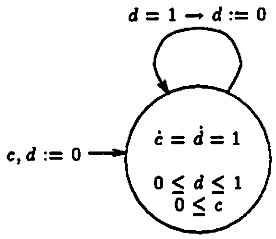

Figure 13: The timed automaton $\hat{B}$ for which $Reach(\hat{B}) \supsetneq Post^i(Init)$ for all $i \geq 0$

its slope to 0. The automaton A'_{bd} has the vertex set $V_{A'_{bd}} = V_A \times \{0,1,2\}^n$. Put $low = 0$, $ok = 1$, and $high = 2$, and let ok^n be the n-vector $(ok, ok, \ldots, ok)$. Each state $((v, \vec{\lambda}), \mathbf{x})$ of A'_{bd} is intended to represent all states of A of the form $(v, \mathbf{y})$ with $y_i \leq g$ if $\lambda_i = low$, and $y_i = x_i \in [g, h]$ if $\lambda_i = ok$, and $y_i \geq h$ if $\lambda_i = high$. All of these states will be shown equivalent for reachability purposes.

The remaining components of A'_{bd} are defined as follows. The observation alphabet of A'_{bd} is the same as for A. The initial, invariant, and flow functions of A'_{bd} are defined by

$$init_{A'_{bd}}(v, \vec{\lambda})_i = \begin{cases} init_A(v)_i \cap [g, h], & \text{if } \lambda_i = ok, \\ \emptyset, & \text{otherwise;} \end{cases}$$

$$inv_{A'_{bd}}(v, \vec{\lambda})_i = \begin{cases} [h, h], & \text{if } \lambda_i = high, \\ inv_A(v)_i \cap [g, h], & \text{if } \lambda_i = ok, \\ [g, g], & \text{if } \lambda_i = low; \end{cases}$$

$$flow_{A'_{bd}}(v, \vec{\lambda})_i = \begin{cases} flow_A(v)_i, & \text{if } \lambda_i = ok, \\ [0, 0], & \text{otherwise.} \end{cases}$$

For each edge $e = (v, w)$ of A, the automaton A'_{bd} has an edge $e' = ((v, ok^n), (w, ok^n))$ with $obs_{A'_{bd}}(e') = obs_A(e)$, $pre_{A'_{bd}}(e') = pre_A(e) \cap [g, h]^n$, $post_{A'_{bd}}(e') = post_A(e) \cap [g, h]^n$, and $jump_{A'_{bd}}(e') = jump_A(e)$. Define $trunc : \mathbb{R}^n \rightarrow [g, h]^n$ by $trunc(\mathbf{x})_i = g$ if $x_i < g$, $trunc(\mathbf{x})_i = x_i$ if $g \leq x_i \leq h$, and $trunc(\mathbf{x})_i = h$ if $x_i > h$. A rectangular region $R \subset \mathbb{R}^n$ is *gh-limited* if for every coordinate $i \in \{1, \ldots, n\}$, (1) either $\inf(R_i) = -\infty$ or $g + 1 \leq \inf(R_i) \leq h - 1$, and (2) either $\sup(R_i) = \infty$ or $g + 1 \leq \sup(R_i) \leq h - 1$. By definition of g and h, all initial, invariant, preguard, and postguard regions of A, as well as $[Z_f]^v$ for all vertices v of A, are gh-limited regions. The following fact ensures that the guards of A'_{bd} have the same effect as the guards of A.

Fact 3.31 *Let* $\mathbf{x} \in \mathbb{R}^n$, *and let* $R \subset \mathbb{R}^n$ *be a gh-limited rectangular region. Then* $\mathbf{x} \in R$ *iff* $trunc(\mathbf{x}) \in R \cap [g, h]^n$.

In addition to the edges inherited from A, the automaton A'_{bd} has ε edges for toggling the components of $\vec{\lambda}$. For every vertex v of A, every coordinate $i \in \{1, \ldots, n\}$, and every vector $\vec{\lambda}$ with $\lambda_i = ok$, there is an ε edge from $(v, \vec{\lambda})$ to $(v, \vec{\lambda}[\lambda_i := low])$, and another ε edge in the reverse direction, from $(v, \vec{\lambda}[\lambda_i := low])$ to $(v, \vec{\lambda})$, each annotated with the guarded command $a_i = g \rightarrow a_i := g$. Here $\vec{\lambda}[\lambda_i := low]$ stands for the vector that agrees with $\vec{\lambda}$ on all components except for the i-th component, which is low. The trivial assignment is for initialization. Similarly, there are ε edges from $(v, \vec{\lambda})$ to $(v, \vec{\lambda}[\lambda_i := high])$ and from $(v, \vec{\lambda}[\lambda_i := high])$ to $(v, \vec{\lambda})$, each annotated with the guarded command $a_i = h \rightarrow a_i := h$. This completes the definition of A'_{bd}. Note that the rectangular automaton A'_{bd} is initialized and has bounded invariant regions.

Define the function $\zeta : Q_A \to Q_{A'_{bd}}$ by $\zeta(v, \mathbf{x}) = ((v, ok^n), \mathit{trunc}(\mathbf{x}))$, and extend ζ to zones in the natural way. Notice that $\mathit{Init}_{A'_{bd}} = \zeta(\mathit{Init}_A)$. Define $Z'_f = \zeta(Z_f)$.

Lemma 3.32 *Let A be an initialized rectangular automaton, and let Z_f be a rectangular zone of A. Then $\mathit{Reach}(A) \cap Z_f \neq \emptyset$ iff $\mathit{Reach}(A'_{bd}) \cap Z'_f \neq \emptyset$.*

Proof. It suffices to prove the lemma for one-dimensional A. For the "only if" direction, we show that for all labels $\pi \in E_A \cup [\mathbb{R}_{\geq 0}]$, if $q \xrightarrow{\pi}_A r$ then $\zeta(q) \xrightarrow{\pi}_{A'_{bd}} \zeta(r)$. For time steps, suppose that $(v, x^1) \xrightarrow{t}_A (v, x^2)$ for $t > 0$. Then $(x^2 - x^1)/t \in \mathit{flow}_A(v)$. If $g \leq x^1, x^2 \leq h$, then $\mathit{trunc}(x^1) = x^1$ and $\mathit{trunc}(x^2) = x^2$. Since $\mathit{flow}_{A'_{bd}}(v, ok) = \mathit{flow}_A(v)$, we have $\zeta(v, x^1) \xrightarrow{t}_{A'_{bd}} \zeta(v, x^2)$. Now suppose that $x^1 \leq g \leq h \leq x^2$. Then there exist three durations $t_1, t_2, t_3 \in \mathbb{R}_{\geq 0}$ such that $t = t_1 + t_2 + t_3$ and $(v, x^1) \xrightarrow{t_1}_A (v, g) \xrightarrow{t_2}_A (v, h) \xrightarrow{t_3}_A (v, x^2)$. In this case, $\mathit{trunc}(x^1) = g$ and $\mathit{trunc}(x^2) = h$. Since $\mathit{flow}_{A'_{bd}}(v, ok) = \mathit{flow}_A(v)$, we have

$$((v, ok), \mathit{trunc}(x^1)) \xrightarrow{\varepsilon}_{A'_{bd}} ((v, low), g) \xrightarrow{t_1}_{A'_{bd}} ((v, low), g) \xrightarrow{\varepsilon}_{A'_{bd}} ((v, ok), g) \xrightarrow{t_2}_{A'_{bd}} ((v, ok), h) \xrightarrow{\varepsilon}_{A'_{bd}}$$
$$((v, high), h) \xrightarrow{t_3}_{A'_{bd}} ((v, high), h) \xrightarrow{\varepsilon}_{A'_{bd}} ((v, ok), \mathit{trunc}(x^2)).$$

Again $\zeta(v, x^1) \xrightarrow{t}_{A'_{bd}} \zeta(v^2, x^2)$. Other positions of x^1 and x^2 relative to g and h are handled similarly. For edge steps, the key fact is that all preguard and postguard regions are gh-limited. Suppose that $(v^1, x^1) \xrightarrow{e}_A (v^2, x^2)$ for $e \in E_A$. Then $x^1 \in \mathit{pre}_A(e)$ and $x^2 \in \mathit{post}_A(e)$, and so by Fact 3.31, $\mathit{trunc}(x^1) \in \mathit{pre}_{A'_{bd}}(e')$ and $\mathit{trunc}(x^2) \in \mathit{post}_{A'_{bd}}(e')$, where e' is the edge of A'_{bd} inherited from e. In addition, $\mathit{jump}_{A'_{bd}}(e') = \mathit{jump}_A(e)$, and so $\zeta(v^1, x^1) \xrightarrow{e'}_{A'_{bd}} \zeta(v^2, x^2)$.

For the "if" direction we apply a more global argument. Suppose that

$$((v^0, ok), y^0) \xrightarrow{\pi_1}_{A'_{bd}} ((v^1, ok), y^1) \xrightarrow{\pi_2}_{A'_{bd}} \cdots \xrightarrow{\pi_m}_{A'_{bd}} ((v^m, ok), y^m),$$

where $((v^0, ok), y^0) \in \mathit{Init}_{A'_{bd}}$ and $((v^m, ok), y^m) \in Z'_f$. We find $x^0, \ldots, x^m$ such that

$$(v^0, x^0) \xrightarrow{\pi_1}_A (v^1, x^1) \xrightarrow{\pi_2}_A \cdots \xrightarrow{\pi_m}_A (v^m, x^m)$$

and for each $i \in \{0, \ldots, m\}$, $\mathit{trunc}(x^i) = y^i$. Since A is initialized, it suffices to assume that each $\pi_i \in E_{A'_{bd}}$ has $\mathit{jump}(\pi_i) = \emptyset$; otherwise we string together the solutions obtained for the segments between discontinuous jumps. Consequently, $\mathit{flow}_A(v^i) = \mathit{flow}_A(v^j)$ for all $i, j \in \{0, \ldots, m\}$. By Fact 3.31, it suffices to assume that $\pi_i \in \mathbb{R}_{\geq 0}$ for all i, that is, all m steps are time steps. Let flow be the common value of the flow region $\mathit{flow}_A(v^i)$, for $i \in \{0, \ldots, m\}$. If $0 \in \mathit{flow}$, then

$$(v^0, y^0) \xrightarrow{\pi_1}_A (v^1, y^1) \xrightarrow{\pi_2}_A \cdots \xrightarrow{\pi_m}_A (v^m, y^m),$$

so putting $x^i = y^i$ for each i, we are finished, because $y^0 \in \mathit{Init}_A$ and $y^1 \in Z_f$ by Fact 3.31. Now suppose that $\mathit{flow} \subset (0, \infty)$. Here the most interesting case is given by $y^0 = g$ and $y^m = h$, for $g < h$. In this case, there exist $0 \leq j_1 \leq j_2 < m$ such that $y^i = g$ for $i \leq j_1$, and $g < y^i < h$ for $j_1 < i \leq j_2$, and $y^i = h$ for $i > j_2$. We put $x^i = y^i$ for $j_1 < i \leq j_2$. To set the x^i for $i > j_2$, we need only determine a suitable slope. Let k_2 be such that for some $h' \geq h$, we have $k_2 = (h' - y^{j_2})/\pi_{j_2+1} \in \mathit{flow}$. Such a $k_2 > 0$ exists because $((v^{j_2}, ok), y^{j_2}) \xrightarrow{\pi_{j_2+1}}_{A'_{bd}} ((v^{j_2+1}, ok), h)$. Put $x^i = y^{j_2} + k_2 \cdot (i - j_2)$ for each $i > j_2$. Then $(v^i, x^i) \xrightarrow{\pi_{i+1}}_A (v^{i+1}, x^{i+1})$ for all $i \in \{j_2, j_2 + 1, \ldots, m - 1\}$. It remains to set the x^i for $i \leq j_1$, which is done in the same way. Since $((v^{j_1}, ok), g) \xrightarrow{\pi_{j_1+1}}_A ((v^{j_1+1}, ok), y^{j_1+1})$, there exists a $k_1 \in \mathit{flow}$ such that for some $g' \leq g$, we have $k_1 = (y^{j_1+1} - g')/\pi_{j_1+1} \in \mathit{flow}$. For each $i \in \{0, \ldots, j_1\}$, put $x^i = y^{j_1+1} - k_1 \cdot (j_1 + 1 - i)$. Then $(v^i, x^i) \xrightarrow{\pi_{i+1}}_A (v^{i+1}, x^{i+1})$ for all $i \in \{0, \ldots, j_1\}$, and we are finished. All other cases are handled in a similar fashion. ■

A linear preprocessing step

The automaton A'_{bd} uses the discrete part of the state to store information about variables whose values are too small or too large to be relevant. This causes the state space of A'_{bd} to be exponentially larger than the state space of A. We now define an initialized rectangular automaton A_{bd}, with the same dimension and state space as A, which uses the continuous part of the state for the same purpose. Instead of stopping a variable when it reaches g (resp. h), the automaton A_{bd} allows a nondeterministic jump to any value below g (resp. above h). Formally, A_{bd} is identical to A, except for $2n \cdot |V_A|$ additional ε edges. For every vertex v of A, and every coordinate $i \in \{1, \ldots, n\}$, the automaton A_{bd} has two ε edges from v to v, which are annotated respectively with the two guarded commands $a_i \leq g \rightarrow a_i :\in (-\infty, g]$ and $a_i \geq h \rightarrow a_i :\in [h, \infty)$.

The following two lemmas prove that the simple modification A_{bd} can be used to decide reachability on A by symbolic execution. The connection between A_{bd} and A is established via the automaton A'_{bd}. The first lemma implies that A_{bd} is timed bisimilar to A'_{bd}.

Lemma 3.33 *For all states q and r of the rectangular automaton A_{bd}, and every label $\tau \in \Sigma \cup \mathbb{R}_{\geq 0}$, we have $q \xrightarrow{\tau}_{A_{bd}} r$ iff $\zeta(q) \xrightarrow{\tau}_{A'_{bd}} \zeta(r)$.*

Proof. For $\tau \in \Sigma$, the statement of the lemma follows from Fact 3.31. For $\tau \in \mathbb{R}_{\geq 0}$, it suffices to prove the result for one-dimensional A. Suppose that $(v, x^1) \xrightarrow{\tau}_{A_{bd}} (v, x^2)$. If $g < x^1, x^2 < h$, then immediately $\zeta(v, x^1) = ((v, ok), x^1) \xrightarrow{\tau}_{A'_{bd}} ((v, ok), x^2) = \zeta(v, x^2)$. The most interesting case is $x^1 < g < h < x^2$. In this case, there exists a duration $t \leq \tau$ such that $(h - g)/t \in \mathit{flow}_A(v)$. Therefore

$$\zeta(v, x^1) = ((v, ok), g) \xrightarrow{\varepsilon}_{A'_{bd}} ((v, low), g) \xrightarrow{\tau - t}_{A'_{bd}} ((v, low), g) \xrightarrow{\varepsilon}_{A'_{bd}} ((v, ok), g) \xrightarrow{t}_{A'_{bd}} ((v, ok), h) = \zeta(v, x^2).$$

Similar arguments apply to all relative positions of x^1, x^2, g, and h. The reverse implication follows from the ε edges of A_{bd}. ■

Let $Q_{ok} \subset Q_{A'_{bd}}$ be the set of states of A'_{bd} that have the form $((v, ok^n), \mathbf{x})$. Then ζ is an onto function from $Q_{A_{bd}}$ to $Q_{A'_{bd}}$. From Lemma 3.33, it follows that for every zone Z of A_{bd}, $\mathit{Post}_{A'_{bd}}(\zeta(Z)) \cap Q_{ok} = \zeta(\mathit{Post}_{A_{bd}}(Z))$. Since $\mathit{Init}_{A'_{bd}} = \zeta(\mathit{Init}_{A_{bd}})$, we conclude that $\mathit{Reach}(A'_{bd}) \cap Q_{ok} = \zeta(\mathit{Reach}(A_{bd}))$. Another consequence of Lemma 3.33 is the next lemma, which implies that the symbolic execution of A_{bd} must terminate within at most one more step than the symbolic execution of A'_{bd} (whose bounded invariant regions guarantee termination).

Lemma 3.34 *For all zones Z of the rectangular automaton A, if $\mathit{Post}_{A'_{bd}}(\zeta(Z)) = \zeta(Z)$, then $\mathit{Post}^2_{A_{bd}}(Z) = \mathit{Post}_{A_{bd}}(Z)$.*

Proof. Consider a zone Z of A such that $\mathit{Post}_{A'_{bd}}(\zeta(Z)) = \zeta(Z)$. We assume that $q \xrightarrow{\tau}_{A_{bd}} r$ for two states $q \in \mathit{Post}_{A_{bd}}(Z)$ and $r \notin \mathit{Post}_{A_{bd}}(Z)$, and show a contradiction. From Lemma 3.33 it follows that for all states r' with $\zeta(r') = \zeta(r)$, also $r' \notin \mathit{Post}_{A_{bd}}(Z)$. On the other hand, $\zeta(q) \in \zeta(\mathit{Post}_{A_{bd}}(Z)) \subset \mathit{Post}_{A'_{bd}}(\zeta(Z)) = \zeta(Z)$, using Lemma 3.33 a second time. Since $r \in \mathit{Post}_{A_{bd}}(q)$, we have $\zeta(r) \in \mathit{Post}_{A'_{bd}}(\zeta(Z)) \cap Q_{ok} = \zeta(\mathit{Post}_{A_{bd}}(Z))$ by a third and fourth application of Lemma 3.33. So there exists a state $r' \in \mathit{Post}_{A_{bd}}(Z)$ with $\zeta(r') = \zeta(r)$, which gives us a contradiction. ■

From Lemmas 3.32 and 3.33 it follows that reachability in A is equivalent to reachability in A_{bd}: we have $\mathit{Reach}(A) \cap Z_f \neq \emptyset$ iff $\mathit{Reach}(A'_{bd}) \cap Z'_f \neq \emptyset$ iff $\zeta(\mathit{Reach}(A_{bd})) \cap \zeta(Z_f) \neq \emptyset$ iff $\mathit{Reach}(A_{bd}) \cap Z_f \neq \emptyset$

(the final equivalence follows from Fact 3.31, because all regions of Z_f are gh-limited). From Proposition 3.30, Lemma 3.33, and Lemma 3.34 it follows that the symbolic execution of A_{bd} terminates.

Theorem 3.35 *Let A be an initialized rectangular automaton, and let Z_f be a rectangular zone of A. Then there exists a rectangular automaton A_{bd}, obtained from A by adding ε moves, such that (1) $Reach(A) \cap Z_f \neq \emptyset$ iff $Reach(A_{bd}) \cap Z_f \neq \emptyset$, and (2) there is a natural number $i \in \mathbb{N}$ with $Reach(A_{bd}) = Post^i_{A_{bd}}(Init_{A_{bd}})$.*

4 Undecidability

In Section 3, we showed that the initialized rectangular automata form a decidable class of hybrid automata. In this section, we show that they form a maximal such class. We proceed in two steps. First, we show that without initialization, even a single two-slope variable leads to an undecidable reachability problem. Second, we show that the rectangularity of the model must remain inviolate. Any coupling between coordinates, such as comparisons between variables, brings undecidability already with a single nonclock variable. (Timed automata, which have only clock variables, remain decidable in the presence of variable comparisons [AD94].) A main consequence is the undecidability of compact automata with clocks and one stopwatch, which are of interest for the specification of duration properties [KPSY93].

An n-dimensional rectangular automaton A is *simple* if it meets the following restrictions.

1. Exactly one variable of A is *not* a clock.

2. The automaton A has only one initial state q_0, and q_0 has the form $(v, (0, 0, \ldots, 0))$.

3. For every edge e of A, and every coordinate $i \in \{1, \ldots, n\}$, if $i \in jump(e)$ then $post(e)_i = [0, 0]$, and if $i \notin jump(e)$ then $post(e)_i = pre(e)_i$.

4. For every vertex v of A, the invariant region $inv(v)$ and the flow region $flow(v)$ are compact (by restriction (2), the initial region $init(v)$ is compact as well). For every edge e of A, the preguard region $pre(e)$ is compact (by restriction (3), the postguard region $post(e)$ is compact as well).

5. The observation alphabet of A is a one-letter alphabet, and the invariant function inv of A is constant. (One can require that A has the trivial invariant function $\lambda v. \mathbb{R}^n$, with only minor modifications of our proofs.)

The automaton A is *m-simple* if it meets restrictions (2)–(4), and exactly m variables of A are not clocks.

We use simple automata for our undecidability results. Restrictions (2) and (3) ensure that every simple automaton has deterministic jumps, which eliminates the nondeterminism of jumps as a possible source of undecidability. Many limited decidability results are based on a technique, called digitization, which discretizes time steps with noninteger durations [HMP92, BES93, BER94, PV94]. Since the digitization technique requires closed guard and invariant regions, restriction (4) implies that the technique does not generalize beyond very special cases. Many limited decidability results apply to automata with a single stopwatch [BES93, KPSY93, BER94, MV94, BR95, ACH97]. Restriction (1) implies that these results do not generalize either.

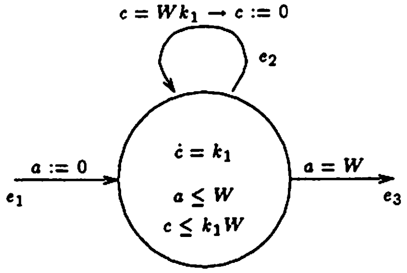

Figure 14: Wrapping lemma: the skewed clock c retains its entry value upon exit

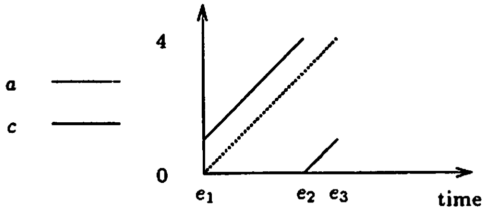

Figure 15: Proof of the wrapping lemma for slope 1

All of our undecidability proofs are reductions from the halting problem for two-counter machines to the reachability problem for simple rectangular automata. A two-counter machine M consists of a finite control and two unbounded nonnegative integer variables called counters. Initially both counter values are 0. Three types of instructions are used: branching based upon whether a specific counter has the value 0, incrementing a counter, and decrementing a counter (which leaves unchanged a counter value of 0). When a specified halting location is reached, the machine halts. In our reductions, the finite control of M is encoded in the finite control of a simple rectangular automaton A_M; in particular, there is a vertex v such that that M halts iff the zone $\{v\} \times inv(v)$ is reachable for A_M. Each counter is encoded by a clock of A_M, and we supply widgets for performing the operations that correspond to incrementing or decrementing a counter. Typically, the counter value u corresponds to the clock value $k_1 \cdot (\frac{k_2}{k_1})^u$, where k_1 and k_2 are the slopes of a two-slope variable of A_M, with k_1 being the larger. When $k_1 = 2k_2$, decrementing (resp. incrementing) a counter corresponds to doubling (resp. halving) the value of the corresponding clock. Notice that since $k_1 > k_2$, it is the density of the continuous domain, rather than its infinite extent, that is used to encode the potentially unbounded counter values.

4.1 Uninitialized Automata

We show that initialization is necessary for a decidable reachability problem.

Theorem 4.1 *For every two slopes $k_1, k_2 \in \mathbb{Q}$ with $k_1 \neq k_2$, the reachability problem is undecidable for simple rectangular automata with a two-slope variable of slopes k_1 and k_2.*

We first prove three lemmas that are basic to all of our undecidability proofs. Let W be a positive rational number. A simple rectangular automaton A is *W-wrapping* if its invariant function is defined as follows.

- For every variable a of A that is a clock, and every vertex v of A, $inv(v)(a) = [0, W]$.

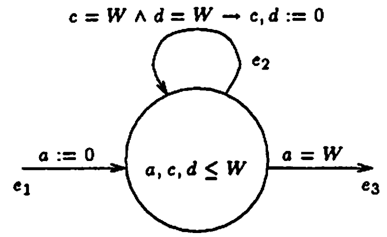

Figure 16: Equality lemma: testing if $c = d$

- If b is the nonclock variable of A, and b takes only nonnegative slopes (i.e., $\mathit{flow}(v)(b) \subset [0, \infty)$ for all v), then for each vertex v of A, $\mathit{inv}(v)(b) = [0, W \cdot \max\{\sup(\mathit{flow}(w)(b)) \mid w \in V\}]$.
- If b is the nonclock variable of A, and b takes only nonpositive slopes, then for each vertex v of A, $\mathit{inv}(v)(b) = [W \cdot \min\{\inf(\mathit{flow}(w)(b)) \mid w \in V\}, 0]$.
- If b is the nonclock variable of A, and b takes both positive and negative slopes, then for each vertex v of A, $\mathit{inv}(v)(b) = [W \cdot \min\{\inf(\mathit{flow}(w)(b)) \mid w \in V\}, W \cdot \max\{\sup(\mathit{flow}(w)(b)) \mid w \in V\}]$.

A *W-wrapping edge* for a clock a and a vertex v is an edge from v to itself that is annotated with the guarded command $a = W \rightarrow a := 0$. A *W-wrapping edge* for a nonclock variable b and a vertex v with $\mathit{flow}(v)(b) = [k, k]$ is an edge from v to itself that is annotated with the guarded command $b = k \cdot W \rightarrow b := 0$. The invariant of a wrapping automaton forces wrapping edges to be taken when they are enabled. We use wrapping to simulate discrete events by continuous rounds taking W (or some multiple thereof) units of time. The wrapping edges ensure that variables take the same values at the beginning and end of a round, unless they are explicitly reassigned by a nonwrapping edge. This is the content of the wrapping lemma. A similar wrapping technique can be found in [Cer92].

In figures of simple automata, we use the following conventions. First, all variables whose slopes are not listed are clocks, i.e., they have slope 1. Second, wrapping conditions are left implicit; in particular, we omit invariants from every figure after those regarding the three basic lemmas, and we omit wrapping edges beginning with Figure 20.

Wrapping lemma. *Let W be a positive rational number. Let $k_1 \in \mathbb{Q}$, and consider the simple W-wrapping automaton fragment of Figure 14. Suppose that the value of c is x when the edge e_1 is traversed, where $0 < x < k_1 \cdot W$ if $k_1 > 0$, and $k_1 \cdot W < x < 0$ if $k_1 < 0$. Then the next time e_3 is traversed, the value of c is again x.*

Proof. Figure 15 contains a time portrait that illustrates the proof for $W = 4$ and $k_1 = 1$. The markings e_1, e_2, and e_3 along the time axis show at which points these edges are traversed. We give the proof for $k_1 > 0$. In order for e_3 to be traversed in the future, the following series of events must occur: (1) e_1 is traversed; (2) exactly $(W \cdot k_1 - x)/k_1$ time units elapse, after which c has the value $W \cdot k_1$, and a has the value $(W \cdot k_1 - x)/k_1$; (3) the wrapping edge e_2 is traversed, after which c has the value 0, and a has the value $(W \cdot k_1 - x)/k_1$; (4) exactly $W - (W \cdot k_1 - x)/k_1 = x/k_1$ time units elapse, after which a has the value W and c has the value x. ■

By allowing clocks c and d to wrap only simultaneously, we can check if the two clocks have the same value.

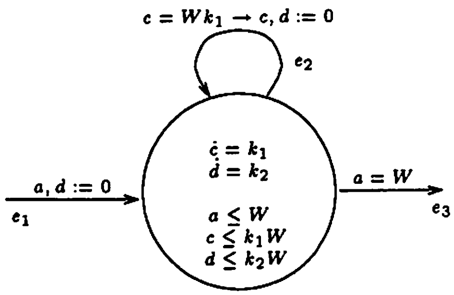

Figure 17: Assignment lemma: performing the assignment $d := \frac{k_2}{k_1} \cdot c$

Equality lemma. *Let W be a positive rational number. Consider the simple W-wrapping automaton fragment of Figure 16, in which all variables are clocks. Suppose that the value of c is x and the value of d is y when the edge e_1 is traversed, where $0 < x, y < W$. Then the edge e_3 can be traversed later iff $x = y$. Furthermore, the next time e_3 is traversed, the value of both c and d is x (hich is equal to y).*

Similarly, by assigning skewed clock d to 0 at the same time as wrapping skewed clock c to 0, we perform the assignment $d := \frac{k_2}{k_1} \cdot c$, where $\dot{c} = k_1$ and $\dot{d} = k_2$.

Assignment lemma. *Let W be a positive rational number. Let $k_1, k_2 \in \mathbb{Q}$ with $k_1 \neq 0$, and consider the simple W-wrapping automaton fragment of Figure 17.*[1] *Suppose that the value of c is x when the edge e_1 is traversed, where $0 < x < k_1 \cdot W$ if $k_1 > 0$, and $k_1 \cdot W < x < 0$ if $k_1 < 0$. Then the next time e_3 is traversed, the value of c is again x and the value of d is $\frac{k_2}{k_1} \cdot x$.*

Now we are ready to prove Theorem 4.1.

Proof of Theorem 4.1. We reduce the halting problem for two-counter machines to the reachability problem for simple wrapping automata with a two-slope variable taking slopes k_1 and k_2. Let M be a two-counter machine with counters C and D. We describe the construction of a simple wrapping automaton A_M with the following six variables: a, b, b', c, and d are clocks, and z is a two-slope variable with slopes k_1 and k_2. The values of the counters C and D are encoded in the values of the clocks c and d, respectively.

Case $k_1 > k_2 > 0$ or $k_1 < k_2 < 0$. The automaton A_M is W-wrapping, where W may be chosen to be any number larger than $|k_1|$. We encode counter value u by clock value $|k_1| \cdot (\frac{k_2}{k_1})^u$. The relationships $c = |k_1| \cdot (\frac{k_2}{k_1})^C$ and $d = |k_1| \cdot (\frac{k_2}{k_1})^D$ hold when $a = 0$ or $a = W$, except when more than one time interval of duration W is needed to simulate one computation step of M. The initialization of C and D is implemented by the initial vertex of A_M and an outgoing edge e with the preguard $pre(e)(c) = pre(e)(d) = [|k_1|, |k_1|]$. The test $C = 0$ is implemented by two edges e_1 and e_2, where $pre(e_1)(c) = [k_1, k_1]$ (corresponding to $C = 0$) and $pre(e_2)(c) = [0, k_2]$ (corresponding to $C \neq 0$). Decrementing a counter corresponds to first checking if its value is 0 as above, and if not, then multiplying the corresponding clock value by $\frac{k_1}{k_2}$. This is implemented by concatenating two assignment lemma constructions, as shown in Figure 18 for counter C. In the first, $\dot{z} = k_1$; it performs $z := k_1 \cdot c$. In the second, $\dot{z} = k_2$; it performs $c := \frac{1}{k_2} \cdot z$. The bottom portion of Figure 18 shows a time portrait of the decrementation fragment for $W = 4$, $k_1 = 2$, and $k_2 = 1$. Notice that

[1] Figure 17 assumes that $k_1, k_2 \geq 0$. If $k_1 < 0$, replace $c \leq k_1 \cdot W$ by $c \geq k_1 \cdot W$; if $k_2 < 0$, replace $d \leq k_1 \cdot W$ by $d \geq k_1 \cdot W$.

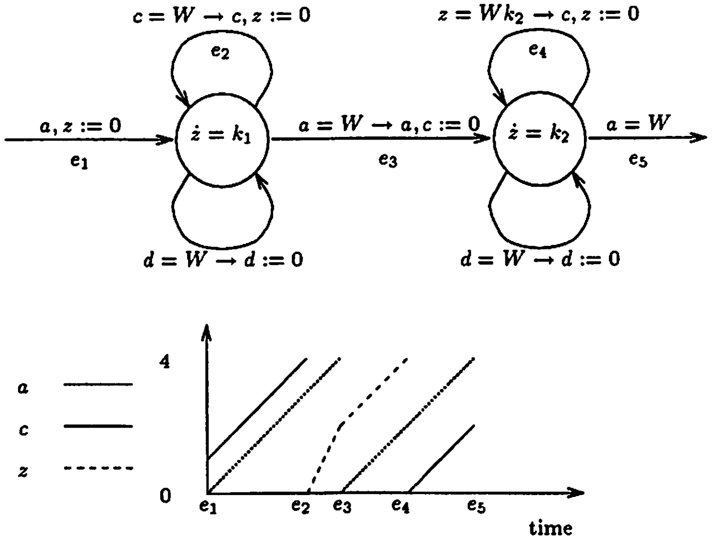
$c = W \rightarrow c, z := 0$
e_2
$z = Wk_2 \rightarrow c, z := 0$
e_4
$a, z := 0$
e_1
$\dot{z} = k_1$
$a = W \rightarrow a, c := 0$
e_3
$\dot{z} = k_2$
$a = W$
e_5
$d = W \rightarrow d := 0$
$d = W \rightarrow d := 0$
4
0
a
c
z
e_1
e_2
e_3
e_4
e_5
time

Figure 18: Counter decrement: multiplying c by $\frac{k_1}{k_2}$ using the two-slope variable z

the value of d, which represents the unchanged counter D, is not affected by the decrementation fragment. Incrementing a counter corresponds to multiplying the corresponding clock value by $\frac{k_2}{k_1}$. This is done by reversing the two assignments, as shown in Figure 19 for counter C. First $z := k_2 \cdot c$ is performed, and then $c := \frac{1}{k_1} \cdot z$. The bottom portion of Figure 19 shows a time portrait of the incrementation fragment for $W = 4$, $k_1 = 2$, and $k_2 = 1$.

Each instruction of M can be implemented as outlined above, with the terminal edge of the widget for instruction i coinciding with the initial edge of the widget for instruction $i+1$ (with the obvious modifications for the branch instruction). It follows that the vertex of A_M that corresponds to the halting location of M is reachable iff M halts.

Case $k_1 \neq 0$ and $k_2 = 0$. The construction is insensitive to the sign of k_1. We use the wrapping constant $W = 4$ and encode counter value u by clock value 2^{1-u}. Initialization and test for zero are implemented easily. Decrementing a counter corresponds to doubling the corresponding clock. The doubling procedure is shown in Figure 20 for the clock c; in this and all remaining figures, we omit the wrapping edges required to maintain the value of d. The idea is to perform $z := k_1 \cdot c$ using the assignment lemma, then to put $\dot{z} = 0$ until c reaches W again, and then to put $\dot{z} = k_1$ so that when a reaches W, we have $z = 2k_1 \cdot x$, where x is the original value of c. Finally, we perform $c := \frac{1}{k_1} \cdot z$ with the assignment lemma. The bottom portion of Figure 20 a time portrait for $k_1 = 2$. Incrementing a counter corresponds to halving the corresponding clock, which can be done with two auxiliary clocks b and b'. To halve the value of c, first a value is nondeterministically guessed in b. Then $b' := 2b$ is performed using the above doubling procedure. Then $c = b'$ is checked by the equality lemma, and if this succeeds, then $c := b$ is performed using the assignment lemma.

Case $k_1 > 0 > k_2$. First suppose that $|k_2| < k_1$. The wrapping constant W can be any number larger than k_1. We encode counter value u by clock value $k_1 \cdot (\frac{|k_2|}{k_1})^u$. Now we need two synchronization clocks, a and b. Clock c is synchronized with a, and clock d is synchronized with b. The relationship $c = k_1 \cdot (\frac{|k_2|}{k_1})^C$ holds when $a = 0$ or $a = W$, and the relationship $d = k_1 \cdot (\frac{|k_2|}{k_1})^D$ holds when $b = 0$ or $b = W$. To multiply c by $\frac{k_1}{|k_2|}$, we first perform $z := k_1 \cdot c$ and reset c to 0. Then we

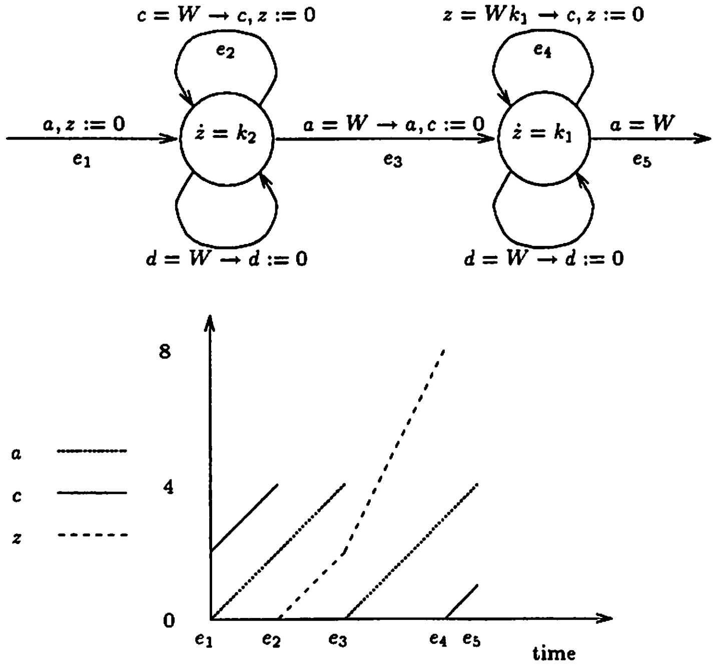
$c = W \rightarrow c, z := 0$
e_2
$z = Wk_1 \rightarrow c, z := 0$
e_4
$a, z := 0$
e_1
$\dot{z} = k_2$
$a = W \rightarrow a, c := 0$
e_3
$\dot{z} = k_1$
$a = W$
e_5
$d = W \rightarrow d := 0$
$d = W \rightarrow d := 0$
8
4
0
a
c
z
e_1
e_2
e_3
e_4
e_5
time

Figure 19: Counter increment: multiplying c by $\frac{k_2}{k_1}$ using the two-slope variable z

put $\dot{z} = k_2$, and when z reaches 0, we reset a to 0. At this point $c = \frac{k_1}{|k_2|} \cdot x$, where x is the original value of c. See Figure 21. The bottom portion of the figure shows a time portrait for $W = 4$,

$k_1 = 2$, and $k_2 = -1$. To multiply c by $\frac{|k_2|}{k_1}$, simply reverse the slopes of z. Specifically, perform $z := k_2 \cdot c$, reset c to 0, then put $\dot{z} = k_1$, and when z reaches 0, reset a to 0.

If $|k_2| > k_1$, we use a wrapping constant larger than $|k_2|$ and encode counter value u by clock value $|k_2| \cdot (\frac{k_1}{|k_2|})^u$. This simply switches the roles of multiplying by $\frac{k_1}{|k_2|}$ and multiplying by $\frac{|k_2|}{k_1}$.

Finally, suppose that $k_2 = -k_1$. In this case we use the wrapping constant 4 and encode counter value u by clock value clock value 2^{1-u}. Again we use separate synchronization clocks a and b for c and d. To double c, perform $z := k_1 \cdot c$, and then put $\dot{z} = -k_1$, resetting a when z reaches 0. See Figure 22, which gives the construction, and also a time portrait for $k_1 = 3$. Halving c is done by nondeterministically guessing the midpoint of the interval of time between $c = 4$ and $a = 4$. The guess is checked by starting z at value 0 when c reaches 4, keeping z at slope k_1 for the first half of the interval, and at slope $-k_1$ for the second half. If z returns to 0 at the same instant that a reaches 4, the guess was correct. See Figure 23, which gives the construction, and a time portrait for $k_1 = 5$. ■

4.2 Generalized Automata

A slight generalization of the invariant, flow, preguard, postguard, or jump function leads to the undecidability of rectangular automata, even under the stringent restrictions of simplicity and initialization. For the remainder of this section, fix an n-dimensional rectangular automaton A.

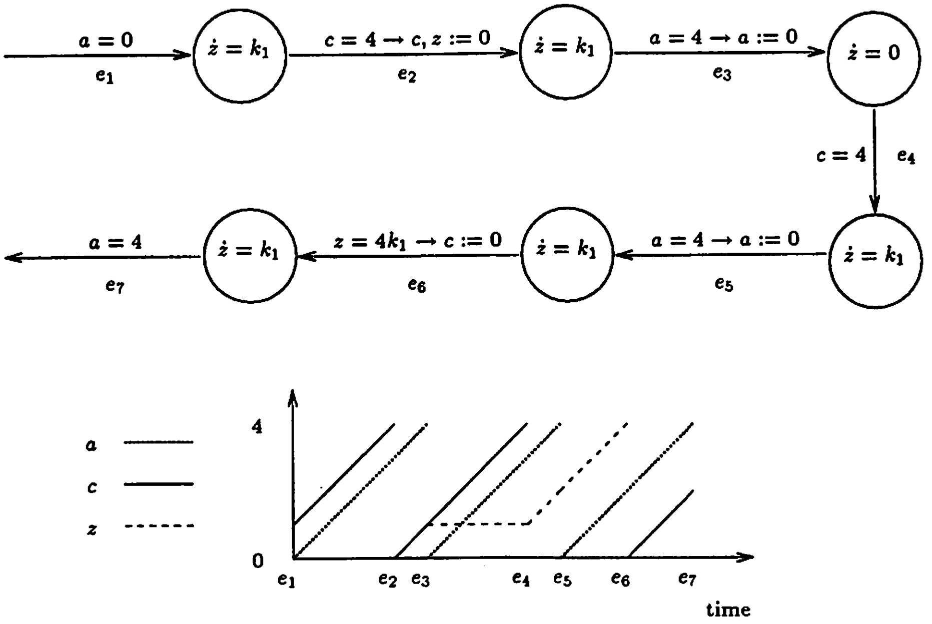

Figure 20: Doubling c using the two-slope variable z with the slopes 0 and k_1

Rectangular automata with assignments

The jump function of A can be generalized to allow, during edge steps, the value of one variable to be assigned to another variable. An *jump function with assignments* for A assigns to each edge e of A both a jump set $jump(e) \subset \{1, \ldots, n\}$ and an assignment function $assign(e)\colon \{1, \ldots, n\} \to \{1, \ldots, n\}$. The edge-step relation $\xrightarrow{\sigma}$, for $\sigma \in \Sigma$, is then redefined as follows: $(v, \mathbf{x}) \xrightarrow{\sigma} (w, \mathbf{y})$ iff there is an edge $e = (v, w)$ of A such that $obs(e) = \sigma$, and $\mathbf{x} \in pre(v)$, and $\mathbf{y} \in post(w)$, and for all $i \in \{1, \ldots, n\}$ with $i \notin jump(e)$, we have $y_i = x_{assign(i)}$. A *rectangular automaton with assignments* is a rectangular automaton whose jump function is replaced by a jump function with assignments.

Using a jump function with assignments, the proof of Theorem 4.1 can be replicated even if the two-slope variable is replaced by a memory cell (with slope 0) or by a skewed clock (with any slope different from 0 and 1). The former gives a new proof of a result from [Cer92]. In the following theorem, notice that every simple rectangular automaton whose nonclock variable is a one-slope variable is necessarily initialized.

Theorem 4.2 *For every slope $k \in \mathbb{Q} \setminus \{1\}$, the reachability problem is undecidable for simple rectangular automata with assignments and a one-slope variable of slope k.*

Proof. Consider $k \in \mathbb{Q} \setminus \{1\}$. We repeat the construction given in the proof of Theorem 4.1 for the case $k_1 = 1$ and $k_2 = k$ with the following modifications. The two-slope variable z is replaced by a clock z_1 and a one-slope variable z_2 with slope k. Each vertex of A_M is augmented with a bit $slp \in \{1, 2\}$ that indicates if the value of z corresponds to the current value of z_1 or to the current value of z_2. Assigments are used to copy the value of z_1 into z_2, or vice versa, whenever the bit slp changes. More precisely, for each edge $e = (v, w)$ of A_M, we have an edge e' from (v, slp) to (w, slp'), where $slp = i$ iff the slope of z in v is k_i, and $slp' = i$ iff the slope of z in w is k_i. For all coordinates other than z, z_1, and z_2 the preguard and postguard intervals and the jump sets of e and e' coincide. In addition, $pre(e')(z_{slp}) = pre(e)(z)$, $post(e')(z_{slp'}) = post(e)(z)$, and $z_{slp'}$ is in

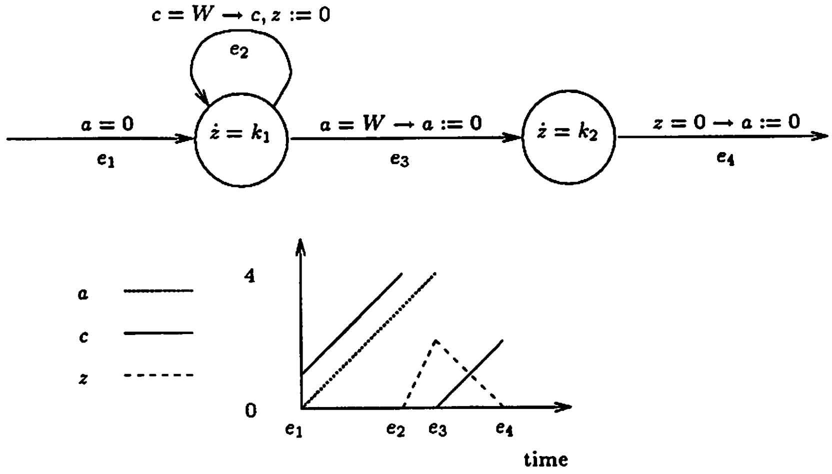

Figure 21: Multiplying c by $\frac{k_1}{|k_2|}$ when $k_1 > 0 > k_2$

the jump set of e' iff z is in the jump set of e. Finally, the assignment function of e' assigns to $z_{slp'}$ the value of z_{slp}. ■

Triangular preguard, postguard, and invariant constraints

The preguard, postguard, and invariant functions of A can be generalized to allow comparisons between the values of variables. We call preguard, postguard, and invariant constraints of the form $a \leq b$, where a and b are variables, triangular, because they define triangular regions of $\mathbb{R}^n$. A *triangular restriction* $\leq$ for A is a reflexive and transitive binary relation on $\{1, \ldots, n\}$. A *triangular preguard* (resp. *postguard*) *function* for A assigns to each edge e of A both a rectangular region *pre*(e) (resp. *post*(e)) and a triangular restriction $\leq_e$. The edge-step relation $\xrightarrow{\sigma}$, for $\sigma \in \Sigma$, is then redefined as follows: $(v, \mathbf{x}) \xrightarrow{\sigma} (w, \mathbf{y})$ iff there is an edge $e = (v, w)$ of A such that *obs*$(e) = \sigma$, $\mathbf{x} \in$ *pre*(v), $\mathbf{y} \in$ *post*(w), for all $i \in \{1, \ldots, n\}$ with $i \notin$ *jump*(e), we have $x_i = y_i$, and for all $i, j \in \{1, \ldots, n\}$ with $i \leq_e j$, we have $x_i \leq x_j$ (resp. $y_i \leq y_j$). A *triangular invariant function* for A assigns to each vertex v of A both a rectangular region *inv*(v) and a triangular restriction $\leq_v$. The state space Q of A is then redefined to contain a pair $(v, \mathbf{x}) \in V \times \mathbb{R}^n$ iff $\mathbf{x} \in$ *inv*(v) and for all $i, j \in \{1, \ldots, n\}$ with $i \leq_v j$, we have $x_i \leq x_j$. An *automaton with triangular preguards* (resp. *postguards*; *invariants*) is a rectangular automaton whose preguard (resp. postguard; invariant) function is replaced by a triangular preguard (resp. postguard; invariant) function.

Using any of the three types of triangular constraints, the proof of Theorem 4.2 can be replicated without assignments.

Theorem 4.3 *For every slope $k \in \mathbb{Q} \setminus \{1\}$, the reachability problem is undecidable for simple automata with triangular preguards or postguards or invariants, and a one-slope variable of slope k.*

Proof. Triangular preguards, postguards, or invariants permit comparisons between variables of the form $a = b$. This allows an assignment $a := b$ to be simulated by a reset $a := 0$ at a nondeterministically chosen point in time, followed later by the test $a = b$ (assuming positive values and slopes). As usual, wrapping ensures that all other variables, as well as b, maintain their values while the assignment is simulated. In this way, the construction outlined in the proof of Theorem 4.2 can be modified appropriately. ■

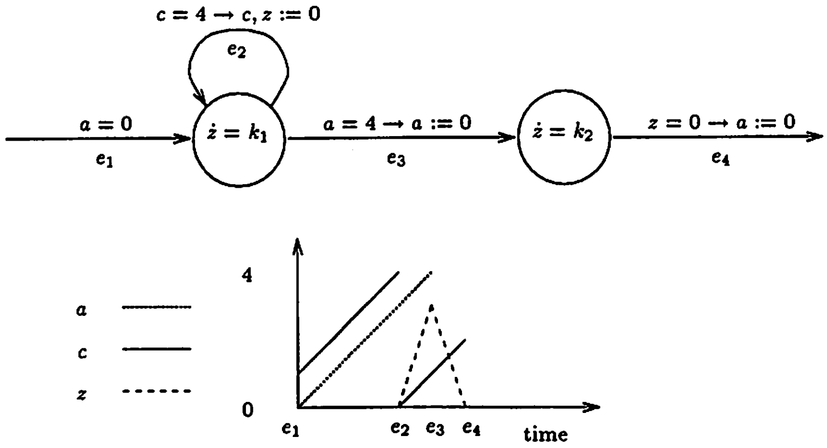

Figure 22: Doubling c when $k_2 = -k_1$

Triangular flow constraints

The flow functions of A can be generalized to impose an ordering on the first derivatives of variables. For example, the triangular flow constraint $1 \le \dot{a} \le \dot{b} \le 2$ says that both a and b may increase at any slopes between 1 and 2, but b increases always as least as fast as a. A *triangular flow function* for A assigns to each vertex v of A both a rectangular region *flow*(v) and a triangular restriction $\le_v$. For $t > 0$, the time-step relation $\xrightarrow{t}$ is then redefined as follows: $(v, \mathbf{x}) \xrightarrow{t} (w, \mathbf{y})$ iff $v = w$, and $(\mathbf{y} - \mathbf{x})/t \in$ *flow*(v), and for all $i, j \in \{1, \ldots, n\}$ with $i \le_v j$, we have $y_i - x_i \le y_j - x_j$. The triangular flow function is called *constant* if the functions *flow* and $\lambda v.\ \le_v$ are both constant functions on the set of vertices. An *automaton with constant triangular flow* is a rectangular automaton whose flow function is replaced by a constant triangular flow function.

Using a constant triangular flow constraint, the proof of Theorem 4.1 can be replicated. This is nontrivial, because Theorem 4.1 permits a two-slope variable that is not governed by a constant flow function. For the simulation of the two-slope variable, we use three infinite-slope variables that do follow a constant flow function, albeit a triangular one.

Theorem 4.4 *The reachability problem is undecidable for 3-simple automata with constant triangular flow.*

Proof. We use three nonclock variables z, z_1, and z_2 with the constant triangular flow constraint $1 \le \dot{z}_1 \le \dot{z} \le \dot{z}_2 \le 2$. The idea is to repeat the construction given in the proof of Theorem 4.1 for the case $k_1 = 1$ and $k_2 = 2$ with two additional variables, z_1 and z_2, which enforce that the slope of z is always either 1 or 2. The slope 1 of z is enforced by resetting z_2 to 0 whenever a wraps to 0, and later checking that $a = 4 \wedge z_2 = 4$. Similarly, the slope 2 of z is enforced by resetting z_1 to 0 whenever a wraps to 0, and later checking that $a = 4 \wedge z_1 = 8$. ■

5 Conclusion

There are three uniform extensions of finite-state machines with real-valued variables. *Timed automata* [AD94] equip finite-state machines with perfect clocks, and the reachability and ω-language emptiness problems for timed automata are decidable. *Linear hybrid automata* [AHH96] equip finite-state machines with continuous variables whose behavior satisfies linear constraints, and the

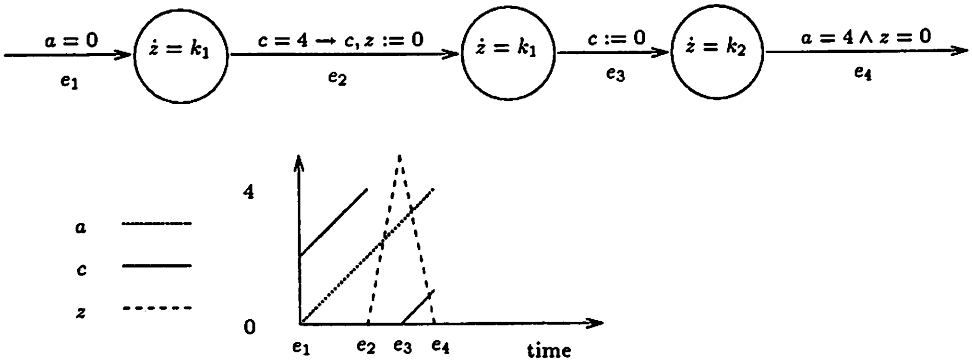

Figure 23: Halving c when $k_2 = -k_1$

reachability problem for linear hybrid automata is undecidable. Yet because the *Pre* and *Post* operations of linear hybrid automata maintain the linearity of zones, the reachability problem is semidecidable, and thus the verification of many linear hybrid systems is possible. This observation has been exploited in the model checker HyTech [HHWT97]. *Initialized rectangular automata* equip finite-state machines with drifting clocks, that is, continuous variables whose behavior satisfies rectangular constraints. Initialized rectangular automata lie strictly between timed automata and linear hybrid automata, at the boundary of decidability. On one hand, initialized rectangular automata generalize timed automata without incurring a complexity penalty. Their reachability problem is PSPACE-complete, and under the natural restriction of bounded nondeterminism, so is their ω-language emptiness problem. (We do not know the complexity of the ω-language emptiness problem without the restriction of bounded nondeterminism.) On the other hand, initialized rectangular automata form a maximal decidable class of hybrid systems, because even the simplest uninitialized or nonrectangular systems have undecidable reachability problems.

In summary, there are two factors for decidability: (1) *rectangularity*, that is, the behavior of all variables is decoupled, and (2) *initialization*, that is, a variable is reinitialized whenever its flow changes.

Initialized rectangular automata are also interesting from a practical perspective. First, reachability analysis using HyTech terminates on every initialized rectangular automaton with bounded invariants, and on every initialized rectangular automaton after a linear preprocessing step. Second, many distributed communication protocols assume that local clocks have bounded drift. Such protocols are naturally modeled as initialized rectangular hybrid automata. For example, HyTech has been applied successfully to verify one such protocol used in Philips audio components [HW95]. Third, initialized rectangular automata can be used to conservatively approximate, arbitrarily closely, hybrid systems with general dynamical laws [OSY94, HH95, PBV96].

Acknowledgement. We thank Howard Wong-Toi for a careful reading and for A_{bd}.

References

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[ACH97] R. Alur, C. Courcoubetis, and T.A. Henzinger. Computing accumulated delays in real-time systems. *Formal Methods in System Design*, 11(2):137–156, 1997.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems I*, Lecture Notes in Computer Science 736, pages 209–229. Springer-Verlag, 1993.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AHH96] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[AHV93] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, pages 592–601. ACM Press, 1993.

[BER94] A. Bouajjani, R. Echahed, and R. Robbana. Verifying invariance properties of timed systems with duration variables. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *FTRTFT 94: Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 863, pages 193–210. Springer-Verlag, 1994.

[BES93] A. Bouajjani, R. Echahed, and J. Sifakis. On model checking for real-time properties with durations. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science*, pages 147–159. IEEE Computer Society Press, 1993.

[BR95] A. Bouajjani and R. Robbana. Verifying ω-regular properties for subclasses of linear hybrid systems. In P. Wolper, editor, *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science 939, pages 437–450. Springer-Verlag, 1995.

[Cer92] K. Cerāns. *Algorithmic Problems in Analysis of Real-time System Specifications.* PhD thesis, University of Latvia, 1992.

[Cor96] J.C. Corbett. Timing analysis of ADA tasking programs. *IEEE Transactions on Software Engineering*, 22(7):461–483, 1996.

[HH95] T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In P. Wolper, editor, *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science 939, pages 225–238. Springer-Verlag, 1995.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: the next generation. In *Proceedings of the 16th Annual Real-time Systems Symposium*, pages 56–65. IEEE Computer Society Press, 1995.

[HHWT97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: a model checker for hybrid systems. In O. Grumberg, editor, *CAV 97: Computer-aided Verification*, Lecture Notes in Computer Science 1254, pages 460–463. Springer-Verlag, 1997.

[HMP92] T.A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In W. Kuich, editor, *ICALP 92: Automata, Languages, and Programming*, Lecture Notes in Computer Science 623, pages 545–558. Springer-Verlag, 1992.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[HRP94] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximation. In B. LeCharlier, editor, *SAS 94: Static Analysis Symposium*, Lecture Notes in Computer Science 864, pages 223–237. Springer-Verlag, 1994.

[HW95] P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In P. Wolper, editor, *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science 939, pages 381–394. Springer-Verlag, 1995.

[HWT96] T.A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, Lecture Notes in Computer Science 1165, pages 265–282. Springer-Verlag, 1996.

[Kop96] P.W. Kopke. *The Theory of Rectangular Hybrid Automata*. PhD thesis, Cornell University, 1996.

[KPSY93] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: a class of decidable hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 179–208. Springer-Verlag, 1993.

[LV96] N.A. Lynch and F. Vaandrager. Forward and backward simulations, part ii: Timing-based systems. *Information and Computation*, 128(1):1–25, 1996.

[MV94] J. McManis and P. Varaiya. Suspension automata: a decidable class of hybrid automata. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 105–117. Springer-Verlag, 1994.

[NOSY93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems I*, Lecture Notes in Computer Science 736, pages 149–178. Springer-Verlag, 1993.

[NS95] S. Nadjm-Tehrani and J.-E. Strömberg. Proving dynamic properties in an aerospace application. In *Proceedings of the 16th Annual Real-time Systems Symposium*, pages 2–10. IEEE Computer Society Press, 1995.

[OSY94] A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 81–94. Springer-Verlag, 1994.

[PBV96] A. Puri, V. Borkar, and P. Varaiya. ε-approximation of differential inclusions. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 362–376. Springer-Verlag, 1996.

[PV94] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 95–104. Springer-Verlag, 1994.

[SMF97] T. Stauner, O. Müller, and M. Fuchs. Using HyTech to verify an automotive control system. In O. Maler, editor, *HART 97: Hybrid and Real-time Systems*, Lecture Notes in Computer Science 1201, pages 139–153. Springer-Verlag, 1997.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Annual Symposium on Logic in Computer Science*, pages 322–331. IEEE Computer Society Press, 1986.