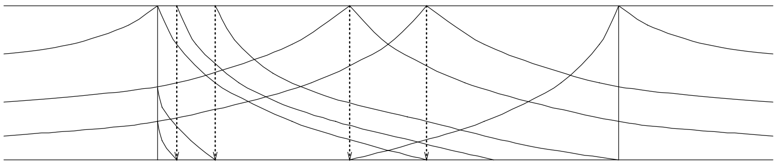
Hybrid Automata with Finite Bisimulations*

Thomas A. Henzinger[†]
Department of Computer Science, Cornell University, Ithaca, NY 14850

Abstract. The analysis, verification, and control of hybrid automata with finite bisimulations can be reduced to finite-state problems. We advocate a time-abstract, phase-based methodology for checking if a given hybrid automaton has a finite bisimulation. First, we factor the automaton into two components, a boolean automaton with a discrete dynamics on the finite state space $\mathbb{B}^m$ and a euclidean automaton with a continuous dynamics on the infinite state space $\mathbb{R}^n$. Second, we investigate the phase portrait of the euclidean component. In this fashion, we obtain new decidability results for hybrid systems as well as new, uniform proofs of known decidability results. For example, we prove that if two hybrid automata have finite bisimulations, and both can be calibrated to a common time scale, then their product also has a finite bisimulation.



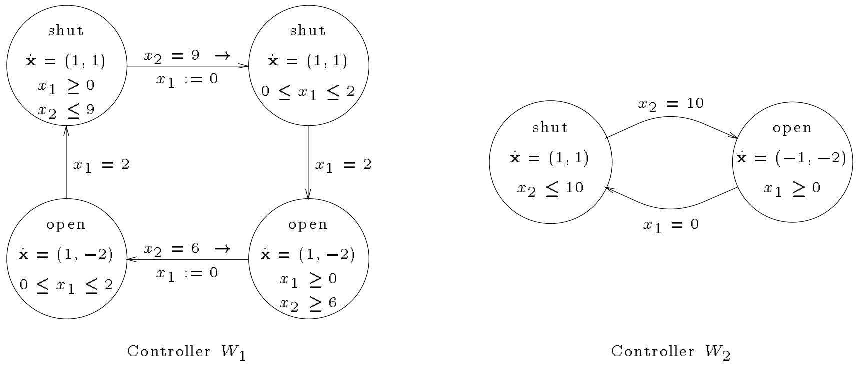
1 Introduction

A *hybrid automaton* [2] is a mathematical model for a digital program that interacts with an analog environment. Hybrid automata are useful for analyzing embedded systems [4, 10, 14, 21, 22]. We advocate the view of hybrid automata as infinite-state transition systems. We call an infinite-state transition system *finitary* and *effective* if it has a finite bisimulation that can be constructed effectively, and we examine the question of which hybrid automata are finitary and effective. This question is important for the analysis of embedded systems, because the problems of language emptiness, model checking, and controller synthesis are decidable for hybrid automata that are finitary and effective [1, 13, 19].

There have been several results of the type "the emptiness problem is undecidable for hybrid automata with clocks and a stopwatch" [2, 12, 15]. While these results leave little hope for finding

[*] A preliminary version of this paper appeared in the *Proceedings of the 22nd International Colloquium on Automata, Languages, and Programming* (ICALP), *Lecture Notes in Computer Science* **944**, Springer-Verlag, 1995, pp. 324–335.

[†] Supported in part by the National Science Foundation under grant CCR-9200794, by the United States Air Force Office of Scientific Research under contract F49620-93-1-0056, and by the Defense Advanced Research Projects Agency under grant NAG2-892.



Controller W_1 Controller W_2

Figure 1: Two water-level controllers

a general class of finitary hybrid automata (those with finite bisimulations), they must be brought into accord with our experience with the symbolic model checker HyTech for hybrid automata [4]. Although HyTech is guaranteed to terminate only for finitary hybrid automata, the procedure does terminate on many automata with clocks and a stopwatch, including the well-known example of a leaking gas burner [9]. It is this apparent discrepancy between verification theory and practice we set out to explain in this paper.

Consider, for example, the two water-level controllers shown in Figure 1. The real-valued variable x_2 represents the water level in a tank. The water level x_2 increases at the rate of 1 m s^{-1} if the valve at the bottom of the tank is shut, and x_2 decreases at the rate of 2 m s^{-1} if the valve is open. The controller W_1 uses a clock x_1 to open the valve 2 s after the water level hits 9 m, and it closes the valve 2 s after the water level falls to 6 m. The controller W_2 opens the valve when the water level hits 10 m, and keeps the valve open for precisely the amount of time that it was closed. Neither controller falls into a known decidable subclass of hybrid automata. If we run HyTech to check if the water level stays within a given range, the verification succeeds only for controller W_1. We explain this phenomenon, and we provide a technique that allows us to see, a priori, that controller W_1 is finitary, while controller W_2 is not.

Hybrid automata are time-invariant (autonomous) in the sense that a transition depends only on the current state of the automaton, and not on the current time. All timing aspects are explicitly encoded in the state space, say, through the values of clock variables. Time-invariance allows us to take a phase view of hybrid automata, which abstracts time and gives geometric insight into the state-transition structure of hybrid automata. The phase domain was first exploited by Alur and Dill for proving that the so-called region quotient is a finite bisimulation of a timed automaton [3]. This paper can be seen as an extension of and a tribute to their work.

Consider, for example, a finitary hybrid automaton H with four control locations, two clocks, and one stopwatch. There are two extreme approaches for checking that H is finitary. The *specific* approach looks at the phase portrait of H in the 5D space $\mathbb{B}^2 \times \mathbb{R}^3$. The *generic* approach studies the possible phase portraits of hybrid automata with clocks and a stopwatch. While the specific approach may be unnecessarily concrete and expensive, the general approach is too abstract to yield the desired result (there are hybrid automata with clocks and a stopwatch that are not finitary). Inspired by the theory of dynamical systems, we advocate a natural intermediate approach. We factor the hybrid automaton $H = (B, E)$ into a boolean component B, with a discrete dynamics on the state space $\mathbb{B}^2$, and a euclidean component E, with a continuous dynamics on the state space $\mathbb{R}^3$. A sufficient condition for the finitariness of H, then, is the finitariness of E. We check that the euclidean automaton E is finitary by looking at the phase portrait of E, which does contain clocks and a stopwatch, but also takes into account the concrete set of constraints that H puts on these variables.

This approach leads, first, to a finer distinction between the decidability and undecidability of hybrid automata than previous results indicate and, second, to a uniform explanation of many previous decidability results [2, 6, 20, 21]. While their decidability proofs relied either on transformations into timed automata or on digitizations of trajectories, our finitariness proofs provide insight into the state-transition structure of hybrid automata and guarantee the termination of HyTech. Second, we prove new classes of hybrid automata to be finitary. Among them are a class of hybrid automata with periodic dynamics (Theorem 9), a class of hybrid automata that includes the water-level controller W_1 shown above (Theorem 11), and several product classes of finitary hybrid automata (Theorem 13).

While also geometric in its intuition, the hybrid-automaton model differs from the related approach of [18]. First, their dynamical systems are deterministic and our euclidean automata are nondeterministic, both as far as discrete and continuous progress is concerned. Second, we consider the product of euclidean automata with boolean automata, which results in multiple copies of euclidean state spaces. In particular, unlike deterministic dynamical systems, hybrid automata are undecidable already in the 2D (planar) case.

We begin with developing a terminology for infinite-state transition systems (Section 2), then proceed to study euclidean automata (Section 3), and finally combine boolean and euclidean automata to obtain hybrid automata (Section 4).

2 Labeled Transition Systems

A (*labeled*) *transition system* $A = (Q, \Sigma, \rightarrow, \overleftarrow{Q}, \overrightarrow{Q})$ consists of a set Q of states, a finite alphabet Σ, a labeled transition relation $\rightarrow \subseteq Q \times \Sigma \times Q$, a set $\overleftarrow{Q} \subseteq Q$ of initial states, and a set $\overrightarrow{Q} \subseteq Q$ of final states. We write $q \xrightarrow{\sigma} q'$ for $(q, \sigma, q') \in \rightarrow$. An *$A$-trace* $(\underline{q}, \underline{\sigma}) = q_0\sigma_0 q_1 \cdots \sigma_{n-1} q_n$ consists of a finite sequence $\underline{q} \in Q^*$ of states and a word $\underline{\sigma} \in \Sigma^*$ such that (1) for all $0 \leq i < n$, $q_i \xrightarrow{\sigma_i} q_{i+1}$, and (2) $q_0 \in \overleftarrow{Q}$ and $q_n \in \overrightarrow{Q}$. If $(\underline{q}, \underline{\sigma})$ is an A-trace, then $\underline{\sigma}$ is an *A-word*. We write $\llbracket A \rrbracket$ for the set of A-traces, and $\llbracket A \rrbracket_L \subseteq \Sigma^*$ for the set of A-words —the *language* of the transition system A. The transition system A is a *finite automaton* if the state space Q is finite. The languages of the finite automata are the regular languages.

Effective transition systems. A *region* is a set $R \subseteq Q$ of states. For a letter $\sigma \in \Sigma$, the *weakest σ-precondition* $pre_\sigma(R)$ of the region R is the set of states from which a state in R can be reached by a σ-transition: $q \in pre_\sigma(R)$ iff $(\exists q' \in R \mid q \xrightarrow{\sigma} q')$. We write $dom(pre_\sigma)$ for the region $\{q \in Q \mid pre_\sigma(\{q\}) \neq \emptyset\}$, and $pre(R)$ for the region $\bigcup_{\sigma \in \Sigma} pre_\sigma(R)$. The transition system A is *effective* if there is a class $\mathcal{R}$ of effectively representable regions such that (1) $\overleftarrow{Q}, \overrightarrow{Q} \in \mathcal{R}$, (2) $\mathcal{R}$ is effectively closed under all boolean operations and all pre_σ-operations, for $\sigma \in \Sigma$, and (3) the emptiness problem is decidable for the regions in $\mathcal{R}$. For example, every Turing machine is effective (let $\mathcal{R}$ be the class of all regions, where each region is represented as a set of states).

Finitary transition systems. Let $\sim \subseteq Q^2$ be an equivalence relation of states, and let $Q/_\sim$ be the corresponding partition of the state space Q. A *$\sim$-block* is a union of $\sim$-equivalence classes. For a region $R \subseteq Q$, let $R/_\sim$ be the smallest $\sim$-block that contains R. For two regions $R, R' \subseteq Q$, let $R \xrightarrow{\sigma}_\exists R'$ if there are two states $q \in R$ and $q' \in R'$ with $q \xrightarrow{\sigma} q'$. The *quotient system* $A/_\sim$ is the transition system with the state space $Q/_\sim$, the alphabet Σ, the transition relation $\rightarrow_\exists$, the initial region $\overleftarrow{Q}/_\sim$, and the final region $\overrightarrow{Q}/_\sim$.

The equivalence $\sim$ is *finite* if it has a finite number of equivalence classes. The equivalence $\sim$ is a *bisimulation* for the transition system A if $\overrightarrow{Q}$ is a $\sim$-block and for all letters $\sigma \in \Sigma$ and all $\sim$-equivalence classes R, the region $pre_\sigma(R)$ is a $\sim$-block. The two states $p, q \in Q$ are *bisimilar*, written $p \approx q$, if there is a bisimulation $\sim$ such that $p \sim q$. If p and q are bisimilar, then (1) $p \in \overrightarrow{Q}$ iff $q \in \overrightarrow{Q}$; (2) if $p \xrightarrow{\sigma} p'$, then there is a state $q' \in Q$ such that $q \xrightarrow{\sigma} q'$ and $p' \approx q'$; and (3) if $q \xrightarrow{\sigma} q'$, then there is a state $p' \in Q$ such that $p \xrightarrow{\sigma} p'$ and $p' \approx q'$.

Proposition 1. *For every transition system A, if $\sim$ is a bisimulation for A, then $[\![A]\!]_L = [\![A/_\sim]\!]_L$.*

The transition system A is *finitary* if there is a finite bisimulation for A. In particular, every finite automaton is finitary. Examples of finitary transition systems with infinite state spaces are affine transition systems [17] and finite automata with real-valued clocks [3]. By Proposition 1, the languages of the finitary transition systems are the regular languages.

The bisimilarity partition $Q/_\approx$ for the transition system A can be computed by successive approximation:

```
procedure BisimApprox:
    $Q/_\sim := \{\overrightarrow{Q}, Q - \overrightarrow{Q}\}$;
    while {assert $\approx \subseteq \sim$} there are two regions $R, R' \in Q/_\sim$ and a letter
        $\sigma \in \Sigma$ such that $\emptyset \subset R \cap pre_\sigma(R') \subset R$ do
        $Q/_\sim := (Q/_\sim - \{R\}) \cup \{R \cap pre_\sigma(R'), R - pre_\sigma(R')\}$
        od
    {assert $\approx = \sim$}.
```

Each step of the procedure BisimApprox is effectively computable if A is effective. The successive approximation converges —i.e., the procedure BisimApprox terminates— iff A is finitary. Implementations of the procedure BisimApprox are discussed in [16, 23] for finite automata and in [7, 17] for arbitrary transition systems.

Questions about transition systems.[1] The *emptiness problem* for transition systems asks, given a transition system A, is the language $[\![A]\!]_L$ empty. If A is effective and finitary, then the emptiness problem can be solved by first computing the bisimilarity quotient $A/_\approx$, and then checking the emptiness of the finite quotient system $A/_\approx$.

Theorem 2. *The emptiness problem is decidable for transition systems that are effective and finitary.*

Instead of computing the quotient system $A/_\approx$, it typically is more efficient to compute, by successive approximation, the region $pre^*(\overrightarrow{Q})$ of states from which a final state can be reached by any finite number of transitions:

```
procedure ReachApprox:
    $R := \overrightarrow{Q}$;
    while {assert $R \subseteq pre^*(\overrightarrow{Q})$ and $R$ is a $\approx$-block} $pre(R) \not\subseteq R$ do
        $R := R \cup pre(R)$;
        if $R \cap \overleftarrow{Q} \neq \emptyset$ then return "$[\![A]\!]_L \neq \emptyset$" fi
        od;
    {assert $R = pre^*(R)$} return "$[\![A]\!]_L = \emptyset$".
```

[1] In the full paper we discuss, in addition to the emptiness problem, also the model-checking problem and the control problem for transition systems.

Again, each step of the procedure ReachApprox is effectively computable if A is effective, and the procedure terminates if A is finitary. While in the worst case both ReachApprox and BisimApprox proceed identically, the successive approximation of $pre^*(\overrightarrow{Q})$ may converge in fewer iterations than the successive approximation of $Q/_\approx$; indeed, the successive approximation of $pre^*(\overrightarrow{Q})$ may converge even if the successive approximation of $Q/_\approx$ does not.

In the full paper, we also discuss the model-checking problem and the control problem for labeled transition systems. The *model-checking problem* asks, given a labeled transition system A and a μ-calculus formula χ, to compute the characteristic region $[\![\chi]\!]_A \subseteq Q$ of states that satisfy χ. The *control problem* asks, given A, a partition of Σ into input and output letters, and a $\approx$-block $R \subseteq Q$, to construct a finite automaton B such that the product system $A \times B$ is as input-enabled as A and its trajectories stay within the region R. If A is effective and finitary, then both problems can be reduced to the finite quotient sytem $A/_\approx$ (for example, if all atoms of the μ-calculus formula χ define $\approx$-blocks, then $[\![\chi]\!]_A = [\![\chi]\!]_{A/\approx}$). Also, in both cases, there are successive-approximation procedures that depend on χ and R, respectively, and by computing problem-dependent quotients of A typically converge faster than the successive approximation of the bisimilarity quotient $A/_\approx$ [13, 19].

Operations on transition systems. Let $A_1 = (Q_1, \Sigma_1, \rightarrow_1, \overleftarrow{Q}_1, \overrightarrow{Q}_1)$ and $A_2 = (Q_2, \Sigma_2, \rightarrow_2, \overleftarrow{Q}_2, \overrightarrow{Q}_2)$ be two transition systems. An *alphabet combinator* γ is a partial function on $\Sigma_1 \times \Sigma_2$. We write $\gamma(\Sigma_1, \Sigma_2)$ for the range of γ. Let $(q_1, q_2) \overset{\sigma}{\Rightarrow} (q_1', q_2')$ if $\gamma(\sigma_1, \sigma_2) = \sigma$, $q_1 \xrightarrow{\sigma_1} q_1'$, and $q_2 \xrightarrow{\sigma_2} q_2'$. The *product* $A_1 \times_\gamma A_2$ is the transition system with the state space $Q_1 \times Q_2$, the alphabet $\gamma(\Sigma_1, \Sigma_2)$, the transition relation $\Rightarrow$, the initial region $\overleftarrow{Q_1} \times \overleftarrow{Q_2}$, and the final region $\overrightarrow{Q_1} \times \overrightarrow{Q_2}$. It follows that the product of two effective transition systems is again effective.

Proposition 3. *For all transition systems A_1 and A_2, if $\sim_1$ is a bisimulation for A_1, and $\sim_2$ is a bisimulation for A_2, then $\sim_1 \times \sim_2$ is a bisimulation for $A_1 \times_\gamma A_2$ (where $(p_1, p_2) \sim_1 \times \sim_2 (q_1, q_2)$ iff $p_1 \sim_1 q_1$ and $p_2 \sim_2 q_2$).*

Corollary 4. *The product of two finitary transition systems is finitary.*

The finitariness of transition systems, however, is not preserved by the reversal of the transition relation. Let $A = (Q, \Sigma, \rightarrow, \overleftarrow{Q}, \overrightarrow{Q})$ be a transition system. The *reverse system* A^{-1} is the transition system with the state space Q, the alphabet Σ, the transition relation $\leftarrow$ (where $q \overset{\sigma}{\leftarrow} q'$ iff $q' \xrightarrow{\sigma} q$), the initial region $\overrightarrow{Q}$, and the final region $\overleftarrow{Q}$. For a region $R \subseteq Q$, the weakest σ-precondition $pre_\sigma(R)$ of R with respect to A is the strongest σ-postcondition of R with respect to the reverse system A^{-1}. For a language $L \subseteq \Sigma^*$, the *reverse language* $L^{-1} \subseteq \Sigma^*$ contains all words from L read backwards.

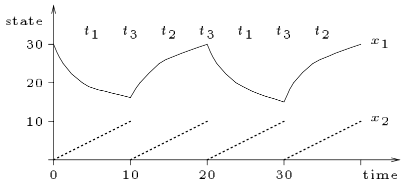
Proposition 5. *For every transition system A, $[\![A^{-1}]\!]_L = [\![A]\!]_L^{-1}$.*

It follows that the language of A is empty iff the language of A^{-1} is empty.

Corollary 6. *The emptiness problem is decidable for transition systems whose reversal is effective and finitary.*

The reversal operation is interesting, as A^{-1} may be effective or finitary if A is not.[2] Consider, for example, the transition system $\hat{A}$ with the state space $\mathbb{N}$, a unary alphabet, the transition relation $(n+1) \rightarrow n$, for all $n \in \mathbb{N}$, a finite initial region $\overleftarrow{Q} \subset \mathbb{N}$, and a nonempty finite final region $\overrightarrow{Q} \subset \mathbb{N}$. Then $\hat{A}$ is not finitary, and $\hat{A}^{-1}$ is finitary.

[2] Looking at the reversal of a transition system is equivalent to considering postconditions instead of preconditions.



t_1: $0 \leq x_2 \leq 10 \wedge 10\dot{x}_1 = -x_1 \wedge \dot{x}_2 = 1$
t_2: $0 \leq x_2 \leq 10 \wedge 10\dot{x}_1 = -x_1 + 50 \wedge \dot{x}_2 = 1$
t_3: $x_2 = 10 \wedge x_1' = x_1 \wedge x_2' = 0$

Figure 2: A trace of the temperature controller T

3 Euclidean Automata

Linear-time semantics: curves, jumps, and trajectories. The *dimension* of a euclidean automaton is a nonnegative integer n. The *state space* of a euclidean automaton of dimension n is the euclidean space $\mathbb{R}^n$. A *state* is a point $\mathbf{x} \in \mathbb{R}^n$ of the state space. A *region*, then, is a set $R \subseteq \mathbb{R}^n$ of points.

A *curve* (δ, f) consists of a positive real $\delta \in \mathbb{R}_{>0}$ —the *duration* of the curve— and a differentiable function $f\colon [0, \delta] \to \mathbb{R}^n$. The states $f(0)$ and $f(\delta)$ are called the *source* and the *target* of the curve. Given $t \in [0, \delta]$, we write $f'(t)$ for the first derivative $df(t)/dt$ of f at t. A (nondeterministic) *activity* is a set of curves. The activity F is *additive* if $f(\delta) = g(0)$ and $f'(\delta) = g'(0)$ implies that $(\delta+\varepsilon, f \uplus g) \in F$ iff both $(\delta, f) \in F$ and $(\varepsilon, g) \in F$. The activity F can be *slowed down* if $(\delta, f) \in F$ implies that $(\varepsilon, g) \in F$ for all positive durations $\varepsilon < \delta$ and all functions g with $g(t) = f(t \cdot \delta/\varepsilon)$. The activity F can be *sped up* if $(\delta, f) \in F$ implies that $(\varepsilon, g) \in F$ for all durations $\varepsilon > \delta$ and all functions g with $g(t) = f(t \cdot \delta/\varepsilon)$. The activity F is *time-abstract* if F can be sped up and slowed down.

A *jump* is a pair $(\mathbf{x}, \mathbf{x}') \in \mathbb{R}^n \times \mathbb{R}^n$ of states —the *source* $\mathbf{x}$ and the *target* $\mathbf{x}'$. The *duration* of a jump is 0. A (nondeterministic) *action* is a set of jumps.

A (*trajectory*) *segment* is a curve or a jump. Given a segment τ, we write $\overleftarrow{\tau} \in \mathbb{R}^n$ for the source, $\overrightarrow{\tau} \in \mathbb{R}^n$ for the target, and $|\tau| \in \mathbb{R}_{\geq 0}$ for the duration of τ. The two segments τ_1 and τ_2 are *adjacent* if $\overrightarrow{\tau_1} = \overleftarrow{\tau_2}$. A *trajectory* $\underline{\tau} = \tau_0 \cdots \tau_n$ is a finite sequence of segments such that for all $0 \leq i < n$, the segments τ_i and τ_{i+1} are adjacent. The trajectory $\underline{\tau}$ has the *source* $\overleftarrow{\underline{\tau}} = \overleftarrow{\tau_0}$, the *target* $\overrightarrow{\underline{\tau}} = \overrightarrow{\tau_n}$, and the *duration* $|\underline{\tau}| = \Sigma_{0 \leq i \leq n} |\tau_i|$.

Syntax: region, activity, and action formulas. Let $X = \{x_1, \ldots, x_n\}$, $dX = \{\dot{x}_1, \ldots, \dot{x}_n\}$, and $X' = \{x_1', \ldots, x_n'\}$ be three disjoint ordered sets of real-valued variables. A *region formula* is a formula whose free variables are in X. Every region formula defines a region: the state $\mathbf{x} \in \mathbb{R}^n$ *satisfies* the region formula $\rho(X)$, written $\mathbf{x} \models \rho$, if $\rho(\mathbf{x})$ holds. An *activity formula* is a formula whose free variables are in $X \cup dX$. Every activity formula defines an additive activity: the curve (δ, f) *satisfies* the activity formula $\varphi(X, dX)$ if $\varphi(f(t), f'(t))$ holds for all $t \in [0, \delta]$. We write $\mathbf{x} \xrightarrow{\varphi} \mathbf{x}'$ if there is a curve (δ, f) with source $\mathbf{x}$ and target $\mathbf{x}'$ that satisfies φ. An *action formula* is a formula whose free variables are in $X \cup X'$. Every action formula defines an action: the jump $(\mathbf{x}, \mathbf{x}')$ *satisfies* the action formula $\psi(X, X')$, written $\mathbf{x} \xrightarrow{\psi} \mathbf{x}'$, if $\psi(\mathbf{x}, \mathbf{x}')$ holds. By $[\![\rho]\!]$ ($[\![\varphi]\!]$; $[\![\psi]\!]$) we denote the set of states (curves; jumps) that satisfy the corresponding formula.

The *alphabet* $\Sigma = \Phi \uplus \Psi$ of a euclidean automaton is the disjoint union of a finite set Φ of activity formulas —the *activity alphabet*— and a finite set Ψ of action formulas —the *action alphabet*. A Σ-*trace* $(\underline{\sigma}, \underline{\tau}) = (\sigma_0, \tau_0) \cdots (\sigma_n, \tau_n)$ consists of a word $\underline{\sigma} \in \Sigma^*$ and a trajectory $\underline{\tau}$ such that for all $0 \leq i \leq n$, the segment τ_i satisfies the formula σ_i.

Example 1. Consider the 2D euclidean component T of a temperature-control system. The variable

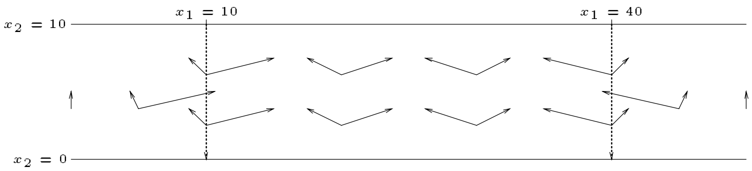


Figure 3: Phase portrait of the temperature controller T

x_1 represents the temperature, and the variable x_2 represents a clock of the controller. If the heater is turned off, then x_1 decreases according to the linear differential equation $\dot{x}_1 = -1/10x_1$, and if the heater is turned on, then x_1 follows $\dot{x}_1 = -1/10x_1 + 5$. We assume that the controller resets the clock x_2 to 0 every 10 s. Thus the alphabet $\Sigma_T = \{t_1, t_2, t_3\}$ consists of the two activity formulas t_1 and t_2, and the action formula t_3 of Figure 2. The figure also shows a sample Σ_T-trace.[3] $\square$

A *euclidean automaton* $E = (n, \Sigma, \overleftarrow{\rho}, \overrightarrow{\rho})$ consists of a dimension n, an alphabet Σ, and two region formulas $\overleftarrow{\rho}$ and $\overrightarrow{\rho}$. The region formula $\overleftarrow{\rho}$ defines the *initial region* of E; the region formula $\overrightarrow{\rho}$ defines the *final region* of E. The Σ-trace $(\underline{\sigma}, \underline{\tau})$ is an *E-trace* if it leads from the initial to the final region —that is, $\overleftarrow{\underline{\tau}} \models \overleftarrow{\rho}$ and $\overrightarrow{\underline{\tau}} \models \overrightarrow{\rho}$. If $(\underline{\sigma}, \underline{\tau})$ is an E-trace, then $\underline{\sigma}$ is an *E-word* and $\underline{\tau}$ is an *E-trajectory.* We write $[\![E]\!]$ for the set of E-traces, and $[\![E]\!]_L \subseteq \Sigma^*$ for the set of E-words —the *language* of the euclidean automaton E. We write $[\![E]\!]$ for the set of E-traces, $[\![E]\!]_L$ for the set of E-words, and $[\![E]\!]_T$ for the set of E-trajectories.[4] The set $[\![E]\!]_L \subseteq \Sigma^*$ is called the *language* of the euclidean automaton E.

Phase semantics: flow and jump fields. We represent a vector with the origin $\mathbf{x} \in \mathbb{R}^n$ and the offset $\mathbf{y} \in \mathbb{R}^n$ by the pair $(\mathbf{x}, \mathbf{y})$ of states. We consider two types of vectors. The origin of a *flow vector* represents the source of a curve, and the offset represents the initial tangent of the curve. The origin of a *jump vector* represents the source of a jump, and the offset represents the relative effect of the jump. A (nondeterministic) vector field is a set of vectors. A field of flow vectors is called a *flow field*; a field of jump vectors, a *jump field.*

The phase portrait of an activity is a flow field. The flow vector $(\mathbf{x}, \mathbf{y})$ satisfies the activity formula $\varphi(X, dX)$ if $\varphi(\mathbf{x}, \mathbf{y})$ holds. The activity formula φ, then, defines the flow field $[\varphi]_F$ of all flow vectors that satisfy φ. The activity alphabet Φ defines the flow field $[\Phi]_F = \bigcup_{\varphi \in \Phi} [\varphi]_F$.

The phase portrait of an action is a jump field. The jump vector $(\mathbf{x}, \mathbf{y})$ satisfies the action formula $\psi(X, X')$ if $\psi(\mathbf{x}, \mathbf{x} + \mathbf{y})$ holds. The action formula ψ, then, defines the jump field $[\psi]_J$ of all jump vectors that satisfy ψ. The action alphabet Ψ defines the jump field $[\Psi]_J = \bigcup_{\psi \in \Psi} [\psi]_J$.

The phase portrait $[\Sigma]$ of the alphabet $\Sigma = \Phi \uplus \Psi$ consists of the flow field $[\Phi]_F$ and the jump field $[\Psi]_J$. The *phase portrait* $[E]$ of the euclidean automaton $E = (n, \Sigma, \overleftarrow{\rho}, \overrightarrow{\rho})$ consists of the flow field $[\Phi]_F$, the jump field $[\Psi]_J$, the initial region $[\![\overleftarrow{\rho}]\!]$, and the final region $[\![\overrightarrow{\rho}]\!]$.

Example 2. The phase portrait of the temperature controller T is shown in Figure 3. The solid vectors represent sample flow vectors, and the dotted vectors represent sample jump vectors. $\square$

Branching-time semantics: transition systems. With the euclidean automaton $E = (n, \Sigma, \overleftarrow{\rho}, \overrightarrow{\rho})$ we associate the transition relation $\xrightarrow{E} = \bigcup_{\sigma \in \Sigma} \xrightarrow{\sigma}$. The euclidean automaton E, then, defines the infinite-state transition system $\langle E \rangle$ with the state space $\mathbb{R}^n$, the alphabet Σ, the transition relation $\xrightarrow{E}$, the initial region $[\![\overleftarrow{\rho}]\!]$, and the final region $[\![\overrightarrow{\rho}]\!]$. We say that the euclidean automaton

[3] More details about examples and proofs are given in the full paper.

[4] Notice that the trajectory set $[\![E]\!]_T$ does not change if the alphabet $\Sigma = \Phi \uplus \Psi$ of E is replaced by the binary alphabet $\{\bigvee_{\varphi \in \Phi} \varphi\} \uplus \{\bigvee_{\psi \in \Psi} \psi\}$.

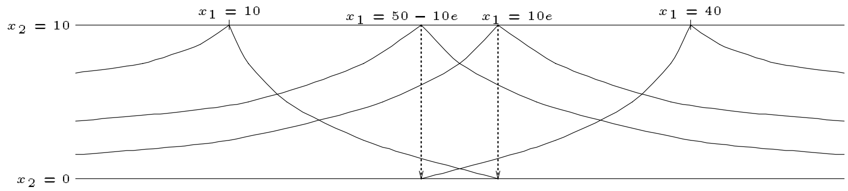


Figure 4: BisimApprox for the temperature controller T

E is effective (finitary) if the corresponding transition system $\langle E \rangle$ is effective (finitary). Since $[\![E]\!]_L = [\![\langle E \rangle]\!]_L$, the emptiness problem is decidable for euclidean automata that are both effective and finitary.

3.1 Effective Euclidean Automata

A *linear inequality* on the set V of variables compares a linear integer combination of variables from V with an integer constant. A *linear constraint* on V is a (finite) conjunction of linear inequalities on V. A *linear region formula* is a linear constraint on X. Every linear region formula ρ, then, defines a (convex) rational polyhedron $[\![\rho]\!] \subseteq \mathbb{R}^n$. A *linear activity formula* $\varphi = \varphi_P \wedge \varphi_S$ consists of a linear region formula φ_P —the *point invariant*— and a linear constraint φ_S on dX —the *slope invariant*. A *linear action formula* ψ is a linear constraint on $X \cup X'$. The euclidean automaton E is *linear* if all activity, action, and region formulas of E are linear. The linear euclidean automaton E is *bounded* if all point invariants of E define bounded regions (i.e., polytopes), and E is *positive* if all point invariants define subsets of $\mathbb{R}_{\geq 0}{}^n$. The *slope set* of the linear euclidean automaton E is the set of vectors $\mathbf{x} \in \mathbb{R}^n$ such that $\varphi_S(\mathbf{x})$ holds for some slope invariant $\varphi_S(dX)$ of E.

Theorem 7. *Every linear euclidean automaton is effective.*

Proof. The region $R \subseteq \mathbb{R}^n$ is *linear* if R is a finite union of rational polyhedra in $\mathbb{R}^n$. We choose $\mathcal{R}$ to be the class of linear regions. The linear regions can be represented effectively by formulas of the first-order theory $(\mathbb{R}, \leq, +)$ of the reals with order and addition, which is closed under all boolean operations, admits quantifier elimination, and has a decidable satisfiability problem. Indeed, every linear region can be defined by a (finite) disjunction of linear region formulas, and all *pre*-operations distribute over disjunction. So consider the linear region formula ρ and the linear activity formula $\varphi = \varphi_P(X) \wedge \varphi_S(\dot{X})$. From the mean-value theorem it follows that if a curve with source $\mathbf{x} \in \mathbb{R}^n$, target $\mathbf{y} \in \mathbb{R}^n$, and duration $\delta \in \mathbb{R}_{>0}$ satisfies φ, then the straight line from $\mathbf{x}$ to $\mathbf{y}$ of duration δ satisfies φ. The weakest precondition $pre_\varphi([\![\rho]\!])$, then, is defined by the formula

$$(\exists Y \mid \varphi_P(X) \,\wedge\, (\exists \delta > 0 \mid \varphi_S((Y - X)/\delta)) \,\wedge\, \varphi_P(Y) \,\wedge\, \rho(Y)).$$

Next, consider the linear action formula $\psi(X, X')$. The weakest precondition $pre_\psi([\![\rho]\!])$ is defined by the formula

$$(\exists X' \mid \psi(X, X') \,\wedge\, \rho(X')). \ \square$$

The procedures BisimApprox and ReachApprox are therefore semi-decision procedure for checking the emptiness of linear euclidean automata. Linearity, on the other hand, does not ensure the existence of a finite bisimulation. To see this, observe that every n-counter machine can be viewed as a linear euclidean automaton of dimension $n + 1$, where the $(n + 1)$-st dimension encodes the control of the machine.

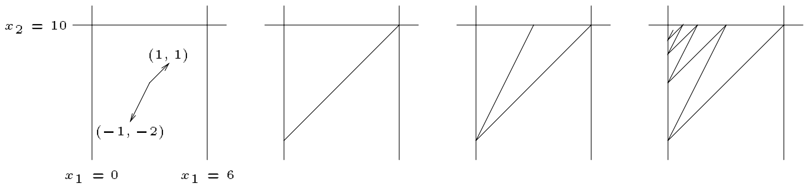


Figure 5: Phase portrait and BisimApprox for the water-level controller W_2

3.2 Finitary Euclidean Automata

To check if a euclidean automaton is finitary, it is convenient to use a phase interpretation of the procedure BisimApprox. Let E be a euclidean automaton of dimension n. The partition $Q/_\sim$ of $\mathbb{R}^n$ *is split along* the region $R \subseteq \mathbb{R}^n$ by splitting each region $R' \in Q/_\sim$ into the two parts $R' \cap R$ and $R' - R$ (one of the parts may be empty). We approximate the bisimilarity partition $Q/_\approx$ of E as follows:

```
1. Initialize $Q/_\sim$ to the result of splitting the state space $\mathbb{R}^n$ along the final region of $E$;
2. For every activity formula $\varphi$ of $E$, split $Q/_\sim$ along $dom(pre_\varphi)$;
3. For every action formula $\psi$ of $E$, split $Q/_\sim$ along $dom(pre_\psi)$;
4. Repeat until further splitting does not refine $Q/_\sim$:
   4.1. For every activity formula $\varphi$ of $E$ and every region $R \in Q/_\sim$, split $Q/_\sim$ along $pre_\varphi(R)$;
   4.2. For every action formula $\psi$ of $E$ and every region $R \in Q/_\sim$, split $Q/_\sim$ along $pre_\psi(R)$;
5. Return $Q/_\sim$.
```

This procedure depends solely on the phase portrait of E, and not on the trajectories of E. Indeed, in two dimensions, a quick visual inspection of the phase portrait suffices to determine if E is finitary: the region $R \in Q/_\sim$ needs to be split with respect to the activity or action formula σ iff there are two states in R whose sets of possible σ-successor regions in $Q/_\sim$ are different.

Example 3. Let t_4: $10 \leq x_1 \leq 40 \wedge x_2 = 10$ define the final region of the temperature controller T. Then the procedure BisimApprox yields the finite partition of the state space that is shown in Figure 4. The result illustrates that the final region $[\![t_4]\!]$ is reachable from all states that satisfy t_5: $0 \leq x_2 \leq 10$. It follows that $[\![t_5]\!]$ is the characteristic region of the temporal formula $\exists\Diamond t_4$, and a finite-state controller can be built to bring the system from any initial state in $[\![t_5]\!]$ to a final state in $[\![t_4]\!]$ (see full paper). If we choose, instead, the final region $[\![10 \leq x_1 \leq 40]\!]$, then the procedure BisimApprox does not converge (see title page). □

Grid automata. In order to study the activities of euclidean automata, we begin by limiting the power of actions. A linear constraint is *rectangular* if it is a conjunction of atomic constraints of the forms $x_i \geq a_i$ and $x_i \leq b_i$, for rational constants a_i and b_i. The activity formula $\varphi = \varphi_P \wedge \varphi_S$ is *rectangular* if the point invariant φ_P is rectangular. A *rectangular action formula* $\psi = \psi_G \wedge \psi_U \wedge \psi_F$ consists of a rectangular constraint ψ_G over X —the *guard*— a conjunction ψ_U of atomic constraints of the form $x'_i = x_i$ —the *update*— and a rectangular constraint ψ_F over X' —the *filter*. The linear euclidean automaton E is *rectangular* if all activity, action, and region formulas of E are rectangular.

The rectangular automaton E is a *grid automaton* if for each action formula ψ of E, the update ψ_U is either *true* or $\bigwedge_{i=1}^{n}(x'_i = x_i)$. The n-dimensional *finite grid* of unit $c \in \mathbb{Q}_{>0}$ and size $d \in \mathbb{N}$ consists of all $(n-1)$-dimensional regions that are defined by formulas of the form $x_i = k \cdot c$, for $k \in \mathbb{N}$ with $|k| \leq d$. During the procedure BisimApprox, the actions of the grid automaton E

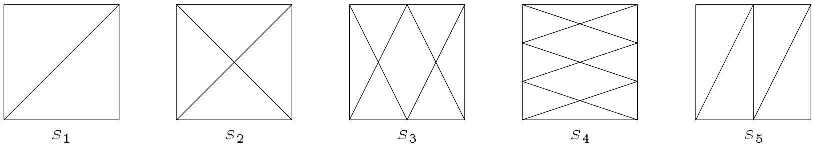


Figure 6: Periodic partitions of the unit cube

contribute, independent of its activities, a subset of the finite grid whose unit is determined by the g.c.d. of the constants in E, and whose size is determined by the size of the constants in E.

Proposition 8. *Every 1D rectangular automaton is finitary, and there is a bounded 2D grid automaton that is not finitary.*

Example 4. Recall the water-level controller W_2 from the introduction. The euclidean component of W_2 is a 2D grid automaton with the two activity formulas $x_2 \leq 10 \wedge \dot{x}_1 = 1 \wedge \dot{x}_2 = 1$ and $x_1 \geq 0 \wedge \dot{x}_1 = -1 \wedge \dot{x}_2 = -2$, and the two action formulas $x_2 = 10 \wedge x'_1 = x_1 \wedge x'_2 = x_2$ and $x_1 = 0 \wedge x'_1 = x_1 \wedge x'_2 = x_2$. If we assume the final region $\llbracket x_1 \leq 6 \rrbracket$, then Figure 5 illustrates that there is no finite bisimulation. (By contrast, if we assume the initial region $\llbracket x_1 = 0 \wedge x_2 = 10 \rrbracket$, then the reverse automaton W_2^{-1} has a trivial finite bisimulation.) □

Periodic automata. For periodic activities, it suffices to look at a single grid cell. The n-dimensional *unit cube* U^n is the region $[0,1]^n$ that is defined by the formula $\bigwedge_{i=1}^{n}(0 \leq x_i \leq 1)$. The unit cube has $2n$ facets of dimension $n-1$, each of which results from intersecting U^n with a region of the form $\llbracket x_i = 0 \rrbracket$ or $\llbracket x_i = 1 \rrbracket$. By closing the facets of U^n under intersection, we obtain the faces of U^n. In particular, the unit cube has 2^n faces of dimension 0 (i.e., corner points). The corner points of U^n and their negations are called the *boolean vectors* of dimension n. Given an $(n-2)$-dimensional region $R \subseteq \mathbb{R}^n$ and a vector $\mathbf{x} \in \mathbb{R}^n$, by $R \lhd \mathbf{x}$ we denote the $(n-1)$-dimensional region $\{\mathbf{y} - k \cdot \mathbf{x} \mid \mathbf{y} \in R \wedge k \in \mathbb{R}_{\geq 0}\}$. The slope set $S \subseteq \mathbb{R}^n$ is *periodic* if S is finite and there is a finite equivalence $\sim \subseteq U^n \times U^n$ on the unit cube such that (1) all faces of U^n are $\sim$-blocks and (2) for all $\sim$-equivalence classes $R \subseteq U^n$, all boolean vectors $\mathbf{u} \in \mathbb{R}^n$, and all slope vectors $\mathbf{x} \in S$, the region $((R + \mathbf{u}) \lhd \mathbf{x}) \cap U^n$ is a $\sim$-block. For example, Figure 6 shows that the 2D slope sets

$$\begin{aligned}
&S_1 = \{(1,1),(1,0),(0,1)\},\\
&S_2 = \{(1,1),(1,-1),(-1,1),(-1,-1)\},\\
&S_3 = \{(1,2),(1,-2),(-1,2),(-1,-2)\},\\
&S_4 = \{(3,1),(-3,1),(3,-1),(-3,-1)\},\\
&S_5 = \{(1,2),(-1,-2),(0,1),(0,-1)\}
\end{aligned}$$

are periodic. The periodic slope sets are closed under subsets and product (see full paper), but not under union. The rectangular automaton E is *periodic* if the slope set of E is periodic.

Theorem 9. *Every bounded periodic grid automaton is finitary.*

Figure 7 shows the result of applying the procedure BisimApprox to a bounded grid automaton with the slope set S_2 (the grid size is determined by the constants of the automaton). It follows that bounded grid automata with the slope set S_2 can be generalized to admit, in addition to grid actions, also rectangular action formulas with atomic filter constraints of the form $x'_i = c_i$, for $c_i \in \mathbb{Q}$. The result of the procedure BisimApprox is stable under these actions. Similarly, action formulas such as $x_2 = c \wedge x'_1 = x_1 \wedge x'_2 = d$ are admissible with the slope set S_3. In the full paper, we study which actions are safe for periodic slope sets. In particular, with the slope set

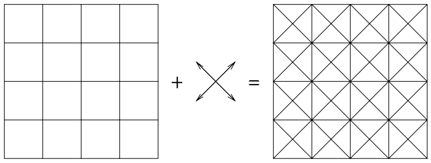


Figure 7: Phase portrait and BisimApprox for a bounded grid automaton with slope set S_2

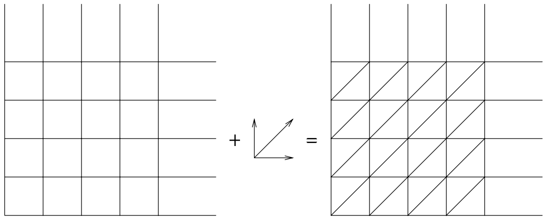


Figure 8: Phase portrait and BisimApprox for a positive grid automaton with slope set S_1

S_1 all rectangular action formulas are safe, and we can relax the condition of boundedness to the condition of positiveness. The resulting phase portrait, shown in Figure 8, is the region quotient of timed automata [3].

Clock and stopwatch automata. We next look at the euclidean components of several generalizations of timed automata. Let E be a positive rectangular automaton of dimension n. The variable x_i is a *drifting clock* if there are two slopes $a_i, b_i \in \mathbb{Q}_{\geq 0}$ such that each slope invariant of E contains the conjunct $a_i \leq \dot{x}_i \leq b_i$, and no other occurrences of $\dot{x}_i$. If $a_i = b_i$, then x_i is a *clock*; if $a_i = b_i = 1$, then x_i is a *precise clock*. The variable x_i is a *stopwatch* if there are two slopes $a_i, b_i \in \mathbb{Q}_{\geq 0}$ such that each slope invariant of E contains either the conjunct $a_i \leq \dot{x}_i \leq b_i$ or the conjunct $\dot{x}_i = 0$, and no other occurrences of $\dot{x}_i$. The positive rectangular automaton E is a *(drifting) clock automaton* if all variables of E are (drifting) clocks. The positive rectangular automaton E is a *stopwatch automaton* if all variables of E are stopwatches and each slope invariant of E implies $\dot{x}_i = 0$ for all but one $1 \leq i \leq n$. Stopwatch automata are useful for modeling real-time multi-tasking [20].

Theorem 10. *Every clock automaton and every stopwatch automaton is periodic and finitary, and there is a bounded 2D drifting-clock automaton that is not finitary.*

Proof. To show the first claim, we construct a finite bisimulation $\approx$ by generalizing the region quotient of timed automata [3]. Let $k \in \mathbb{N}$ be the least common multiple of all constants that occur in the given automaton. For state $\mathbf{x} \in \mathbb{R}^n$, the next significant event of $\mathbf{x}$ has index i if, when started in $\mathbf{x}$, of all variables x_i is the one whose fractional part will first pass either the fractional part of another variable or an integer boundary. In the case of a stop-watch automaton, i is the index of the one stopwatch with $\dot{x}_i \neq 0$. The two states $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ are $\approx$*-equivalent* if (1) $\mathbf{x}$ and $\mathbf{y}$ agree on the integer parts of all variables that do not exceed k; (2) $\mathbf{x}$ and $\mathbf{y}$ agree on the ordering of the fractional parts of all variables; and (3) $\mathbf{x}$ and $\mathbf{y}$ agree on the index of the next significant event. Figure 8 shows the $\approx$-quotient in the case of two precise clocks.

To show the second claim, consider the 2D automaton with the precise clock x_1 and the drifting clock x_2, where $1 \leq \dot{x}_2 \leq 2$. Figure 9 illustrates that there is no finite bisimulation (the horizontal and vertical lines represent actions). □

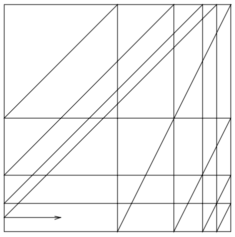


Figure 9: BisimApprox for a drifting-clock automaton

Figure 8, which shows the bisimilarity quotient in the case of two precise clocks, indicates that automata with two precise clocks can be generalized to admit, in addition to clock activities, also slope invariants of the forms $\dot{x}_1 = \dot{x}_2$, $0 \leq \dot{x}_1 \leq \dot{x}_2 \leq 1$, etc. (see full paper). The result of the procedure BisimApprox is stable under activities with these slope invariants. If, however, two clocks with different slopes are compared by a guard, filter, or point invariant, or both a clock and a stopwatch advance according to a slope invariant, then emptiness is undecidable [2, 12], and it is not difficult to check that the procedure BisimApprox does not terminate. Finitariness, on the other hand, is only sufficient but not necessary for decidability; indeed, the emptiness problem is decidable for drifting-clock automata [12, 24].

Directed automata. While not periodic, the euclidean component of the water-level controller W_1 from the introduction is nonetheless finitary. The slope set $S \subseteq \mathbb{R}^n$ is *i-directed*, for $1 \leq i \leq n$, if $x_i \geq 0$ for all $\mathbf{x} \in S$; the slope set S is *$-i$-directed* if $x_i \leq 0$ for all $\mathbf{x} \in S$. For example, a 2-dimensional automaton with the precise clock x_1 and the precise stopwatch x_2 has the 1-directed slope set $S = \{(1,0),(1,1)\}$. The region $R \subseteq \mathbb{R}^n$ is *i-bounded* if there is a constant $b_i \in \mathbb{R}$ such that $x_i \leq b_i$ for all $\mathbf{x} \in R$; the region R is *$-i$-bounded* if there is an $a_i \in \mathbb{R}$ such that $x_i \geq a_i$ for all $\mathbf{x} \in R$. The rectangular automaton E is *directed* if there is an i such that all point invariants of E define i-bounded regions, and the slope set of E is both finite and i-directed.

Theorem 11. *Every 2-dimensional directed grid automaton is finitary.*

Proof. Let (δ, f) be a curve that follows the slopes in a directed slope set. The proof is based on the observation that the crossing points of (δ, f) with the integer grid lines cannot converge. In the full paper we extend this result to n dimensions. □

Example 5. The euclidean component of the water-level controller W_1 is a grid automaton with the 1-directed slope set $\{(1,1),(1,-2)\}$ and 1-bounded point invariants. Figure 10 illustrates the finite bisimilarity relation of W_1 for the final region $[\![1 \leq x_2 \leq 12]\!]$. From that we can construct a finite-state controller that keeps the water level between 1 m and 12 m. □

Time-abstract products of finitary automata. Let $E_1 = (m, \Phi_1 \uplus \Psi_1, \overleftarrow{\rho_1}, \overrightarrow{\rho_1})$ and $E_2 = (n, \Phi_2 \uplus \Psi_2, \overleftarrow{\rho_2}, \overrightarrow{\rho_2})$ be two euclidean automata. Given two region formulas $\rho_1(x_1, \ldots, x_m)$ and $\rho_2(x_1, \ldots, x_n)$, let $\gamma(\rho_1, \rho_2)$ stand for the region formula $\rho_1(x_1, \ldots, x_m) \wedge \rho_2(x_{m+1}, \ldots, x_{m+n})$, and adopt analogous conventions for activity and action formulas. The *product* $E_1 \times_\gamma E_2$ is the euclidean automaton with the dimension $m+n$, the alphabet $\gamma(\Phi_1, \Phi_2) \uplus \gamma(\Psi_1, \Psi_2)$, the initial region formula $\gamma(\overleftarrow{\rho_1}, \overleftarrow{\rho_2})$, and the final region formula $\gamma(\overrightarrow{\rho_1}, \overrightarrow{\rho_2})$.

Proposition 12. *For all euclidean automata E_1 and E_2, $[\![E_1 \times_\gamma E_2]\!]_T = [\![E_1]\!]_T \times [\![E_2]\!]_T$.*[5]

[5] The product of two trajectory sets T_1 and T_2 contains all trajectory pairs $(\underline{\tau}_1, \underline{\tau}_2)$ such that $\underline{\tau}_1 \in T_1$, $\underline{\tau}_2 \in T_2$, $\underline{\tau}_1$ and $\underline{\tau}_2$ have the same number of segments, and corresponding segments have the same duration.

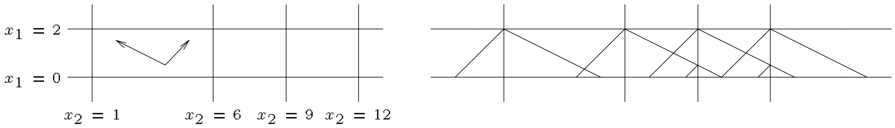


Figure 10: Phase portrait and BisimApprox for the water-level controller W_1

As the transition relation of a euclidean automaton abstracts the duration of transitions, the transition relation of the product $E_1 \times_\gamma E_2$ typically is more restrictive than the product of the individual transition relations. This observation motivates the following subclasses of euclidean automata. The euclidean automaton E can be *slowed down* (*sped up*) if for all activity formulas $\varphi \in \Phi$, the activity $[\![\varphi]\!]$ can be slowed down (sped up). The euclidean automaton E is *time-abstract* if it can be both slowed down and sped up —i.e., all activities of E are time-abstract. It is useful to model system components in a time-abstract way if the time scales of individual system components are not known a priori. For instance, unlike $\dot{x}_1 = 1 \wedge \dot{x}_2 = 1$, the slope invariant $\dot{x}_1 = \dot{x}_2$ defines a time-abstract activity with the same phase portrait [12]. Similarly, the slope invariants $\dot{x}_1 + 2\dot{x}_2 \geq 5$ and $\dot{x}_1 \leq 5 \wedge \dot{x}_2 = \dot{x}_1$ define activities that can be sped up and slowed down, respectively.

Theorem 13. *Let E_1 and E_2 be two euclidean automata such that one of E_1 and E_2 is time-abstract, or both E_1 and E_2 can be slowed down, or both E_1 and E_2 can be sped up. Then $\langle E_1 \times_\gamma E_2 \rangle = \langle E_1 \rangle \times_\gamma \langle E_2 \rangle$.*

Corollary 14. *Under the conditions of Theorem 13, $[\![E_1 \times_\gamma E_2]\!]_L = [\![E_1]\!]_L \times_\gamma [\![E_2]\!]_L$.*[6]

It follows that under the conditions of Theorem 13, if both E_1 and E_2 are effective and finitary, then so is the product $E_1 \times_\gamma E_2$.

4 Hybrid Automata

A *hybrid automaton* $H = (B, E)$ consists of a finite (or boolean[7]) automaton B and a euclidean automaton E over a common alphabet Σ. The branching-time semantics of H is the transition system $\langle H \rangle = B \times_\gamma \langle E \rangle$, where $\gamma(\sigma, \sigma) = \sigma$, and $\gamma(\sigma_1, \sigma_2)$ is undefined for $\sigma_1 \neq \sigma_2$. The linear-time semantics of H is the language $[\![H]\!]_L = [\![B]\!]_L \cap [\![E]\!]_L$ over the alphabet Σ.

Proposition 3 implies that if the euclidean automaton E is effective or finitary, then so is the hybrid automaton (B, E). It follows that the procedures BisimApprox and ReachApprox are semi-decision procedures for the emptiness problem of hybrid automata with effective euclidean components, and both are decision procedures in the finitary case. The procedure ReachApprox has been implemented, in HyTech, for hybrid automata with linear euclidean components [4].

If E is a clock automaton with precise clocks, then (B, E) is a *timed automaton* [3]. It is not difficult to check that, in addition to timed automata, also the multirate automata of [2, 21], the suspension automata of [20], and the 1-integrator automata of [6] have finitary euclidean components. We have thus provided an alternative, uniform decidability proof for these classes of hybrid automata. Unlike the original proofs, which are based on clock-translation or digitization techniques, our proof via finitariness provides several advantages: (1) it is not restricted to closed

[6] The product of two language sets L_1 and L_2 contains all words $\gamma(\underline{\sigma}_1, \underline{\sigma}_2)$ such that $\underline{\sigma}_1 \in L_1$, $\underline{\sigma}_2 \in L_2$, $\underline{\sigma}_1$ and $\underline{\sigma}_2$ have the same length, and corresponding formulas have the same type (activity or action).

[7] A finite automaton with 2^m locations can be viewed as *boolean automaton* of dimension m, which has the state space $\mathbb{B}^m$.

regions; (2) it provides direct insight into the state spaces of hybrid automata by identifying bisimilar states; (3) it guarantees the termination of successive-approximation procedures, such as those implemented in HyTech; and (4) it reduces problems on hybrid automata, such as model checking and control, to the corresponding problems on finite automata.

We close with two remarks. First, as pointed out before, finitariness is sufficient but not necessary for decidability. In particular, hybrid automata with infinite 1-counter encodable bisimulations are decidable, because emptiness can be reduced to the emptiness problem for pushdown automata [8]. Examples of such automata include 2D grid automata with the slope set S_2 and only one bounded variable (see full paper). Second, we expect that our phase view of hybrid systems will also lead to a theory of conservative approximations for nonlinear hybrid automata with the property that, unlike in the time domain [11], approximation errors do not accumulate.

Acknowledgments. Our view of hybrid systems has been shaped in collaboration with many people, including Rajeev Alur, Costas Courcoubetis, Pei-Hsin Ho, Peter Kopke, Amir Pnueli, Joseph Sifakis, Anuj Puri, and Pravin Varaiya. We also thank Oded Maler and Howard Wong-Toi for valuable comments.

References

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

[2] R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. *Hybrid Systems*, Springer LNCS 736, pp. 209–229, 1993.

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

[4] R. Alur, T.A. Henzinger, P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Real-time Systems Symp.*, pp. 2–11, 1993.

[5] P. Abdulla, B. Jonsson. Verifying programs with unreliable channels. *IEEE Symp. Logic in Computer Science*, pp. 160–170, 1993.

[6] A. Bouajjani, R. Echahed, R. Robbana. Verifying invariance properties of timed systems with duration variables. *Formal Techniques in Real-time and Fault-tolerant Systems*, Springer LNCS 863, pp. 193–210, 1994.

[7] A. Bouajjani, J.-C. Fernandez, N. Halbwachs. Minimal model generation. *Computer-aided Verification*, Springer LNCS 531, pp. 197–203, 1990.

[8] A. Bouajjani, R. Robbana. Verifying ω-regular properties for subclasses of linear hybrid systems. *Computer-aided Verification*, Springer LNCS, 1995.

[9] Z. Chaochen, C.A.R. Hoare, A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–276, 1991.

[10] T.A. Henzinger, P.-H. Ho. Model-checking strategies for linear hybrid systems. Workshop on Hybrid Systems and Autonomous Control (Ithaca, NY), 1994.

[11] T.A. Henzinger, P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. *Computer-aided Verification*, Springer LNCS, 1995.

[12] T.A. Henzinger, P. Kopke, A. Puri, P. Varaiya. What's decidable about hybrid automata? *ACM Symp. Theory of Computing*, 1995.

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

[14] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximation. *Static Analysis Symp.*, Springer LNCS 864, 1994.

[15] Y. Kesten, A. Pnueli, J. Sifakis, S. Yovine. Integration graphs: a class of decidable hybrid systems. *Hybrid Systems*, Springer LNCS 736, pp. 179–208, 1993.

[16] P.C. Kanellakis, S.A. Smolka. CCS expressions, finite-state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[17] D. Lee, M. Yannakakis. Online minimization of transition systems. *ACM Symp. Theory of Computing*, pp. 264–274, 1992.

[18] O. Maler, A. Pnueli. Reachability analysis of planar multi-linear systems. *Computer-aided Verification*, Springer LNCS 697, pp. 194–209, 1993.

[19] O. Maler, A. Pnueli, J. Sifakis. On the synthesis of discrete controllers for timed systems. *Theoretical Aspects of Computer Science*, Springer LNCS, 1995.

[20] J. McManis, P. Varaiya. Suspension automata: a decidable class of hybrid automata. *Computer-aided Verification*, Springer LNCS 818, pp. 105–117, 1994.

[21] X. Nicollin, A. Olivero, J. Sifakis, S. Yovine. An approach to the description and analysis of hybrid systems. *Hybrid Systems*, Springer LNCS 736, pp. 149–178, 1993.

[22] A. Olivero, J. Sifakis, S. Yovine. Using abstractions for the verification of linear hybrid systems. *Computer-aided Verification*, Springer LNCS 818, pp. 81–94, 1994.

[23] R. Paige, R.E. Tarjan. Three partition-refinement algorithms. *SIAM J. Computing*, 16:973–989, 1987.

[24] A. Puri, P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. *Computer-aided Verification*, Springer LNCS 818, pp. 95–104, 1994.