Computing Simulations on Finite and Infinite Graphs*†

Monika R. Henzinger Thomas A. Henzinger Peter W. Kopke

Computer Science Department, Cornell University
{mhr,tah,pkpk}@cs.cornell.edu

July 5, 1996

Abstract. We present algorithms for computing similarity relations of labeled graphs. Similarity relations have applications for the refinement and verification of reactive systems. For finite graphs, we present an $O(mn)$ algorithm for computing the similarity relation of a graph with n vertices and m edges (assuming $m \geq n$). For effectively presented infinite graphs, we present a symbolic similarity-checking procedure that terminates if a finite similarity relation exists. We show that 2D rectangular automata, which model discrete reactive systems with continuous environments, define effectively presented infinite graphs with finite similarity relations. It follows that the refinement problem and the $\forall$CTL* model-checking problem are decidable for 2D rectangular automata.

1 Introduction

A *labeled graph* $G = (V, E, A, \langle\langle\cdot\rangle\rangle)$ consist of a (possibly infinite) set V of vertices, a set $E \subseteq V^2$ of edges, a set A of labels, and a function $\langle\langle\cdot\rangle\rangle : V \to A$ that maps each vertex v to a label $\langle\langle v\rangle\rangle$. We write $post(v) = \{u \mid (v, u) \in E\}$ for the successor set of the vertex v. A binary relation $\leq \subseteq V^2$ on the vertex set is a *simulation* if $u \leq v$ implies (1) $\langle\langle u\rangle\rangle = \langle\langle v\rangle\rangle$ and (2) for all vertices $u' \in post(u)$, there is a vertex $v' \in post(v)$ such that $u' \leq v'$. The vertex v *simulates* the vertex u if there is a simulation $\leq$ such that $u \leq v$ [31]. The vertices u and v are *similar*, written $u \approx^S v$, if u simulates v and v simulates u. The similarity relation $\approx^S \subseteq V^2$ is an equivalence relation. We consider the problem of computing the similarity relation $\approx^S$.

Motivation. Labeled graphs are useful for modeling reactive systems: the vertices represent system states, the edges represent system moves, and the labels represent observations such as variable values or I/O events. Simulations arise in two situations of formal system design and analysis—*system refinement* and *system abstraction*. First, a system G_1 refines (or implements) a specification G_2 if every start state of G_1 is simulated by a start state of G_2 [1]. Then, in an infinite two-player game of system versus specification, each move of the system G_1 can be matched by a move of the specification G_2 that leads to the same observation. Second, the algorithmic analysis

*This research was supported in part by the NSF grants CCR-9200794 and CCR-9504469, by the NSF CAREER awards CCR-9501708 and CCR-9501712, by the AFOSR contract F49620-93-1-0056, by the ONR YIP award N00014-95-1-0520, and by the ARPA grant NAG2-892.

†A preliminary version of this paper appeared in the *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science* (FOCS 95), October 1995, pp. 453–462.

of large, or infinite, state spaces is practical, or possible, only if we are able to identify "equivalent" system states. The resulting quotient graph, which represents an abstraction of the system, may be much smaller than the full system graph, and may even be finite for infinite system graphs. The notion of "state equivalence" depends, of course, on the class of system properties that are considered [5, 13, 18]. In verification, two particularly important state equivalences are trace equivalence and bisimilarity.

Trace equivalence. The vertex v *trace-dominates* the vertex u if for every finite u-rooted path $\overline{u}$ there is a v-rooted path $\overline{v}$ such that $\langle\langle\overline{u}\rangle\rangle = \langle\langle\overline{v}\rangle\rangle$.[1] The vertices u and v are *trace-equivalent*, written $u \approx^T v$, if u trace-dominates v and v trace-dominates u. Trace equivalence is an equivalence relation that is coarser than similarity; that is, $u \approx^S v$ implies $u \approx^T v$, but not vice versa. Trace equivalence is important, because two vertices are trace-equivalent iff they satisfy the same formulas of linear temporal logic, and the quotient graph $G/_{\approx^T}$ therefore suffices to check linear temporal properties of the system G. Trace equivalence, however, is difficult to compute: the problem of checking if two vertices of a finite labeled graph are trace equivalent is PSPACE-complete [34].

Bisimilarity. A binary relation $\cong \subseteq V^2$ is a *bisimulation* if $u \cong v$ implies (1) $\langle\langle u\rangle\rangle = \langle\langle v\rangle\rangle$, (2) for all vertices $u' \in post(u)$, there is a vertex $v' \in post(v)$ such that $u' \cong v'$, and (3) for all vertices $v' \in post(v)$, there is a vertex $u' \in post(u)$ such that $u' \cong v'$. The vertices u and v are *bisimilar*, written $u \approx^B v$, if there is a bisimulation $\cong$ such that $u \cong v$ [32]. Bisimilarity is an equivalence relation that is finer than similarity; that is, $u \approx^B v$ implies $u \approx^S v$, but not vice versa. Bisimilarity is important, because two vertices are bisimilar iff they satisfy the same formulas of branching temporal logic, and the quotient graph $G/_{\approx^B}$ therefore suffices to check branching temporal properties of the system G. In addition, bisimilarity is easier to compute than trace equivalence: the Paige-Tarjan algorithm checks in time $O(m \log n)$ if two vertices of a finite labeled graph G with n vertices and m edges are bisimilar (assuming $m \geq n$) [33]. However, since most interesting system properties can be expressed in linear temporal logic, bisimilarity usually provides an unnecessarily weak reduction of the state space.

Similarity. We argue that in many cases, neither trace equivalence nor bisimilarity, but similarity is the appropriate abstraction for computer-aided verification. (Process algebra, which studies the behavior of state equivalences under various operations on reactive systems, provides additional justification for using equivalences that are closely related to similarity [7].)

First, for finite-state systems, the similarity quotient can be computed in polynomial time, and for effectively presented infinite-state systems, the similarity quotient can be computed symbolically. We present an $O(mn)$ algorithm for checking if two vertices of the finite labeled graph G are similar, and a symbolic procedure for computing the similarity quotients of labeled graphs that, finite or infinite, are presented effectively. Symbolic procedures are essential for the success of computer-aided verification, which typically deals with infinite state spaces, or with state spaces that are too large to be enumerated [11]. Our symbolic similarity-checking procedure uses the same primitives as symbolic minimization procedures for computing bisimilarity quotients [9, 28], and the procedure terminates if the input graph has a finite similarity quotient.

Second, since similarity lies strictly between trace equivalence and bisimilarity, it provides a better state-space reduction than bisimilarity, and the similarity quotient is still adequate for checking all linear temporal formulas. Indeed, two vertices are similar iff they satisfy the same formulas of branching temporal logic without quantifier switches [17]. Hence the similarity quotient is adequate for checking, say, all formulas of the branching temporal logic $\forall$CTL*, which contain only universal path quantifiers [20].

Third, infinite labeled graphs may be presented effectively using the formalism of hybrid au-

[1] Let $\langle\langle u_1 u_2 \ldots u_n\rangle\rangle = \langle\langle u_1\rangle\rangle\langle\langle u_2\rangle\rangle \cdots \langle\langle u_n\rangle\rangle$.

tomata for modeling reactive systems with discrete and continuous components [2]. Within hybrid automata, a maximal subclass with a decidable graph reachability problem are rectangular automata [25]. We show that 2D rectangular automata define infinite-state systems with infinite bisimilarity quotients, yet finite similarity quotients. This result gives a structural explanation for the decidability of reachability for rectangular automata, and shows that also refinement and $\forall$CTL* model-checking are decidable for 2D rectangular automata. Our work suggests that similarity is the natural state equivalence for the analysis and abstract interpretation of rectangular automata. We therefore plan to implement our symbolic procedure for computing similarity quotients within HyTech, an automatic tool for the verification of hybrid automata [23].

Outline. This paper consists of two parts. In Section 2, we present an $O(mn)$ similarity-checking algorithm for finite graphs, and a symbolic similarity-checking algorithm for effectively presented infinite graphs. In Section 3, we show that all 2D rectangular automata have finite similarity relations.[2]

Related work. The model theory and the proof theory of similarity are studied extensively in process algebra [21, 26, 35], and in deductive approaches to the refinement of reactive systems [1, 29, 30]. Polynomial-time algorithms for computing similarity quotients are presented in [6] ($O(mn^6)$), [14] ($O(mn^4)$), and [15] ($O(m^2)$). Recently it has come to our attention that an $O(mn)$ algorithm was found independently by Bloom and Paige [8]. (All of these algorithms are set in somewhat different contexts, and solve somewhat different problems, but can be translated into our framework.) A symbolic procedure for model-checking $\forall$CTL* formulas is given in [17]. While that procedure can be used for computing similarity quotients symbolically, by evaluating increasingly larger $\forall$CTL* formulas, this approach does not seem practical.

2 Similarity Checking

2.1 Finite graphs

We compute for each vertex v the *simulator set* $sim(v)$ of vertices that simulate v. Then $u \approx^S v$ iff $v \in sim(u)$ and $u \in sim(v)$. We develop our algorithm in three steps. For a vertex v, we use the notation $pre(v)$ for the predecessor set $\{u \mid (u,v) \in E\}$ of v.[3]

Step 1. We start with the schema *SchematicSimilarity1* shown at the top of Figure 1. For each vertex v, the set $sim(v)$ contains vertices that are candidates for simulating v. Initially, $sim(v)$ contains all vertices with the label of v. If $(u,v) \in E$ and $w \in sim(u)$, but there is no $w' \in sim(v)$ such that $(w,w') \in E$, then w cannot simulate u and is removed from $sim(u)$. In this case, we say that $sim(u)$ is *sharpened* with respect to the edge (u,v). It is easy to check that if no edges allow a sharpening of $sim(u)$ for any vertex u, then for all v, all vertices in $sim(v)$ can simulate v. If the input graph has n vertices, there can be at most n^2 iterations of the WHILE loop. A naive implementation of the algorithm *SchematicSimilarity1* therefore requires time $O(m^2n^3)$, where $m \geq n$ is the number of edges in the input graph. We will improve the running time to $O(mn)$.

Step 2. The procedure *RefinedSimilarity* refines the schema *SchematicSimilarity1* as shown in the center of Figure 1. The key idea of the refinement is the introduction of a set $prevsim(v)$ for each vertex v. For each vertex v, the set $prevsim(v)$ is a superset of $sim(v)$ and contains vertices that once were considered candidates for simulating v. The crucial invariant I2 of the WHILE loop allows us to sharpen $sim(u)$ with respect to the edge (u,v) by looking only at vertices in $prevsim(v)$ when

[2] An equivalence relation is *finite* if it has a finite number of equivalence classes.

[3] For a set U of vertices, let $pre(U) = \cup_{u \in U} pre(u)$.

procedure *SchematicSimilarity1*:
  Input: a labeled graph $G = (V, E, A, \langle\langle\cdot\rangle\rangle)$.
  Output: for each vertex $v \in V$, the simulator set $sim(v)$.

  for all $v \in V$ **do** $sim(v) := \{u \in V \mid \langle\langle u\rangle\rangle = \langle\langle v\rangle\rangle\}$ **od**;
  while there are three vertices u, v, and w such that
        $v \in post(u)$, $w \in sim(u)$, and $post(w) \cap sim(v) = \emptyset$ **do**
    $sim(u) := sim(u) \backslash \{w\}$
    od.

procedure *RefinedSimilarity*:
  for all $v \in V$ **do**
    $prevsim(v) := V$;
    if $post(v) = \emptyset$
      then $sim(v) := \{u \in V \mid \langle\langle u\rangle\rangle = \langle\langle v\rangle\rangle\}$
      else $sim(v) := \{u \in V \mid \langle\langle u\rangle\rangle = \langle\langle v\rangle\rangle$ and $post(u) \neq \emptyset\}$
      fi
    od;
  while there is a vertex $v \in V$ such that $sim(v) \neq prevsim(v)$ **do**
    {I1: **assert** for all $v \in V$, $sim(v) \subseteq prevsim(v)$}
    {I2: **assert** for all $u, v, w \in V$, if $(u, v) \in E$ and $w \in sim(u)$, then $post(w) \cap prevsim(v) \neq \emptyset$}
    $remove := pre(prevsim(v)) \backslash pre(sim(v))$;
    for all $u \in pre(v)$ **do** $sim(u) := sim(u) \backslash remove$ **od**;
    $prevsim(v) := sim(v)$
    od.

procedure *EfficientSimilarity*:
  for all $v \in V$ **do**
    {**let** $prevsim(v) := V$}
    if $post(v) = \emptyset$
      then $sim(v) := \{u \in V \mid \langle\langle u\rangle\rangle = \langle\langle v\rangle\rangle\}$
      else $sim(v) := \{u \in V \mid \langle\langle u\rangle\rangle = \langle\langle v\rangle\rangle$ and $post(u) \neq \emptyset\}$
      fi;
    $remove(v) := pre(V) \backslash pre(sim(v))$
    od;
  while there is a vertex $v \in V$ such that $remove(v) \neq \emptyset$ **do**
    {I3: **assert** for all $v \in V$, $remove(v) = pre(prevsim(v)) \backslash pre(sim(v))$}
    for all $u \in pre(v)$ **do**
      for all $w \in remove(v)$ **do**
        if $w \in sim(u)$ **then**
          $sim(u) := sim(u) \backslash \{w\}$;
          for all $w'' \in pre(w)$ **do**
            if $post(w'') \cap sim(u) = \emptyset$ **then** $remove(u) := remove(u) \cup \{w''\}$ **fi**
            od
          fi
        od
      od;
    {**let** $prevsim(v) := sim(v)$}
    $remove(v) := \emptyset$
    od.

Figure 1: $O(mn)$ similarity checking

checking if a vertex $w \in sim(u)$ has a successor in $sim(v)$. Moreover, once $w' \in prevsim(v) \backslash sim(v)$ is looked at once, w' is removed from $prevsim(v)$ forever.

The initial FOR loop of *RefinedSimilarity* performs, in addition to the work of the initial FOR loop of *SchematicSimilarity1*, also some of the work of the WHILE loop of *SchematicSimilarity1*. For each vertex v, the set $prevsim(v)$ is initialized to contain all vertices, and $sim(v)$ is initialized to contain all vertices with the label of v that have a successor if v does. This initialization establishes the two invariants I1 and I2. In each iteration of the WHILE loop, we nondeterministically pick a vertex v for which $sim(v)$ improves on $prevsim(v)$, and we sharpen $sim(u)$ for all predecessors u of v with respect to the edge (u, v). By I2, all vertices in $sim(u)$ have successors in $prevsim(v)$, and we can find all vertices in $sim(u)$ that do not have successors in $sim(v)$ by looking at the predecessor set of $prevsim(v)$. These vertices are collected in the set *remove* and deleted from $sim(u)$. Once all predecessors of v have been processed in this fashion, we update $prevsim(v)$ to $sim(v)$. If $sim(v) = prevsim(v)$ for all vertices v, then I2 implies the termination condition of *Schematic-Similarity1*.

Step 3. The algorithm *EfficientSimilarity*, shown at the bottom of Figure 1, implements the procedure *RefinedSimilarity* using two data structures. First, instead of recomputing the set *remove* in each iteration of the WHILE loop, we dynamically maintain for each vertex v a set $remove(v)$ that satisfies the invariant I3 of the WHILE loop. If $remove(v) = \emptyset$ for all vertices v, then I1 and I3 imply the termination condition of *RefinedSimilarity*. Second (not shown in the figure), we maintain a 2D array $count[1..n, 1..n]$ of nonnegative integers such that $count[w'', u] = |post(w'') \cap sim(u)|$ for all vertices w'' and u. The array *count* is initialized in time $O(mn)$. Whenever a vertex w is removed from $sim(u)$, then the counters $count[w'', u]$ are decremented for all predecessors w'' of w. The cost of these decrements is absorbed in the cost of the innermost IF statement. With the array *count*, the test $post(w'') \cap sim(u) = \emptyset$ of that IF statement can be executed in constant time, by checking if $count[w'', u] = 0$.

The initialization of $sim(v)$ for all v requires time $O(n^2)$ (recall that $n \leq m$). The initialization of $remove(v)$ for all v requires time $O(mn)$. Given two vertices v and w, if the test $w \in remove(v)$ is positive in iteration i of the WHILE loop, then the test $w \in remove(v)$ is negative in all iterations $j > i$. This is because (1) in all iterations, $w \in remove(v)$ implies that $w \notin pre(sim(v))$, (2) the value of $prevsim(v)$ in all iterations $j > i$ is a subset of the value of $sim(v)$ in iteration i, and (3) invariant I1. It follows that the test $w \in sim(u)$ is executed $\Sigma_v \Sigma_w |pre(v)| = O(mn)$ times. The test $w \in sim(u)$ is positive at most once for every w and u, because after a positive test w is removed from $sim(u)$ and never put back. Therefore the body of the outer IF statement in the WHILE loop contributes time $\Sigma_w \Sigma_u (1 + |pre(w)|) = O(mn)$. This gives a total running time of $O(mn)$.

Theorem 2.1 *Given a finite labeled graph with n vertices and $m \geq n$ edges, the algorithm* Efficient-Similarity *computes the simulator sets for all vertices in time $O(mn)$.*

Corollary 2.1 *The similarity of two vertices of a finite labeled graph can be decided in time $O(mn)$.*

2.2 Infinite graphs

While enumerative procedures operate on data structures that represent graphs as collections of vertices and edges, symbolic procedures operate on data structures that represent the vertex and edge sets of graphs using symbolic constraints such as logical formulas (in the finite case, boolean formulas). Symbolic procedures apply to infinite graphs also. For example, for the infinite graph with the vertex set $\mathbb{R}$, the symbolic constraint $1 \leq u \leq 2$ represents the infinite set $[1, 2] \subseteq \mathbb{R}$ of

procedure *SchematicSimilarity2*:
Input: a labeled graph $G = (V, E, A, \langle\langle\cdot\rangle\rangle)$.
Output: for each vertex $v \in V$, the simulator set $sim(v)$.

for all $v \in V$ **do** $sim(v) := \{u \in V \mid \langle\langle u\rangle\rangle = \langle\langle v\rangle\rangle\}$ **od**;
while there are three vertices u, v, and w such that
$post(u) \cap sim(v) \neq \emptyset$, $w \in sim(u)$, and $post(w) \cap sim(v) = \emptyset$ **do**
{I4: **assert** for all $v \in V$, $v \in sim(v)$}
{I5: **assert** for all $v, w', w'' \in V$, if $w' \leq w''$ and $w' \in sim(v)$, then $w'' \in sim(v)$}
$sim(u) := sim(u) \backslash \{w\}$
od.

procedure *SymbolicSimilarity*:
Input: a labeled graph $G = (V, E, A, \langle\langle\cdot\rangle\rangle)$.
Output: the symbolic simulator structure (Π, Sim) for G.

$\Pi := \{V_a \mid a \in A \text{ and } V_a \neq \emptyset\}$;
for all $U \in \Pi$ **do** $Sim(U) := U$ **od**;
while there are two regions $U, V' \in \Pi$ such that $U \cap pre(Sim(V')) \neq \emptyset$ and $Sim(U) \backslash pre(Sim(V')) \neq \emptyset$ **do**
{I6: **assert** for all $U \in \Pi$ and all $v \in U$, $sim(v) = Sim(U)$}
{I7: **assert** for all $U \in \Pi$, both U and $Sim(U)$ are blocks of $\approx^S$}
$(U', U'') := (U \cap pre(Sim(V')), U \backslash pre(Sim(V')))$;
$\Pi := (\Pi \backslash \{U\}) \cup \{U'\}$;
$Sim(U') := Sim(U) \cap pre(Sim(V'))$;
if $U'' \neq \emptyset$ **then** $\Pi := \Pi \cup \{U''\}$; $Sim(U'') := Sim(U)$ **fi**
od.

Figure 2: Symbolic similarity checking

vertices, and the symbolic constraint $1 \leq u \leq 2 \wedge u' = u + 1$ represents an infinite set of edges, one from each vertex $u \in [1, 2]$ to the vertex $u + 1$.

A *region* is a (possibly infinite) set of vertices. Symbolic procedures operate on regions, rather than vertices. Instead of computing simulator sets for individual vertices, we compute simulator sets for entire regions. Note the following two facts. First, if two vertices are similar, then their simulator sets are identical. Second, the simulator set of every vertex is a block (i.e., a union of equivalence classes) of the similarity relation $\approx^S$. These two facts lead us to the following definition. Given a labeled graph G, the *symbolic simulator structure* $(\Pi, Sim : \Pi \to \mathbf{2}^\Pi)$ for G consists of (1) the partition Π of the vertex set V that is induced by the similarity relation $\approx^S$, and (2) for each equivalence class $U \in \Pi$, the set $Sim(U)$ of vertices that that simulate any (equivalently, all) vertices in U. Then Π is a set of regions, and Sim maps each of these regions to another region.

For a label a, let $V_a = \{v \in V \mid \langle\langle v\rangle\rangle = a\}$ be the region of vertices with the label a. The labeled graph G is *effective* if there is a class $\mathcal{R}$ of effectively representable regions such that (1) $V_a \in \mathcal{R}$ for all $a \in A$, (2) $\mathcal{R}$ is effectively closed under all boolean operations and the *pre*-operation, and (3) the emptiness problem is decidable for the regions in $\mathcal{R}$. Given an effective labeled graph G, we compute the symbolic simulator structure (Π, Sim) for G. We develop our procedure in two steps.

Step 1. We start with the schema *SchematicSimilarity2* shown at the top of Figure 2, which relaxes the schema *SchematicSimilarity1* of Figure 1. The initial FOR loops are identical, and establish the two invariants I4 and I5. The invariant I5 asserts that whenever a simulator set $sim(v)$ contains a vertex w', and w'' simulates w', then $sim(v)$ contains also w''. Assuming I5, if

$w \in sim(u)$, $w' \in sim(v)$, and $(u, w') \in E$, but there is no $w'' \in sim(v)$ such that $(w, w'') \in E$, then w cannot simulate u. This is because in order for w to simulate u, some successor of w would have to simulate w', which is not possible, because by I5 all vertices that simulate w' are contained in $sim(v)$. We can therefore remove w from $sim(u)$, maintaining both invariants, even if *SchematicSimilarity1* would not have allowed us to do so. In this case, we say that $sim(u)$ is *freely sharpened* with respect to the edge (u, w'). If the edge (u, v) allows a sharpening of $sim(u)$, then I4 implies that (u, v) also allows a free sharpening of $sim(u)$. Consequently, if no edges allow a free sharpening of $sim(u)$ for any vertex u, then the termination condition of *SchematicSimilarity1* is satisfied. This implies the partial correctness of *SchematicSimilarity2*.

Step 2. The procedure *SymbolicSimilarity*, shown at the bottom of Figure 2, is an instance of the schema *SchematicSimilarity2*. The only primitive operations of *SymbolicSimilarity* are boolean operations and the *pre*-operation on regions, and emptiness checking of regions. The procedure *SymbolicSimilarity* can therefore be executed for finite labeled graphs, and for infinite labeled graphs that are effective. If the similarity relation $\approx^S$ of the input graph is finite, then it has only finitely many blocks, and the invariant I7 ensures that the procedure *SymbolicSimilarity* terminates. (If $\approx^S$ is infinite, then the partition Π needs to be refined infinitely often, and the procedure does not terminate.)

In implementing the procedure *SymbolicSimilarity*, we can enforce the invariant that for all regions $U \in \Pi$, the region $Sim(U)$ is a block of Π, by refining the partition Π whenever this becomes necessary due to the creation of a new simulator set. Such an implementation maintains a finite partition Π of the vertex set V together with pointers from each region $U \in \Pi$ to all regions $W \in \Pi$ with $W \subseteq Sim(U)$, without represententing the simulator set $Sim(U)$ explicitly.

Theorem 2.2 *Given an effective labeled graph G with a finite similarity relation, the algorithm* SymbolicSimilarity *terminates and computes the symbolic simulator structure for G.*

Corollary 2.2 *The similarity of two vertices of an effective labeled graph can be decided.*

It follows that the refinement problem ("Does a system refine a specification?") is decidable if both the system and the specification are given by effective labeled graphs with finite similarity relations, provided the sets of start states form blocks of the initial partition $\Pi_0 = \{V_a \mid a \in A\}$. Also the $\forall$CTL* model-checking problem is decidable for such systems, provided all atomic formulas define blocks of the initial partition Π_0. The next section gives an example for a class of systems that define effective labeled graphs with finite similarity relations.

3 Hybrid Automata with Finite Similarity Relations

A hybrid automaton is a finite automaton in tandem with a dynamical system [2]. Hybrid automata are useful for the algorithmic analysis of discrete programs that interact with a continuous environment [4, 23]. Each hybrid automaton defines an infinite labeled graph. Verification must either proceed symbolically on the infinite state space, or otherwise reduce the state space to a finite quotient. Such a reduction is possible for *timed* (hybrid) automata, where all continuous variables are accurate clocks, because all timed automata have finite bisimilarity relations [3]. However, finite bisimilarity relations no longer exist for simple extensions of timed automata, such as *rectangular* (hybrid) automata, where all continuous variables are clocks with bounded drift [22]. Yet the reachability problem is known to be decidable for rectangular automata [25]. We explain this fact by showing that every 2D rectangular automaton has a finite similarity relation, which is the

intersection of the two finite bisimilarity relations obtained by looking at the extremal slopes of both drifting clocks. Since every rectangular automaton defines an effective labeled graph, it follows that the refinement problem and the $\forall$CTL* model-checking problem are decidable for 2D rectangular automata.

In a *generalized rectangular automaton*, the drifting clocks may take on negative slopes. We show that every 2D generalized rectangular automaton has a similarity quotient that, though infinite, tiles the plane in a regular manner. We conclude that linear temporal properties of 2D rectangular automata can be decided using pushdown automata.

3.1 Rectangular automata

Definition of rectangular automata [25]. An *n-dimensional rectangle* is a product of n nonempty intervals over $\mathbb{R}_{\geq 0}$ (open, half-open, or closed; bounded or unbounded), all of whose finite endpoints are integers. Let $\mathcal{B}^n$ be the set of n-dimensional rectangles. An *n-dimensional rectangular automaton H* consists of a finite directed graph $(Loc, Trans)$, a bounded rectangle $act \in \mathcal{B}^n$, a vertex labeling function $inv\colon Loc \to \mathcal{B}^n$, and three edge labeling functions $preguard, postguard\colon Trans \to \mathcal{B}^n$ and $update\colon Trans \to 2^{\{1,\ldots,n\}}$. The vertices in Loc are called *locations*, and the edges in $Trans$ are called *transitions*.

The rectangular automaton H defines the infinite labeled graph $G_H = (V_H, E_H, Loc, \langle\langle\cdot\rangle\rangle)$. Each vertex $(\ell, \mathbf{x}) \in V_H$ consists of a discrete state $\ell \in Loc$ and a continuous state $\mathbf{x} \in \mathbb{R}^n_{\geq 0}$. When the discrete state is ℓ, the continuous state must lie in the invariant rectangle $inv(\ell)$. This gives us the vertex set $V_H = \{(\ell, \mathbf{x}) \mid \ell \in Loc \text{ and } \mathbf{x} \in inv(\ell)\}$. Each edge in E_H is either a time step or a transition step.

Time steps are constrained by the invariant function inv and the activity rectangle act: define $(\ell, \mathbf{x}) \to^{\text{time}} (\ell', \mathbf{y})$ iff (1) $\ell = \ell'$ and (2) either $\mathbf{x} = \mathbf{y}$, or there exists a positive real $t \in \mathbb{R}_{>0}$ such that $\frac{\mathbf{y}-\mathbf{x}}{t} \in act$. Due to the convexity of rectangles, it follows that $(\ell, \mathbf{x}) \to^{\text{time}} (\ell', \mathbf{y})$ iff $\ell = \ell'$ and there exists a differentiable trajectory $f : [0, t] \to inv(\ell)$ such that $f(0) = \mathbf{x}$, $f(t) = \mathbf{y}$, and the time derivative $\dot{f}(s) \in act$ for all $s \in (0, t)$. Thus a time step does not change the discrete state and involves a differentiable evolution in the continuous state.

Transition steps are constrained by the functions $preguard$, $postguard$, and $update$: define $(\ell, \mathbf{x}) \to^{\text{trans}} (\ell', \mathbf{y})$ iff there is a transition $e = (\ell, \ell') \in Trans$ such that (1) $\mathbf{x} \in preguard(e)$, (2) $\mathbf{y} \in postguard(e)$, and (3) for all $1 \leq i \leq n$, if $i \notin update(e)$, then $\mathbf{x}_i = \mathbf{y}_i$. Thus a transition step involves a change in the discrete state together with a discontinuous jump in the continuous state. If $i \notin update(e)$, then the ith coordinate $\mathbf{x}_i$ of the continuous state remains unchanged; otherwise, $\mathbf{x}_i$ is nondeterministically reassigned a new value in the ith component of the rectangle $postguard(e)$.

This gives us the edge set $E_H = \{(u, v) \in V_H^2 \mid u \to^{\text{time}} v \text{ or } u \to^{\text{trans}} v\}$. Finally, the label of a vertex is its location: for each $(\ell, \mathbf{x}) \in V_H$, $\langle\langle(\ell, \mathbf{x})\rangle\rangle = \ell$. Thus only the discrete state is observable. (In [24], we handle, in addition to observations of the discrete state, also rectangular observations of the continuous state.)

Effectiveness of rectangular automata [22]. The infinite labeled graph G_H is effective for every rectangular automaton H. To see this, consider the class of regions that are definable by boolean combinations of (1) locations in Loc and (2) quantifier-free formulas of the theory $(\mathbb{R}, +, \leq)$ of the reals with addition. The theory $(\mathbb{R}, +, \leq)$ is decidable, closed under boolean operations, and closed under the *pre*-operation for rectangular automata, which corresponds to quantifier elimination [4].

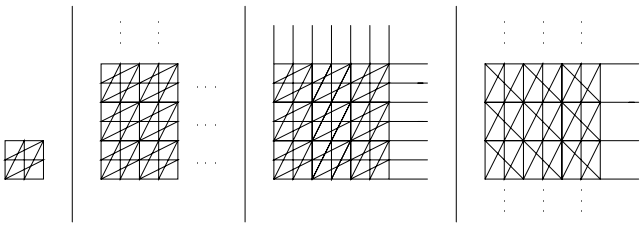

Figure 3: Tilings with the torus similarity relation

Similarity relations of 2D rectangular automata. We show that for every 2D rectangular automaton H, the infinite labeled graph G_H has a finite similarity relation. In the following, we assume that the activity rectangle *act* is closed, and no component of *act* contains 0. This has the effect that during time steps, the range of the derivative ratio $\frac{d\mathbf{x}_2}{d\mathbf{x}_1}$ is a closed bounded interval with positive rational endpoints, namely, the *phase slope interval* $I_{act} = [\frac{\inf act_2}{\sup act_1}, \frac{\sup act_2}{\inf act_1}]$ of H. We also assume that for all transitions $e \in \mathit{Trans}$, if $i \in \mathit{update}(e)$, then the ith component of the rectangle $\mathit{postguard}(e)$ is a singleton. This has the effect of making all discontinuous jumps deterministic. Open and unbounded phase slope intervals and nondeterministic jumps are handled in [24].

We begin by considering only the continuous part $\mathbb{R}^2_{\geq 0}$ of the state space. We first divide the plane into unit squares, and find a finite similarity quotient of each unit square considered as the torus T_2. This quotient on the unit square is shown in the first pane of Figure 3 for an automaton with the activity rectangle $act = [1,2]^2$; that is, $I_{act} = [\frac{1}{2}, 2]$. We then lift the similarity quotient to the positive portion $\mathbb{R}^2_{\geq 0}$ of the plane, obtaining a regular tessellation, as shown in the second pane of Figure 3. While this tesselation gives us an infinite quotient, we can make use of the fact that the definition of the given automaton H contains a largest constant $c \in \mathbb{N}$. Since both derivatives $d\mathbf{x}_1$ and $d\mathbf{x}_2$ are nonnegative, any two continuous states that differ only in the integer parts of coordinates larger than c are bisimilar. Hence two points in $\mathbb{R}^2_{\geq 0}$ are similar if (1) their fractional parts are similar as elements of the torus T_2, and (2) they agree on the integer parts of all coordinates that are no more than c. In our example, the third pane of Figure 3 shows the case $c = 3$ (except that each unbounded region is further partitioned according to (1), into finitely many patches). In a final step, we take the product of the similarity quotient on $\mathbb{R}^2_{\geq 0}$ with the discrete part *Loc* of the state space.

We now begin the formal presentation. For a nonnegative real $x \in \mathbb{R}_{\geq 0}$, let $\mathit{frac}(x)$ be the fractional part $x - \lfloor x \rfloor$ of x; for a point $\mathbf{x} \in \mathbb{R}^2_{\geq 0}$, let $\mathit{frac}(\mathbf{x})$ be the point whose ith coordinate is $\mathit{frac}(\mathbf{x}_i)$ for $i = 1, 2$. By gluing together opposite sides of the closed unit square $[0,1]^2$, we obtain the torus T_2. Define the equivalence relation $\equiv$ on $[0,1]^2$ by $\mathbf{x} \equiv \mathbf{y}$ iff $\mathit{frac}(\mathbf{x}) = \mathit{frac}(\mathbf{y})$. Then $T_2 = [0,1]^2/_{\equiv}$. We use the points in the half-open unit square $(0,1]^2$ to represent the elements of T_2.

Let a and b be positive rationals with $a \leq b$. For a point $\mathbf{x} \in \mathbb{R}^2_{\geq 0}$, by $\mathit{ray}_a(\mathbf{x}) \subset \mathbb{R}^2_{\geq 0}$ we denote the ray from $\mathbf{x}$ with slope a; that is, $\mathbf{y} \in \mathit{ray}_a(\mathbf{x})$ iff $\mathbf{y}_1 \geq \mathbf{x}_1$ and $(\mathbf{y}_1 - \mathbf{x}_1)a = \mathbf{y}_2 - \mathbf{x}_2$. By $\mathit{cone}_{a,b}(\mathbf{x}) \subset \mathbb{R}^2_{\geq 0}$ we denote the cone with the two boundary rays $\mathit{ray}_a(\mathbf{x})$ and $\mathit{ray}_b(\mathbf{x})$; that is, $\mathbf{y} \in \mathit{cone}_{a,b}(\mathbf{x})$ iff either $\mathbf{y} = \mathbf{x}$, or $\mathbf{y}_1 > \mathbf{x}_1$ and $a \leq \frac{\mathbf{y}_2 - \mathbf{x}_2}{\mathbf{y}_1 - \mathbf{x}_1} \leq b$ (see the left part of Figure 4). We now define a labeled graph $G_{a,b}$ with the vertex set $(0,1]^2$. The edges of $G_{a,b}$ correspond to the two types of movement for the continuous state of a 2D rectangular automaton with the phase slope interval $[a,b]$. Formally, $G_{a,b} = ((0,1]^2, E^{\text{time}} \cup E_1^{\text{trans}} \cup E_2^{\text{trans}}, A, \langle\!\langle \cdot \rangle\!\rangle)$.

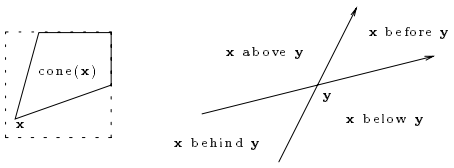

Figure 4: Left: $cone(\mathbf{x})$. Right: Partition of $\mathbb{R}^2_{\geq 0}$.

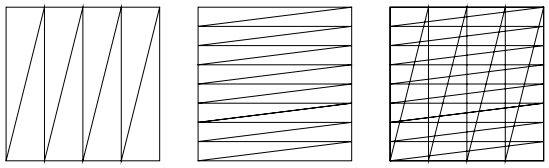

Figure 5: The region bisimulations for slopes 4 and $\frac{1}{8}$, and their intersection

Time steps Continuous evolution is represented by edges of the type E^{time}: define $(\mathbf{x}, \mathbf{y}) \in E^{\text{time}}$ iff there is a point $\mathbf{x}' \in [0,1]^2$ such that $\mathbf{x}' \equiv \mathbf{x}$ and $\mathbf{y} \in cone_{a,b}(\mathbf{x}')$. It follows that boundary points of the unit square can move "around" the torus T_2 once. (A continuous evolution that moves around the torus multiple times is reducible to a sequence of E^{time} edges [24].)

Transition steps Discontinuous jumps of the ith coordinate are represented by edges of the type E_i^{trans}, for $i = 1,2$: define $(\mathbf{x}, \mathbf{y}) \in E_i^{\text{trans}}$ iff (1) $\mathbf{y}_i = 1$, and (2) for $j \neq i$, $\mathbf{y}_j = \mathbf{x}_j$. (A discontinuous jump in both coordinates is reducible to a sequence of two E^{trans} edges [24].)

The labels of two points are equal iff they have the same status with respect to being on the boundaries of the unit square: let A be any set with four elements, and define $\langle\langle \mathbf{x} \rangle\rangle = \langle\langle \mathbf{y} \rangle\rangle$ iff (1) $\mathbf{x}_1 = 1$ iff $\mathbf{y}_1 = 1$, and (2) $\mathbf{x}_2 = 1$ iff $\mathbf{y}_2 = 1$.

If $a = b$, then the labeled graph $G_{a,b}$ has the finite *region bisimulation* $\cong_a$ [2, 3], which is defined by the line on the torus T_2 that begins at the origin and moves at slope a until it meets the origin once again. Then horizontal and vertical lines are drawn from each intersection with the coordinate axes. Formally, for $a = \frac{a_1}{a_2}$ with $\gcd(a_1, a_2) = 1$, define $\mathbf{x} \cong_a \mathbf{y}$ iff for $i = 1,2$, (1) $\lfloor a_i \mathbf{x}_i \rfloor = \lfloor a_i \mathbf{y}_i \rfloor$, (2) $frac(a_i \mathbf{x}_i) = 0$ iff $frac(a_i \mathbf{y}_i) = 0$, and (3) for $j \neq i$, $frac(a_i \mathbf{x}_i) < frac(a_j \mathbf{x}_j)$ iff $frac(a_i \mathbf{y}_i) < frac(a_j \mathbf{y}_j)$. Figure 5 shows the region bisimulations $\cong_4$ and $\cong_{\frac{1}{8}}$, and on the right hand side, their intersection. We prove that this intersection is the similarity relation of $G_{a,b}$.

Lemma 3.1 *Let a and b be two positive rationals with $a \leq b$. The two vertices $\mathbf{x}$ and $\mathbf{y}$ of the labeled graph $G_{a,b}$ are similar iff $\mathbf{x} \cong_a \mathbf{y}$ and $\mathbf{x} \cong_b \mathbf{y}$.*

Proof. We argue only that $\mathbf{x} \cong_a \mathbf{y}$ and $\mathbf{x} \cong_b \mathbf{y}$ implies that $\mathbf{x}$ and $\mathbf{y}$ are similar; for the converse see [24]. Consider simulation as an infinite two-player game between an evader and a pursuer playing on the torus T_2. The evader begins at the point $\mathbf{x}$, and the pursuer begins at $\mathbf{y}$. When the evader moves to $\mathbf{x}'$, the pursuer must move to a point $\mathbf{y}'$ with $\mathbf{y}'_i = 1$ iff $\mathbf{x}'_i = 1$ for $i = 1,2$; that is, the pursuer must be able to match the moves of the evader that fall on a boundary of the unit square. The evader wins if ever the pursuer cannot match its move. The pursuer wins the

infinite game if it matches every evader move, ad infinitum. In particular, the pursuer can win if it can ever move to the same point as the evader. It is clear that $\mathbf{y}$ simulates $\mathbf{x}$ iff the pursuer has a winning strategy.

An optimal strategy for the pursuer is as follows. When it is above the evader, the pursuer moves at the minimal slope, waiting for the evader to enter its cone. Similarly, when it is below the evader, the pursuer moves at the maximal slope. Once the evader enters the cone of the pursuer, the pursuer intercepts the evader by moving to the same point.

With this in mind, we define a relation $\leq$ on the torus T_2, and show that $\leq$ is a simulation for $G_{a,b}$. Given a point $\mathbf{y} \in \mathbb{R}^2_{\geq 0}$, we partition the positive portion $\mathbb{R}^2_{\geq 0}$ of the plane into four disjoint parts:

(1) define $\mathbf{x}$ *before* $\mathbf{y}$ iff $\mathbf{x} \in \mathit{cone}_{a,b}(\mathbf{y})$ and $\mathbf{x} \notin \mathit{ray}_a(\mathbf{y})$ and $\mathbf{x} \notin \mathit{ray}_b(\mathbf{y})$;
(2) define $\mathbf{x}$ *above* $\mathbf{y}$ iff not (1) and $\mathit{cone}_{a,b}(\mathbf{x}) \cap \mathit{ray}_a(\mathbf{y}) = \emptyset$;
(3) define $\mathbf{x}$ *below* $\mathbf{y}$ iff not (1) and $\mathit{cone}_{a,b}(\mathbf{x}) \cap \mathit{ray}_b(\mathbf{y}) = \emptyset$;
(4) define $\mathbf{x}$ *behind* $\mathbf{y}$ iff $\mathbf{y} \in \mathit{cone}_{a,b}(\mathbf{x})$.

This partition of the plane is shown in the right part of Figure 4. For $\mathbf{x}, \mathbf{y} \in (0,1]^2$, define $\mathbf{x} \leq \mathbf{y}$ iff the following four conditions obtain:

(C1) Either $\mathbf{y}$ *above* $\mathbf{x}$ and $\mathbf{y} \cong_a \mathbf{x}$, or $\mathbf{y}$ *below* $\mathbf{x}$ and $\mathbf{y} \cong_b \mathbf{x}$, or $\mathbf{y} \cong_a \mathbf{x}$ and $\mathbf{y} \cong_b \mathbf{x}$.
(C2) For $i = 1, 2$, $\mathbf{y}_i = 1$ iff $\mathbf{x}_i = 1$.
(C3) $\mathbf{y}_2 > \mathbf{x}_2$ implies $(1, \mathbf{y}_2) \cong_a (1, \mathbf{x}_2)$, and $\mathbf{y}_2 < \mathbf{x}_2$ implies $(1, \mathbf{y}_2) \cong_b (1, \mathbf{x}_2)$.
(C4) $\mathbf{y}_1 > \mathbf{x}_1$ implies $(\mathbf{y}_1, 1) \cong_b (\mathbf{x}_1, 1)$, and $\mathbf{y}_1 < \mathbf{x}_1$ implies $(\mathbf{y}_1, 1) \cong_a (\mathbf{x}_1, 1)$.

The first condition is the root of the matter. It says that $\mathbf{y}$ simulates $\mathbf{x}$ if either $\mathbf{y}$ and $\mathbf{x}$ are bisimilar at both extremal slopes; or $\mathbf{y}$ is below $\mathbf{x}$, and $\mathbf{y}$ and $\mathbf{x}$ are bisimilar at the maximal slope; or $\mathbf{y}$ is above $\mathbf{x}$ and $\mathbf{y}$ and $\mathbf{x}$ are bisimilar at the minimal slope.

We now return to pursuit game terminology. The pursuer begins at $\mathbf{y}$, and attempts to match the moves of the evader, who begins at $\mathbf{x}$. We sketch a winning strategy for the pursuer when $\mathbf{x} \leq \mathbf{y}$ (see [24] for details). First we consider jump edges from E_i^{trans}. Suppose that the evader moves to $\mathbf{x}'$ such that $(\mathbf{x}, \mathbf{x}') \in E_i^{\text{trans}}$. Then the pursuer moves to $\mathbf{y}'$ such that $(\mathbf{y}, \mathbf{y}') \in E_i^{\text{trans}}$. By condition C2 + i of the definition of $\leq$, it follows that $\mathbf{x}' \leq \mathbf{y}'$.

Next we consider evolution edges from E^{time}. Assume that $\mathbf{y}$ *above* $\mathbf{x}$ and $\mathbf{y} \cong_a \mathbf{x}$, and suppose that the evader moves to $\mathbf{x}'$ such that $(\mathbf{x}, \mathbf{x}') \in E^{\text{time}}$. If $\mathbf{x}' \in \mathit{cone}_{a,b}(\mathbf{y}')$ for some $\mathbf{y}' \equiv \mathbf{y}$, then the pursuer moves to the same point $\mathbf{x}'$, and wins the game. Otherwise, the pursuer moves at the minimal slope a. Since $\mathbf{y} \cong_a \mathbf{x}$, and the $\cong_a$-equivalence classes are convex, it follows that the pursuer can reach a point $\mathbf{y}'$ with $\mathbf{x}' \leq \mathbf{y}'$. The case $\mathbf{y}$ *below* $\mathbf{x}$ and $\mathbf{y} \cong_b \mathbf{x}$ is handled symmetrically. The final case has $\mathbf{y} \cong_a \mathbf{x}$ and $\mathbf{y} \cong_b \mathbf{x}$, but neither $\mathbf{y}$ *above* $\mathbf{x}$ nor $\mathbf{y}$ *below* $\mathbf{x}$. In this case, either $\mathbf{x}$ *behind* $\mathbf{y}$ or $\mathbf{x}$ *before* $\mathbf{y}$. In the former subcase, the pursuer moves to $\mathbf{x}'$, catching the evader. In the latter subcase, suppose that neither $\mathbf{y} \cong_a \mathbf{x}'$ nor $\mathbf{y} \cong_b \mathbf{x}'$. Then there is a point $\mathbf{x}^*$ on the line segment from $\mathbf{x}$ to $\mathbf{x}'$ such that either $\mathbf{y}$ *above* $\mathbf{x}^*$ and $\mathbf{y} \cong_a \mathbf{x}^*$, or $\mathbf{y}$ *below* $\mathbf{x}^*$ and $\mathbf{y} \cong_b \mathbf{x}^*$. So the pursuer moves as indicated in one of the first two cases, to counter an evader move from $\mathbf{x}^*$ to $\mathbf{x}'$. ■

Let $a = \frac{a_1}{a_2}$ and $b = \frac{b_1}{b_2}$ such that $\gcd(a_1, a_2) = \gcd(b_1, b_2) = 1$. The similarity relation of $G_{a,b}$ has $n_{a,b} = O((a_1 + b_1)(a_2 + b_2))$ many equivalence classes.

Let H be a 2D rectangular automaton with k locations and the phase slope interval $[a, b]$. Let $\cong_{a,b}$ be the similarity relation of $G_{a,b}$. Let c be the largest integer constant that appears in the definition of H. Then the two vertices $(\ell, \mathbf{x})$ and $(\ell', \mathbf{y})$ of G_H are similar if (1) $\ell = \ell'$, (2) $\mathit{frac}(\mathbf{x}) \cong_{a,b} \mathit{frac}(\mathbf{y})$, and (3) for $i = 1, 2$, either $\lfloor \mathbf{x}_i \rfloor = \lfloor \mathbf{y}_i \rfloor$, or both $\mathbf{x}_i > c$ and $\mathbf{y}_i > c$. The similarity relation of G_H has $O(kc^2 n_{a,b}) = O(kc^4)$ many equivalence classes. If integer constants

are represented in logarithmic space, then the size of the similarity quotient is exponentially larger than the description of the automaton.

Theorem 3.1 *Let H be a 2D rectangular automaton with k locations and integer constants no larger than c. The labeled graph G_H has a finite similarity relation with $O(kc^4)$ many equivalence classes.*

Corollary 3.1 *The algorithm* SymbolicSimilarity *terminates when applied to a 2D rectangular automaton.*

By contrast, 2D rectangular automata generally do not have finite bisimilarity relations [22], and symbolic CTL model-checking procedures [4] may not terminate when applied to a 2D rectangular automaton. We conjecture that rectangular automata of arbitrary dimension have finite similarity relations.

$\forall$CTL* model checking. Let H be a rectangular automaton with the location set *Loc* and the invariant function *inv*. Consider the branching temporal logic $\forall$CTL* [20] whose atomic formulas are the locations in *Loc* (for more generous atomic formulas, see [24]). Then every formula ϕ defines a region $[\![\phi]\!]_H$ of the infinite labeled graph G_H (the atomic formula ℓ defines the region $[\![\ell]\!]_H = \{(\ell, \mathbf{x}) \mid \mathbf{x} \in \mathit{inv}(\ell)\}$). The model-checking problem (H, ϕ) asks if the region $[\![\phi]\!]_H$ is equal to the state space V_H. Since two similar vertices cannot be distinguished by $\forall$CTL* formulas, it follows from Theorem 3.1 that the $\forall$CTL* model-checking problem for 2D rectangular automata can be reduced to model-checking on finite labeled graphs. From the model-checking complexity of CTL* [27], it follows that the $\forall$CTL* model-checking problem for 2D rectangular automata can be solved in PSPACE.

3.2 Generalized rectangular automata

The rectangles of a *generalized 2D rectangular automaton* H are not restricted to the positive portion $\mathbb{R}^2_{\geq 0}$ of the plane. In particular, the activity rectangle $act = act_1 \times act_2$ of H is a product of two arbitrary intervals $act_1, act_2 \subseteq \mathbb{R}$ of the real line. For a detailed treatment of generalized rectangular automata, we refer to [24]; here we only sketch our results.

First consider the case in which only one of the coordinates may take on negative derivatives; that is, $0 < \inf act_1 \leq \sup act_1$ and $\inf act_2 < 0 < \sup act_2$. In this case, $I_{act} = [a, b]$ for the two rationals $a = \frac{\inf act_2}{\inf act_1} < 0$, and $b = \frac{\sup act_2}{\inf act_1} > 0$. The analogue of Lemma 3.1 still holds. Let c be the largest integer constant that appears in the definition of H, and consider a point $\mathbf{x} \in \mathbb{R}_{\geq 0} \times \mathbb{R}$. While the integer part of $\mathbf{x}_1$ is relevant only up to c, this is no longer the case for the integer part of $\mathbf{x}_2$. The similarity quotient of $\mathbb{R}_{\geq 0} \times \mathbb{R}$ consists of a strip that is infinite in one dimension, and tessellated by the torus similarity quotient on each unit square of the strip. This is illustrated in the final pane of Figure 3 for an automaton with $I_{act} = [-1, 2]$ and $c = 3$. Then, as above, the similarity relation $\cong_{a,b}$ of $G_{a,b}$ induces the similarity relation of G_H: the two vertices $(\ell, \mathbf{x})$ and $(\ell', \mathbf{y})$ of G_H are similar if (1) $\ell = \ell'$, (2) $\mathit{frac}(\mathbf{x}) \cong_{a,b} \mathit{frac}(\mathbf{y})$, (3) $\lfloor \mathbf{x}_2 \rfloor = \lfloor \mathbf{y}_2 \rfloor$, and (4) either $\lfloor \mathbf{x}_1 \rfloor = \lfloor \mathbf{y}_1 \rfloor$, or both $\mathbf{x}_1 > c$ and $\mathbf{y}_1 > c$. The similarity quotient of G_H is infinite, but can be encoded by a pushdown ω-automaton whose stack represents the absolute value of the integer part for the second coordinate. (This technique is used in [10] for a similar purpose.)

It remains to consider the case in which both coordinates may take on both positive and negative derivatives; that is, $\inf act_1 < 0 < \sup act_1$, and $\inf act_2 < 0 < \sup act_2$. In this case, $I_{act} = \mathbb{R}$; that is, each point in $\mathbb{R}^2$ may evolve into any direction. Then G_H has a finite bisimilarity relation [22].

Consider the linear temporal logic LTL [19] whose atomic formulas are the locations of H and the rectangles in $\mathbb{R}^2$. From our characterization of the similarity relation of G_H, it follows that

the LTL model-checking problem for generalized 2D rectangular automata can be reduced to the language inclusion problem between a pushdown ω-automaton and a finite ω-automaton, which is known to be decidable [16].

Theorem 3.2 *The LTL model-checking problem is decidable for generalized 2D rectangular automata.*

Acknowledgment. The authors thank Rance Cleaveland for providing many valuable pointers to the literature, in particular to [8] and to [12], which presents an $O(n^3)$ algorithm for computing more general preorders than simulations on finite graphs.

References

[1] M. Abadi, L. Lamport. The existence of refinement mappings. 3rd LICS, IEEE, 165–175, 1988.

[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, 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. 14th RTSS, ACM, 2–11, 1993.

[5] S. Bensalem, A. Bouajjani, C. Loiseaux, J. Sifakis. Property-preserving simulations. 4th CAV, Springer LNCS 663, 260–273, 1992.

[6] B. Bloom. *Ready simulation, bisimulation, and the semantics of CCS-like languages.* PhD thesis, MIT, 1989.

[7] B. Bloom, S. Istrail, A.R. Meyer. Bisimulation can't be traced. 15th POPL, ACM, 229–239, 1988.

[8] B. Bloom, R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Science of Computer Programming*, to appear.

[9] A. Bouajjani, J.-C. Fernandez, N. Halbwachs. Minimal model generation. 2nd CAV, Springer LNCS 531, 197–203, 1990.

[10] A. Bouajjani, R. Robbana. Verifying ω-regular properties for subclasses of linear hybrid systems. 7th CAV, Springer LNCS 939, 437–450, 1995.

[11] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[12] U. Celikkan, R. Cleaveland. Generating diagnostic information for behavioral preorders. *Distributed Computing*, to appear.

[13] E.M. Clarke, O. Grümberg, D.E. Long. Model checking and abstraction. 19th POPL, ACM, 343–354, 1992.

[14] R.J. Cleaveland, J. Parrow, B. Steffen. The Concurrency Workbench: a semantics-based tool for the verification of finite-state systems. *ACM TOPLAS*, 15:36–72, 1993.

[15] R.J. Cleaveland, B. Steffen. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[16] R.S. Cohen, A.Y. Gold. Theory of ω-languages I: characterizations of ω-context-free languages. *JCSS*, 15:169–184, 1977.

[17] D.R. Dams, O. Grümberg, R. Gerth. Generation of reduced models for checking fragments of CTL. 5th CAV, Springer LNCS 697, 479–490, 1993.

[18] D.R. Dams, O. Grümberg, R. Gerth. Abstract interpretation of reactive systems: abstractions preserving $\forall$CTL*, $\exists$CTL*, and CTL*. *IFIP Working Conference on Programming Concepts, Methods, and Calculi*, Elsevier, 1994.

[19] E.A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, vol. B, Elsevier, 995–1072, 1990.

[20] O. Grümberg, D.E. Long. Model checking and modular verification. 2nd CONCUR, Springer LNCS 527, 250–265, 1991.

[21] M.C.B. Hennessy, R. Milner. Algebraic laws for nondeterminism and concurrency. *JACM*, 32:137–161, 1985.

[22] T.A. Henzinger. Hybrid automata with finite bisimulations. 22nd ICALP, Springer LNCS 944, 324–335, 1995.

[23] T.A. Henzinger, P.-H. Ho, H. Wong-Toi. HyTech: The next generation. 16th RTSS, IEEE, 1995.

[24] T.A. Henzinger, P.W. Kopke. Hybrid automata with finite mutual simulations. CSD-TR-95-1497, Cornell Univ., 1995.

[25] T.A. Henzinger, P.W. Kopke, A. Puri, P. Varaiya. What's decidable about hybrid automata? 27th STOC, ACM, 373–382, 1995.

[26] B. Jonsson. Simulations between specifications of distributed systems. 2nd CONCUR, Springer LNCS 527, 346–360, 1991.

[27] O. Kupferman. *Model Checking for Branching-time Temporal Logics*. PhD thesis, The Technion, 1995.

[28] D. Lee, M. Yannakakis. Online minimization of transition systems. 24th STOC, ACM, 264–274, 1992.

[29] N.A. Lynch, M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. 6th PODC, ACM, 137–151, 1987.

[30] N.A. Lynch, F. Vaandrager. Forward and backward simulations for timing-based systems. *Real Time: Theory in Practice*, Springer LNCS 600, 397–446, 1992.

[31] R. Milner. An algebraic definition of simulation between programs. 2nd IJCAI, British Computer Society, 481–489, 1971.

[32] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

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

[34] L.J. Stockmeyer, A.R. Meyer. Word problems requiring exponential time. 5th STOC, ACM, 1–9, 1973.

[35] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, Vrije Universiteit te Amsterdam, 1990.