A Decidable Class of Planar Linear Hybrid Systems

Pavithra Prabhakar, Vladimeros Vladimerou, Mahesh Viswanathan, and Geir E. Dullerud

University of Illinois at Urbana-Champaign.

Abstract. The paper shows the decidability of the reachability problem for planar, monotonic, linear hybrid automata without resets. These automata are a special class of linear hybrid automata with only two variables, whose flows in all states is monotonic along some direction in the plane, and in which the continuous variables are not reset on a discrete transition.

1 Introduction

The use of embedded devices in safety critical systems, has prompted extensive research in the formal modeling and analysis of hybrid systems. *Hybrid automata* [1] are a widely used formalism for modeling such systems. These are machines with finitely many control states and finitely many real-valued variables that evolve continuously with time. The transitions depend on the values of the continuous variables and they change both the discrete control state as well as the values of the variables. The safety of systems modelled by such automata can often be reduced to the question of whether a certain state or *region* of the state space can be reached during an execution. This is called the *reachability problem*.

Due to its importance, the reachability problem for hybrid automata has been carefully investigated in the past couple of decades. The problem has been shown to be decidable for special kinds of hybrid automata including *timed automata* [2], certain special classes of *rectangular hybrid automata* [6], and *o-minimal hybrid automata* [8]. These decidability results often rely on demonstrating the existence of a finite, computable partition of the state space that is *bisimilar* to the original system.

However, such decidability results are the exception rather than the norm. The reachability problem remains stubbornly undecidable even for very simple and special classes of hybrid automata, not just in the general case. One such special class is that of *linear hybrid automata*. In these automata each variable is constrained to evolve along a constant slope (with time), and despite such simple dynamics, have been unamenable to algorithmic analysis even in low dimension (i.e., with very few continuous variables). Timed automata, where each variable evolves synchronously with a global clock, but where the machine is allowed to compare clock values at the time of discrete transitions [1], is undecidable even for systems with 6 clocks [2]. The case of general linear hybrid automata in which variables are constrained to be compared only to constants, remains undecidable even for just 3 variables [1]. Undecidability results for dynamical

[1] The decidability result for timed automata holds when clocks are only compared with constants.

systems with piecewise constant derivative in 3 dimensions, and piecewise affine maps in 2 dimensions [5] provide further evidence.

In this paper, we prove the decidability for a special class of linear hybrid automata that are *planar, monotonic* and *don't have resets*. Planar refers to the fact that the automata has only two variables. Monotonic refers to the fact that we require the existence of a vector ρ such that the derivatives of the variables (viewed as a vector in the plane) in all states have a positive projection along ρ; note, this does not mean that both variables have positive derivatives in each state. Finally, the automaton does not reset/change the values of the variables when taking a discrete transition.

The automaton model that we consider here is more general in some aspects, and at the same time more restrictive in some aspects, when compared with other hybrid automata models for which decidability results are known. First variables are not restricted to clocks, like timed automata. Second, variables are not required to have the same slope in all states, or for them to be reset when the flow is changed, as in some rectangular hybrid automata. Next, transitions don't have strong resets that decouple the continuous dynamics from the discrete, as in o-minimal systems. Finally, the guards and invariants are not required to be disjoint, as in dynamical systems with piecewise constant derivatives [3] or polygonal hybrid systems [4]. On the other hand, our automata only have 2 variables, no resets, and monotonic flows.

Despite the restrictive dynamics and planarity, the decidability proof is very challenging. Like many decidability proofs in this area, we first partition the plane into *regions*, which in our case are convex polygons formed by considering lines associated with the constraints appearing in the automaton description, and lines perpendicular to the direction along which the flow is monotonic. Such regions have a very special geometric structure in that they are bounded by 2 to 4 line segments, at least one of which is a line segment perpendicular to the monotonic direction. The first key idea in the proof is to observe the existence of a line ℓ, perpendicular to the monotonic direction, such that the behavior of the automaton beyond ℓ is bisimilar to a finite state system. Then reachability computation is broken up into two phases: the first phase computes all points before ℓ that are reachable, and the second phase constructs the finite bisimulation for the points beyond ℓ and does the search in the bisimilar transition system.

The computation of the reachable regions before ℓ itself relies on observing that any execution of the automaton can be seen as a concatenation of a series of *almost-inside executions*. An almost-inside execution is an execution that starts at the boundary of a region R, enters R, and then leaves to another boundary of R, all the while staying inside R, while taking both discrete and time steps. The first lemma we prove is that the effect of such almost-inside executions is computable for all regions. However, in order for the decidability proof to go through we need a stronger result for certain special regions that we call *right pinched triangles*; we need to show that the effect of concatenating finitely many almost-inside executions can be computed. We do this through a tree construction reminiscent of the Karp-Miller tree [7] for vector addition systems. Finally, we solve the reachability result for regions before ℓ by another tree construction. A carefully counting argument coupled with the monotonicity of flows ensures that this tree will be finite and hence effectively constructable. Space constraints prevent us from giving detailed proofs of the decidablity result here; complete proofs can be found in [9].

2 An example

We will first illustrate our algorithm for deciding reachability on an example. Consider the hybrid system $\mathbb{H}$ given in Figure 1. It has five locations $s_1, \cdots, s_5$, with flows $f_1, \cdots, f_5$, respectively, associated with them. The locations are labelled by their invariants. For example, the invariant associated with location s_1 is $y < 1$, and this says that the control of the system can be in s_1 only if the value of the variable y is less than 1. When in a certain location the values of the variables change according to their flow. If the system starts with $x = 0$ and $y = 0$ at location s_1, and spends a unit time, then the values of the variables would be $x = 1$ and $y = 2$. However in this case the system is forced by the invariant to leave the location before half time unit. We note that $\mathbb{H}$ is a *monotone linear* hybrid system, where by linear we mean that the flows associated with the locations are constants, and by monotone that the flows have a positive projection along some direction, in this case the x-axis as shown in Figure 2.

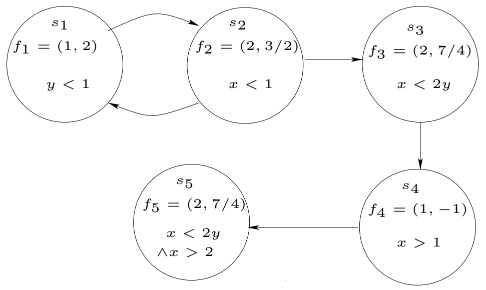

Fig. 1. Linear hybrid system $\mathbb{H}$

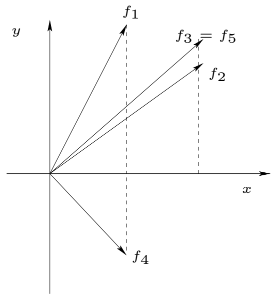

Fig. 2. Flows of the hybrid system $\mathbb{H}$

We will consider the following reachability problem: Is the location s_5 reachable starting from s_1 with $x = 0$ and $y = 0$? As shown in Figure 3, this translates to checking if starting in s_1 at point O, we can reach the shaded region in location s_5.

We first divide the plane into regions depending on the constraints in $\mathbb{H}$. Corresponding to each constraint of $\mathbb{H}$, there is a straight line, as shown by the solid lines in Figure 3. We also add lines parallel to the y-axis passing through the points of intersections of these lines, if one does not already exist. As is easily seen, the interior of a region is invariant with respect to the locations in that either it is contained in the invariant of a location or is disjoint from it. Hence with each element of a region which is its interior, its edge without the end-points or its vertex, we can associate a set of locations whose invariants contain the element. For example, the set of locations corresponding to the interior of region 1 is $\{s_1, s_2, s_3\}$.

The idea of the algorithm is to compute successors for the regions. Given a part of an edge, called a subedge, and a location, the successor with respect to a region is the set

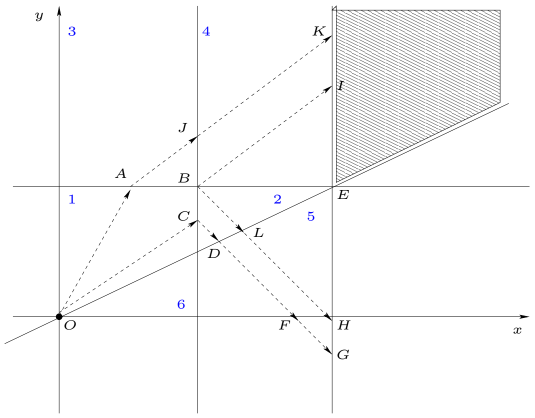

Fig. 3. Regions of the hybrid system $\mathbb{H}$

of all points on the boundary of the region reachable by moving only in its interior, and leaving and entering the boundary at most once. For example, starting from point A in location s_3, we can reach J by following flow f_3 of s_3 and moving only in the interior of region 3. Hence (s_3, J) is in the successor of (s_3, A). As a slightly more interesting example, consider the problem of finding the successors of point O in region 1. These are exactly the points between A and B in locations s_2 and s_3, the points between B and C in locations s_1 and s_3 and the point B in location s_3. We will represent this succinctly as $(s_1, B'C')$, $(s_2, A'B')$, (s_3, BC') and $(s_3, A'B)$, where A' indicates that point A itself is excluded. The above subedges are computed in the following way. The locations corresponding to region 1 are s_1, s_2 and s_3. Let us consider the underlying graph of $\mathbb{H}$ restricted to locations and guards which contain region 1. The same is shown in Figure 4. We observe that any path from O in location s_1 spends time alternately in s_1

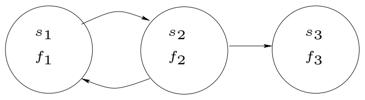

Fig. 4. Underlying graph of $\mathbb{H}$ restricted to region 1

and s_2, and then possibly makes a transition to s_3 where it spends additional time before reaching the boundary. We will show that the set of all points reachable by alternating between s_1 and s_2 is exactly the set of point in the cone generated by f_1 and f_2 which are also in the interior of region 1, namely, the points inside the parallelogram $OABC$ in the figure. This is true only because s_1 and s_2 belong to the same strongly connected component of the underlying graph corresponding to region 1. We then show how to compute the set of points reachable starting from these points with respect to the next maximal strongly connected component, in this case s_3. In this example it turns out that the points reachable by moving along f_3 from points in the parallelogram $OABC$ is $OABC$ itself.

Now coming back to our original problem of finding if there is an execution of $\mathbb{H}$ starting at point O in location s_1 to some point in the shaded region in location s_5, we will build a rooted tree, called the *reachability tree*. Its nodes are labelled with pairs of locations and subedges and the root is labelled $(s_1, 0)$. The children of any node are labelled with the elements of the successors of the label of the current node with respect to every region it is adjacent to. The above computation is carried out with respect to every region to the left of the line $x = 2$. This gives us the set of all pairs of locations and points reachable on this line. Figure 5 shows some part of this tree.

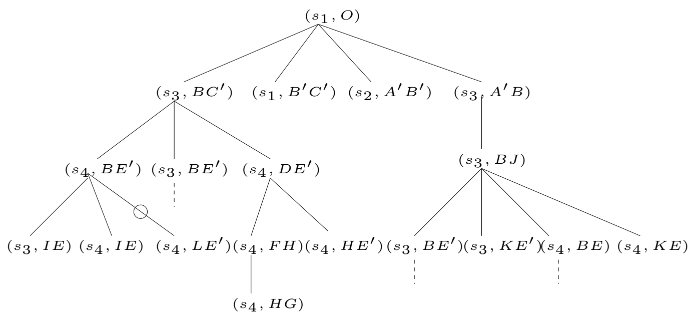

Fig. 5. Reachability tree

Our next goal is to show that this tree is finite. As a first step to achieve this, we prune some branches of the tree. The node (s_4, LE') is removed from the tree as its parent (s_4, BE') contains all the required information. The finiteness of the tree follows from two observations, namely, the number of children of any node is finite and every path in the tree is bounded. We can then apply Konig's Lemma to conclude that the tree is finite. To show that a path is finite, we have from the monotonicity of the flows that the leftmost point of any child of a node is to the right of the leftmost point of the node. For example, the x-coordinate of the left-most point of O which is O itself is less than that of A which is the leftmost point of $A'B$, which is in its successor. However, there is a priori no *minimum* distance by which this shift to the right occurs. Such a bound exists if the successor is with respect to a region which is a trapezium, like region 1. It is not clear for a "left-pinched triangle" like region 6. However for this case we argue that though a global minimum does not exist, given any path of the tree such a minimum exists. In case of a "right-pinched triangle" like region 2, even such a local minimum does not exist. Hence, instead, in this case we compute the "transitive closure" of the successor with respect to the region, which is the set of all points reachable on the boundary by moving within R and touching the boundary any number of times. We show that this is computable when the constraints corresponding to the boundary are strict. We then use the assumption that there are no adjacent right-pinched triangles, to argue that the paths of the tree are finite.

We cannot continue with the construction of the tree beyond the line $x = 2$, because all regions to the right of this line are unbounded. This might potentially lead to infinite paths in the tree. So we stop building the tree at the line l which passes through the leftmost vertex, and show that there is a finite bisimulation of the states corresponding to the regions to the right of this line. This bisimulation can be computed. Hence we can decide the reachability.

3 Preliminaries

3.1 Linear Hybrid Systems

A *linear hybrid system* (LHS) $\mathbb{H}$ is a tuple $(S, S_0, E, X, \mathit{flow}, \mathit{inv}, \mathit{guard})$ where

- S is a finite set of locations,
- $S_0 \subseteq S$ is the set of initial locations,
- $E \subseteq S \times S$ is the set of edges,
- $X = \{y_1, \cdots, y_n\}$ is a finite set of variables,
- $\mathit{flow} : S \rightarrow \mathbb{Q}^n$ associates a flow with every state,
- $\mathit{inv} : S \rightarrow$ *Guards* is a function associating an invariant with each state, and
- $\mathit{guard} : E \rightarrow$ *Guards* is a function associating a guard with each edge,

where *Guards* $= 2^{\mathbb{C}}$ and $\mathbb{C}$ is a finite subset of $\{\sum_{i=1}^{n} a_i y_i \sim b_i \mid a_i, b_i \in \mathbb{Q}, \sim \in \{<, >\}\}$. We call the elements of $\mathbb{C}$ which occur in the codomain of inv and guard, the set of *constraints* associated with $\mathbb{H}$. The size of X is called the *dimension* of $\mathbb{H}$.

We note that the definition of the hybrid system above deviates from the standard definition in that we do *not* allow *resets* and the constraints are restricted to be *strict*

We define the semantics of an LHS in terms of a *transition system*. The transition system of $\mathbb{H}$ is a triple $(X, X_0, \rightarrow)$, where $X = S \times \mathbb{R}^n$ is the set of states of $\mathbb{H}$, $X_0 \subseteq X$ called the set of initial states consists of state (s, v) such that $s \in S_0$ and $v \in \mathit{inv}(s)$, and the *transition relation* $\rightarrow$ is a binary relation on the set of *states* X. The transition relation $\rightarrow$ is defined as the union of *discrete* transitions $\rightarrow_d$ and *continuous* transitions $\rightarrow_c$, which are defined as:

- $(s, v) \rightarrow_d (s', v')$ if $v = v'$ and there exists $e = (s, s') \in E$ such that $v \in \mathit{inv}(s) \cap \mathit{inv}(s') \cap \mathit{guard}(e)$.
- $(s, v) \rightarrow_c (s', v')$ if $s = s'$ and there exists $t \in \mathbb{R}$ such that $t \geq 0$ and $v' = v + \mathit{flow}(s)t$, and for all $t' \in [0, t]$, $v + \mathit{flow}(s)t' \in \mathit{inv}(s)$.

An *execution* of $\mathbb{H}$ from a state (s_1, v_1) is a sequence of states $(s_1, v_1) \cdots (s_n, v_n)$ such that for all $1 \leq i < n$, $(s_i, v_i) \rightarrow (s_{i+1}, v_{i+1})$. We then say that (s_n, v_n) is *reachable* from (s_1, v_1), and denote it by $(s_1, v_1) \rightarrow^* (s_n, v_n)$. We can represent an execution $(s_1, v_1)(s_2, v_2) \cdots (s_n, v_n)$ as a function $\sigma : [0, t] \rightarrow S^+ \times \mathbb{R}^n$. We define σ as a pair of functions (σ^1, σ^2), where $\sigma^1 : [0, t] \rightarrow S^+$ gives the sequence of locations at any time point and $\sigma^2 : [0, t] \rightarrow \mathbb{R}^n$ gives the values of the variables. With each $(s_i, v_i) \rightarrow (s_{i+1}, v_{i+1})$ we associate a *delay* d_i, where $d_i = 0$ if $v_i = v_{i+1}$, and

$d_i = (v_{i+1} - v_i)/\mathit{flow}(s_i)$ otherwise. Let $t_i = \sum_{j=1}^{i} d_j$. We set $t = t_{n-1}$. We define $\sigma^1(t') = s_i$ if $t' \in (t_{i-1}, t_i)$, otherwise $\sigma^1(t') = s_i \cdots s_j$, where $t' = t_i$ and $t_{i-1} \neq t_i = t_{i+1} = \cdots = t_j \neq t_{j+1}$. We define $\sigma^2(t')$ for $t' \in [t_{i-1}, t_i]$ inductively. We set $\sigma^2(0) = v_1$ and $\sigma^2(t') = \sigma^2(t_{i-1}) + \mathit{flow}(s_i)(t' - t_{i-1})$ for $t' \in [t_{i-1}, t_i]$. A *run* of $\mathbb{H}$ is an execution starting from an initial state.

3.2 Elements of the two dimensional plane

We define some elements of the two dimensional plane formed by straight lines. A *convex closed polygonal set* P is the intersection of finitely many closed half-planes. We simply call P a convex polygon. The *interior* of P, denoted *interior*(P), is the intersection of finitely many open half-planes corresponding to the closed half-planes of P. The *boundary* of P, denoted *boundary*(P), is $P -$ *interior*(P). An *edge* of P is a maximal convex subset of *boundary*(P). We denote the set of all edges of P by *edges*(P). A *vertex* of P is a point of intersection of two distinct edges of P. The set of all vertices of P will be denoted by $vertices(P)$.

We call a convex subset of an edge, a *subedge*. The end-points of a subedge e are points a and b such that e consists of all points on the line segment joining a and b, except possibly a and b themselves. We denote this by *end-points*$(e) = \{a\} \cup \{b\}$. The subset of e without the end-points will be denoted *open*(e), which is $e -$ *end-points*(e). The elements of the subedge e are then its end-points which are contained in e and the *open*(e). This is denoted by *elements*$(e) = \{$*open*$(e)\} \cup \{a \mid a \in$ *end-points*$(e), a \in e\}$. From now on, by a convex set, we mean a polygon, interior of a polygon, or a subedge of a polygon.

3.3 Restricted hybrid systems

We call an LHS $\mathbb{H}$ *monotone* if there exists an $f \in \mathbb{R}^n$ such that for all locations s of $\mathbb{H}$, $\mathit{flow}(s).f > 0$, where . is the standard dot product. We call such an f a *direction* of $\mathbb{H}$.

We will call a linear hybrid system *planar*, if its dimension is two. A planar linear hybrid system is said to be *simple* if no three distinct lines corresponding to its constraints intersect at a common point, where the line corresponding to a constraint $\sum_{i=1}^{n} a_i y_i \sim b_i$ is the set of points satisfying $\sum_{i=1}^{n} a_i y_i = b_i$.

3.4 Notations for planar hybrid systems

Let us fix a simple monotone planar linear hybrid system $\mathbb{H} = (S, S_0, E, X, \mathit{flow}, inv, guard)$ for the rest of the paper. Let $X = \{x, y\}$ and $f_{\mathbb{H}}$ be a direction of $\mathbb{H}$. Let us fix our coordinate system such that the x-axis is parallel to $f_{\mathbb{H}}$ and the y-axis is perpendicular to it. Given a subedge e we define *left*(e) to be the infimum of the x-coordinates of the points in e and *right*(e) to be the supremum of the x-coordinates of the points in e.

Let V be the set consisting of the points of intersections of the lines corresponding to the constraints in $\mathbb{H}$. Let us associate with $\mathbb{H}$ a *set of lines* which are parallel to the

y-axis and contain some point in V. We denote this by *lines*$(\mathbb{H})$. We can order the lines of $\mathbb{H}$ as $l_1, l_2, \cdots, l_k$ such that for any $1 \leq i < j \leq k$, if v_i and v_j are the points in

V which are contained in l_i and l_j respectively, then *left*$(v_i) <$ *left*(v_j).

Let L be a set of lines which contains *lines*$(\mathbb{H})$ and the lines corresponding to the constraints in $\mathbb{H}$. We associate a set of *regions* with $\mathbb{H}$ which consists of polygons whose interiors are non-empty and which are formed by choosing exactly one closed half-plane corresponding to each line in L. We denote this by *regions*$(\mathbb{H})$. We use *regions*$(\mathbb{H}, i, j)$ to denote the regions of $\mathbb{H}$ which are contained in the set of points between lines l_i and l_j of *lines*$(\mathbb{H})$. Also *regions*$(\mathbb{H}, 0, j)$ and *regions*$(\mathbb{H}, i, k+1)$ denote the set of regions contained in the set of points which occur to the left of l_j and the set of points which occur to the right of l_i, respectively. Note that two distinct regions in *regions*$(\mathbb{H})$ have non-intersecting interiors, and the union of all the regions gives us the whole plane $\mathbb{R}^2$.

Following are a few observations about the regions of $\mathbb{H}$:

1. The regions in *regions*$(\mathbb{H}, 0, 1)$ are unbounded and have two or three edges.
2. The regions in *regions*$(\mathbb{H}, 1, k)$ are either triangles, or trapeziums, or unbounded regions with three edges. For the triangles, one of the edges is contained in some l_i and its vertex not on that edge is contained in either l_{i+1} or l_{i-1}. If the vertex is contained in l_{i+1}, then we call the triangle a *right-pinched triangle* otherwise we call it a *left-pinched triangle*. For the trapeziums in this region, we will call its edge a *parallel* edge if it lies on one of the l_i's.
3. The regions in *regions*$(\mathbb{H}, k, k+1)$ are unbounded with two or three edges.

From now on by a subedge we mean a subedge of the edge of some region in *regions*$(\mathbb{H})$. We abuse notation and call a pair (s, e) where $s \in S$ is a location and e a subedge, also a subedge. However it will be clear from the context which one we mean. The subedge (s, e) is said to contain the state (s, v) where $v \in e$. Two subedges (s, e) and (s', e') are said to be disjoint if the do not contain any common state. By a state (s, v) or a subedge (s, e) being on a subedge e' or a line l we mean v or e is contained in e' or l. Similarly we use regions also for pairs of states and regions.

We will focus on the following problems in the rest of the paper: the point-to-point reachability and the region-to-region reachability. The *point-to-point reachability problem* is to decide given two states (s_1, v_1) and (s_2, v_2), if $(s_1, v_1) \rightarrow^* (s_2, v_2)$. The *region-to-region* reachability problem is to decide given two location-region pairs (s_1, R_1) and (s_2, R_2), if there exist points $v_1 \in R_1$ and $v_2 \in R_2$ such that $(s_1, v_1) \rightarrow^* (s_2, v_2)$.

4 Decidability of the reachability problem

In this section we show that the point-to-point and region-to-region reachability problems for simple monotone planar linear hybrid systems is decidable. We will continue to use the notations introduced in the previous section. We first present a sketch of the proof of decidability.

1. We first show that the *edge-to-edge reachability* problem is decidable: given a subedge (s, e) of a region $R \in$ *regions*$(\mathbb{H}, 0, k)$, we can compute the set of all states on l_k which are reachable from the states on the subedge.

2. We then show that there exists a computable finite bisimulation of the transition system of $\mathbb{H}$ restricted to the states on and after l_k which respects the partition created by the elements of the regions in *regions*$(\mathbb{H}, k, k+1)$.
3. We then use the above results to decide the point-to-point and region-to-region reachability.

4.1 Edge-to-edge reachability

In this section we solve the problem of finding the set of all states on the line l_k reachable from a subedge (s, e) of some region $R \in$ *regions*$(\mathbb{H}, 0, k)$. Any execution from a state in (s, e) to a state on l_k can be broken up into a sequence of executions each of which is such that they move within a single region and leave or enter its boundary at most once. Our approach is to build a tree whose nodes represent subedges, and the states corresponding to the nodes of the children of a node give the set of all points reachable from the states in the parent node by executions which move within a region. Then any path in the tree would correspond to executions starting from states in the root. We call this the *reachability* tree. We show that the tree is computable and finite. Then the set of all states in the tree which correspond to the states on l_k will give us the required.

We first compute the set of all states reachable from a subedge by moving only within a region. We define an *almost-inside execution* with respect to a region to be an execution which leaves the boundary of the region at most once and enters the boundary of the region at most once, and at all times during the execution is in the region. An almost-inside execution (*AI*-execution) from a state (s, v) to a state (s', v') with respect to a region R is an execution $\sigma : [0, t]$ such that $\sigma^1(0)$ contains s and $\sigma^2(0) = v$, $\sigma^1(t)$ contains s' and $\sigma^2(t) = v'$, and there exist t_1, $t_2 \in [0, t]$ such that for all $t' \in (0, t_1] \cup [t_2, t)$, $\sigma^2(t') \in$ *boundary*(R), and for all $t' \in (t_1, t_2)$, $\sigma^2(t') \in$ *interior*(R). We say that a subedge (s', e') is reachable from a subedge (s, e) by almost-inside executions with respect to a region R, if for every $v' \in e'$, there exists a $v \in e$ and an *AI*-execution from (s, v) to (s', v'). The successor of a subedge (s, e) with respect to a region R is a subedge of R reachable from (s, e) by *AI*-executions with respect to R. We denote by *succ*$((s, e), R)$ the maximal successors of (s, e) with respect to R, where a successor (s', e') is maximal if for every successor (s', e''), $e'' \subseteq e'$.

In the next lemma, we show that *succ*$((s, e), R)$ is computable. A notion that we use is that of the underlying graph of the hybrid system restricted to those locations and edges whose invariants and guards respectively are satisfied by the elements of a region. Given a set of points V, we define the underlying graph of $\mathbb{H}$ with respect to V to be *graph*$(\mathbb{H}, V) = (V_{\mathbb{H}}, E_{\mathbb{H}},)$ such that $V_{\mathbb{H}} = \{s \in S \,|\, V \subseteq inv(s)\}$ and $E_{\mathbb{H}} = \{e \in E \,|\, V \subseteq guard(e)\}$.

Lemma 1. *Given a region $R \in$ regions$(\mathbb{H})$ and a subedge (s, e) of R, succ$((s, e), R)$ is computable.*

Proof We consider the maximal strongly connected components of the underlying graph *graph*$(\mathbb{H}$, *interior*$(R))$, and first compute the set of all states on the boundary reachable by moving in a single component. Then we show how this can be used to compute all the states reachable.

Given a graph G, let us call the graph with these strongly connected components as vertices, the component graph of G, and denote it as *SCC*(G). There is an edge between two vertices in *SCC*(G) if there is one between two states of the components in the original graph. Note that maximality of the components gives us that *SCC*(G) is a directed acyclic graph.

We observe that any *AI*-execution from a state in (s, e) to a state on the boundary of R would correspond to a path in *SCC*(G). For each such path $\pi = C_1 C_2 \cdots C_n$ where C_i's are the strongly connected components, we compute the states on the boundary of R reachable by *AI*-executions which follow this path. We do the computation iteratively. We first find the states reachable by moving only in the component C_1.

To compute the above, we need a notion of *post* of a convex subset of a region with respect to a set of flows, which is the set of all points in the region reachable by following the flows and always remaining in the interior of the region except possibly at the end-points. We can show that *post*(P, F, R) is computable, where P is a convex subset of region R and F is a set of at most two flows, and that it can be expressed as a finite union of convex subsets. A crucial observation is that corresponding to a trajectory following F from a point in P to any point in R, there is one which moves only within R. The only exceptions are the vertices of R, but it can be tested separately if they can be reached. It turns out that the set of all points in R reachable from a point in P are those in the cone generated by the flows in F. This can also be extended to any convex subset by taking the convex hull of the sets corresponding to the vertices of P. The details of the computation of *post*(P, F, R) are given in [9].

Now the set of all points reachable on the boundary by following flows in the component C_1 is given by *post*(e, F, R), where F contains those flows associated with the locations in C_1 which make a maximum or a minimum angle with $f_{\mathbb{H}}$. Further the points in *post*(e, F, R) which are in the interior of R can be reached in any location of C_1. A points p in *post*(e, F, R) which is on the boundary and is reached from some point in e by moving in R for some non-zero time, can only be reached if there is a location which is in both C_1 and *graph*$(\mathbb{H}, e)$, that is, there is an execution which can move into the interior, and there is a location which is in both C_1 and *graph*$(\mathbb{H}, p)$, that is, there is an execution which moves from the interior to the boundary. We then compute the set of all states on an edge reached by moving along the boundary from points on the boundary given by *post*(e, F, R). Suppose that we have found the set of all states on the interior and boundary reachable by the prefix of the path π till C_i. We can then compute the post of the interior points with respect to the flows of C_{i+1}, and compute the states reached when in C_{i+1} similar to above. Again the details can be found in [9].

Once we have found the set of states reachable along π, we can take the union of all the states over all the π's to get the set of all states on the boundary reachable. Since at each point in the procedure above we get a representation of the set of states on the boundary reachable as a finite union of subedges, and the number of paths π is finite, we can compute *succ*$((s, e), R)$. $\Box$

Now that we have shown that *succ*$((s, e), R)$ is computable, we can construct the reachability tree. However we also want to show that the tree is finite, and we will show this by ensuring that the paths in the tree are finite. We will do this by showing that along any path the successors move to the right by at least some minimum distance.

In the case of a right-pinched triangle such a minimum does not exist. Hence we will compute the transitive closure of *succ*, called *succ** where we consider points reachable by a sequence of *AI*-executions such that the last state of an execution is same as the first state of the next execution. The intuition behind this is that if we compute *succ** instead of *succ* for a subedge with respect to a region then we will not need to consider the *succ* of the elements in *succ** with respect to the region, as those states are already included in *succ**. We will see that the simplicity of the system can then be used to argue that the paths in the reachability tree are finite. Next lemma says that *succ** is computable.

Lemma 2. *Given a right-pinched triangle R in regions$(\mathbb{H}, 1, k)$ and a subedge (s, e) of R, succ$^*((s, e), R)$ is computable.*

Proof Let the right-pinched triangle R be abc with the edge ab on some l_i and c on l_{i+1} as shown in Figure 6. Let (s, e) be a subedge of ac. We first compute the set of all states

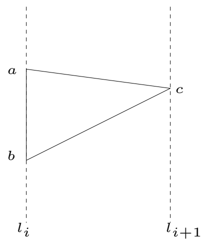

Fig. 6. Right-pinched triangle abc

on ac reachable by a sequence of one or more AI-executions. For this, we build a tree $T^*(s, e)$ rooted at node (s, e). We will need the following new notion of successor. Let us denote by *succ*$_1((s_1, e_1), R)$ the set of states reachable on ac by executions which touch bc at most once in the following sense: *succ*$_1((s_1, e_1), R) = \{(s_2, e_2) \mid (s_2, e_2) \in$ *succ*$((s_1, e_1), R), e_2 \subseteq ac\} \cup \{(s_3, e_3) \mid (s_3, e_3) \in$ *succ*$(s_2, e_2), e_3 \subseteq ac, (s_2, e_2) \in$ *succ*$((s_1, e_1), R), e_2 \subseteq bc\}$.

We now define how the tree is constructed. We will simultaneously mark nodes in the partial tree constructed. The children of a node (s_1, e_1) are the elements (s_2, e_2) in *succ*$_1((s_1, e_1), R)$ such that there is no node $(s_2, -)$ along the path from the root to the node (s_1, e_1). For every element (s_2, e_2) in *succ*$_1((s_1, e_1), R)$ such that there is a node (s_2, e'_2) along the path from the root to the node (s_1, e_1), we mark the node (s_2, e'_2). Note that a node could get marked twice. The construction of tree will terminate since it is finite, which is due to the fact that the number of children of any node is finite and the height of the tree is bounded by the number of locations.

We now describe how to compute *succ*$^*((s, e))$ from the tree constructed above. We form a set A which contains all the nodes of $T^*(s, e)$, and for each node (s_1, e_1) which belongs to a subtree of some marked node, it contains $(s_1, \mathit{full}(e_1))$, where *full*$(e_1)$ is the subedge e_2 of ac such that *left*$(e_2) =$ *left*(e_1) and *right*$(e_2) =$ *right*(c) and e_2 contains the points $\mathit{left}(e_1)$ and c if and only if e_1 contains them. A contains all points on ac' reachable from (s, e) by moving only within the triangle and touching the boundary any number of times. This is because if from a state (s, v_1) we can reach a state (s, v_2) by an execution σ, where v_2 is strictly to the right of v_1, then we can reach any point to the right of v_1 by taking a sequence of one or more executions whose transition sequence is same as that of σ but with possibly less time spent in each location. Similarly if (s_1, e_1) can reach (s_2, e_2), then $(s_1, \mathit{full}(e_1))$ can reach $(s_2, \mathit{full}(e_2))$. For details, see [9]. Hence it makes to sense to take the *full* of all nodes in the subtree of a marked node.

To compute the set of states on bc' reachable, we observe that such a state is reachable only from an AI-execution starting from some state on ac'. Hence the reachable states on bc' B can be computed by taking the *succ* of the maximal subedges of A. Finally, if c is reachable then it is reachable by an AI-execution starting from a state on ac' or bc', hence will be included in the $succ$ of the subedges in A or that of B. Hence all points in *succ*$^*((s, e), R)$ can be computed. □

We show below that the set of all states reachable on the line l_k is computable. As already said before, we construct a tree using the *succ* and *succ** to compute the children of the nodes. The nodes of the tree will correspond exactly to the states on edges of regions in *regions*$(\mathbb{H}, 0, k)$ reachable from some subedge of some region in it for which the tree is built.

Lemma 3. *Given a subedge (s^*, e^*) of a region in regions$(\mathbb{H}, 0, k)$, the set of all states on l_k reachable from some state on the subedge is computable.*

Proof Construction of the reachability tree *T*$_{\mathit{reach}}$$((s^*, e^*))$. We construct the reachability tree, in which the nodes correspond to subedges, and the children of a node capture the set of all states reachable from the states of the current node by *AI*-executions. A particular child of a node corresponds to *AI*-executions with respect to a single region.

We first define *tsucc* of a subedge with respect to a region which consists of states reachable by *AI*-executions in this region. We break up the subedges into its elements, because when computing *tsucc*, we require that all points of a subedge belong to the same set of regions. Note that otherwise, the end-point of a subedge which is a vertex could belong to a different set of regions than the subedge without the end-points.

For a subedge (s, e) of a region R, *tsucc*$((s, e, R))$ is given by:

– If R is not a right-pinched triangle, *tsucc*$((s, e, R)) = \{(s', el, R') \mid (s', e') \in$ *succ*$((s, e), R), el \in$ *elements*$(e'), el \subseteq R', R' \in$ *regions*$(\mathbb{H}, 0, k)\}$.
– If R is a right-pinched triangle, *tsucc*$((s, e, R)) = \{(s', el, R') \mid (s', e') \in$ *succ** $((s, e), R), el \in$ *elements*$(e'), el \subseteq R', R' \in$ *regions*$(\mathbb{H}, 0, k), R \neq R'\}$.

The root of $T_{\mathit{reach}}((s^*, e^*))$ is $*$. The children of $*$ are the element of the set $\{(s^*, e^*, R) \mid e^* \in R, R \in$ *regions*$(\mathbb{H}, 0, k)\}$. The children of any node (s, e, R) are the elements of *tsucc*$((s, e, R))$ which contain at least one state which has not occurred in the current node or any of its ancestors, that is, an element (s_1, e_1, R_1) is present in the *tsucc* of the current node (s, e, R) if for all nodes (s_1, e_2, R_1) which is the current node or its ancestor, there exists a v such that $v \in e_1 - e_2$.

We sketch below a proof of finiteness of the tree $T_{\mathit{reach}}((s^*, e^*))$. Details are given in [9]. First we make a few observations which are crucial in arguing the finiteness.

1. Let (s, e) and (s', e') be elements of subedges of a region R. Then if $(s', e') \in$ *tsucc*$((s, e), R)$, then *left*$(e) \leq$ *left*(e') and *right*$(e) \leq$ *right*(e'). This follows from the monotonicity of the flows in $\mathbb{H}$.
2. Given any region $R \in$ *regions*$(\mathbb{H}, 1, k)$, and (s, e) and (s', e') elements of subedges of R which are not on the l_i's such that $(s', e') \in$ *tsucc*$((s, e), R)$, we have:
   (a) If R is a trapezium or an unbounded region, then either *right*(e') is on some l_i or there exists a $d_R > 0$ such that *right*$(e') \geq$ *right*$(e) + d_R$.
   (b) If R is a left-pinched triangle, then either *right*(e') is on some l_i or there exists a d which increases monotonically with *right*(e) such that *right*$(e') \geq$ *right*$(e) +$ d.

Now turning to the proof, by construction the above tree is finitely branching. To see that every path in the tree is also finite, we can deduce from the above observations that (a) there is a bound on the number of consecutive children whose right-end points do not move closer to l_k (the bound is the number of locations), (b) when the successors are computed with respect to a trapezium and the right-end moves strictly to the right, there is a minimum distance by which the shift occurs namely the minimum of all the d_R's, (c) when the successors are computed with respect to a left-pinched triangle the minimum distance is non-zero and depends on the right-end point of the first occurrence on the path of one of its elements not contained in any l_i. This along with the simplicity of the system which guarantees that two right-pinched triangles are never adjacent to each other, we obtain a bound on the length of any path. Finally, from Konig's Lemma, we have that the tree is finite.

□

4.2 Finite bisimulation

We show that the states of $\mathbb{H}$ corresponding to the regions in *regions*$(\mathbb{H}, k, k+1)$ have a finite bisimulation. A binary relation $\sim$ over a set of states is a *bisimulation* if it is symmetric and for every pair of states (s_1, v_1) and (s_2, v_2), if $(s_1, v_1) \sim (s_2, v_2)$ and $(s_1, v_1) \rightarrow ({s'}_1, {v'}_1)$, then there exists a state $({s'}_2, {v'}_2)$ such that $(s_2, v_2) \rightarrow ({s'}_2, {v'}_2)$ and $({s'}_1, {v'}_1) \sim ({s'}_2, {v'}_2)$. We will show that there exists a computable equivalence relation $\sim$ of finite index on the set of states in *regions*$(\mathbb{H}, k, k+1)$ which is a bisimulation and which respects the partition created by the elements of the regions in *regions*$(\mathbb{H}, k, k+1)$. By partition created by l_k we mean the two parts, one consisting of the states on l_k and the other consisting of the rest of the states in *regions*$(\mathbb{H}, k, k+1)$.

We define $\sim$ as follows. $(s_1, v_1) \sim (s_2, v_2)$ if $s_1 = s_2$ and v_1, v_2 belong to the same element of a region. To see that this is a bisimulation consider (s, v_1) and (s, v_2) where v_1 and v_2 belong to the same element of some region. If (s, v_1) takes a discrete transition to (s', v_1), then so can (s, v_2) to (s', v_2) as the guards and invariants respect the elements of the regions. Suppose (s, v_1) takes a continuous transition to $(s, {v'}_1)$, then there is a straight line from the v_1 to ${v'}_1$ which passes through a finite sequence of infinite edges and interiors of the regions. There exists a straight line from v_2 parallel to the above which moves through the same sequence of edges and regions. Hence we can find a point ${v'}_2$ in the required region.

Since the number of regions in *regions*$(\mathbb{H}, k, k+1)$ is finite, the number of elements of these regions is also finite. Hence we have a finite bisimulation.

4.3 Point-to-point and region-to-region reachability

Theorem 1. *Point-to-point and region-to-region reachability problems are decidable for simple monotone linear hybrid systems.*

Proof To check if state (s', v') is reachable from (s, v), add two more lines to *lines*$(\mathbb{H})$ which pass through v and v', and are parallel to y-axis. Then check if (s', v') corresponds to any node in $T_{reach}((s, v))$.

To decide if (s', R') is reachable from (s, R), where $R, R' \in$ *regions*$(\mathbb{H})$, first compute the set of subedges *init*(R) of R reachable from points in R. For each subedge $(s^*, e^*) \in$ *init*(R), compute the set of subedges in l_k reachable, and then take their union. If $R' \in$ *regions*$(\mathbb{H}, k, k + 1)$, then construct the finite bisimulation to decide if R' is reachable. Otherwise check if any state in (s', R') is reachable from the set of subedges on its boundary reachable from states in *init*(R). □

5 Conclusions

In this paper we identified a new class of planar linear hybrid automata that have a decidable reachability problem. The key aspect in defining the class was requiring flows to be monotonic. One can prove that the reachability problem is undecidable in 4 dimensions; see [9] for details. The 3 dimensional case is an interesting open problem.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. Eugene Asarin, Oded Maler, and Amir Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1):35–65, 1995.
4. Eugene Asarin, Gerardo Schneider, and Sergio Yovine. Algorithmic analysis of polygonal hybrid systems, part I: Reachability. *Theor. Comput. Sci.*, 379(1-2):231–265, 2007.
5. Vincent D. Blondel, Olivier Bournez, Pascal Koiran, Christos H. Papadimitriou, and John N. Tsitsiklis. Deciding stability and mortality of piecewise affine dynamical systems. *Theoretical Computer Science*, 255(1–2):687–696, 2001.
6. Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Proc. 27th Annual ACM Symp. on Theory of Computing (STOC)*, pages 373–382, 1995.
7. R.M. Karp and R.E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.
8. G. Lafferriere, G. Pappas, and S. Sastry. O-minimal hybrid systems, 1998.
9. P. Prabhakar, V. Vladimerou, M. Viswanathan, and G. E. Dullerud. A Decidable Class of Planar Linear Hybrid Systems. Technical Report UIUCDCS-R-2008-2927, UIUC, January 2008.