AN ALGORITHM FOR THE APPROXIMATIVE ANALYSIS OF RECTANGULAR AUTOMATA

by

J. Preußig, S. Kowalewski, H. Wong-Toi and
T. A. Henzinger

Memorandum No. UCB/ERL M99/12

10 February 1999

COVER

AN ALGORITHM FOR THE APPROXIMATIVE ANALYSIS OF RECTANGULAR AUTOMATA

by

J. Preußig, S. Kowalewski, H. Wong-Toi and T. A. Henzinger

Memorandum No. UCB/ERL M99/12

10 February 1999

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

An Algorithm for the Approximative Analysis of Rectangular Automata*

J. Preußig[‡**], S. Kowalewski[‡*], H. Wong-Toi[¶], T.A. Henzinger[†***]

[‡] Dept. of Chemical Engineering, University of Dortmund, Germany.
[¶] Cadence Berkeley Labs, Berkeley, CA.
[†] Dept. of EECS, University of California, Berkeley, CA.

Abstract. Rectangular automata are well suited for approximate modeling of continuous–discrete systems. The exact analysis of these automata is feasible for small examples but can encounter severe numerical problems for even medium-sized systems. This paper presents an analysis algorithm that uses conservative overapproximation to avoid these numerical problems. The algorithm is demonstrated on a simple benchmark system consisting of two connected tanks.

1 Introduction

Embedded systems consist of digital controllers operating in analog environments. They are often used in safety-critical situations, such as for transportation control or logic control for processing systems, where correct behavior is essential. The interaction between the controller and its environment is hybrid, in that it involves both discrete and continuous aspects of behavior. Numerous models for specifying and analyzing such hybrid systems have been proposed: see, for example, the volumes [2,10].

This paper is concerned with hybrid systems verification using automated techniques. A system is modeled as a finite control graph augmented with real-valued variables subject to differential inclusions [7]. This model is far too expressive for immediate automatic verification. Previously, we proposed the following approach: given a complex hybrid system H, we construct a simpler abstraction H', and then automatically analyze H' instead [7]. We require the abstraction H' to be conservative, *i.e.* if H' is correct, then H is correct. Furthermore, the

* An abbreviated version of this paper appeared in the *Proceedings of the Fifth International Symposium on Formal Techniques in Real-time and Fault-tolerant Systems* (FTRTFT), Lecture Notes in Computer Science 1486, Springer-Verlag, 1998, pp. 228–240.

** Supported by the German Research Council (DFG) under grant Ko1430/3 in the special program KONDISK ('Continuous-discrete dynamics of technical systems').

*** Supported in part by the ONR YIP award N00014-95-1-0520, by the NSF CAREER award CCR-9501708, by the NSF grant CCR-9504469, by the DARPA/NASA grant NAG2-1214, by the ARO MURI grant DAAH-04-96-1-0341, and by the SRC contract 97-DC-324.041.

abstraction must be amenable to automated analysis. Typically, this is achieved by ensuring that the abstraction belongs to a restricted class of automata.

In previous work, we proposed that the abstractions H' be *linear hybrid automata* [1], in which the continuous dynamics are given as constant polyhedral differential inclusions. For these automata, reachable state sets can be represented exactly using linear constraints, and generated automatically using the model-checking tool HyTech [8]. Experience, however, shows that this subclass of automata may be still too expressive, and hence too computationally expensive to analyze efficiently and robustly [11, 12]. HyTech manipulates linear constraints over n variables as sets of n-dimensional polyhedra using a polyhedral library that performs exact computation over rationals stored using integers for numerator and denominator [6]. However the geometric algorithms over polyhedra require internal computations that quickly surpass the limits of integer precision when presented with systems with nontrivial continuous dynamics.

The main contribution of this paper is a numerically robust, specialized algorithm for computing (a supserset of) the reachable states of a subclass of linear hybrid automata. The algorithm operates on *rectangular automata* [9], subject to a few additional restrictions. In rectangular automata, the continuous dynamics must be defined using constant lower and upper bounds on the individual rates of variables. Given an arbitrary hybrid system, it is often easy to obtain a rectangular automaton as a conservative abstraction by bounding the rates of each variable. The key to efficient analysis is to avoid HyTech's computation over arbitrary polyhedra. Rather, sets of hybrid states are represented using *face-regions*—convex hulls of sets of $2n$ rectangular faces of dimension $n-1$. By restricting attention to rectangular faces, in which the continuous variables are decoupled, one obtains simpler algorithms for computing sets of successors states. The idea behind the algorithm is to determine, from a given reachable face, the minimal and maximal times it may take to reach some extremal bound, and to use this range of times to compute bounds on the new values of the variables. The algorithm is exact for systems with two continuous variables. It is inaccurate for higher dimensional systems, where it returns a superset of the reachable states. Consider the example of a three-dimensional system with x having rate $\dot{x} \in [1, 2]$, and both y and z having rate 1. The states that are reachable from the origin and for which $x \leq 2$ are $0 \leq x \leq 2 \wedge x/2 \leq y = z \leq x$. But this set is not definable using a face-region, and must be overapproximated by one. Thus, the drawback of the algorithm is its inaccuracy: it leads to a further approximation beyond that already inherent in translating the original hybrid system into a rectangular automaton, c.f. following an approximation step to a linear hybrid automaton, HyTech's algorithms over arbitrary polyhedra would return, if successful, the exact reachable set of that linear hybrid automaton.

The benefits of our algorithm are two-fold: efficiency and robustness. The simple structure of face-regions yields efficient algorithms for finding successor states. Furthermore, the simplicity of manipulating rectangles versus arbitrary polyhedra means that our algorithm can be made robust far more easily than

HyTech's algorithms for analyzing linear hybrid automata: rectangular faces can be conservatively approximated using rounding in the appropriate direction.

In the next section, we review basic definitions and concepts related to rectangular automata, and then introduce our algorithm in Section 3. In Section 4, we demonstrate the algorithm on a simple two-tank example, and Section 5 contains conclusions and a short discussion of related work.

2 Rectangular Automata

We review basic definitions and concepts related to rectangular automata [9].

2.1 Syntax

Let $Y = \{y_1, \ldots, y_k\}$ be a set of variables. A *rectangular inequality* over the set Y is an inequality of the form $y_i \sim c$, for some $y_i \in Y$, some relation $\sim \in \{\leq, =, \geq\}$[1] and some rational $c \in \mathbb{Q}$. A *rectangular predicate* over Y is a conjunction of rectangular inequalities over Y. The set of rectangular predicates over Y is denoted $\mathcal{R}(Y)$. We adopt the convention that for a boldfaced vector $\mathbf{a} \in \mathbb{R}^k$, its ith component is denoted by a_i. For a rectangular predicate φ over Y, let $[\![\varphi]\!]$ denote the set of points $\mathbf{a} \in \mathbb{R}^k$ for which φ is true when each y_i is replaced by a_i for each i. A set B is a *rectangle* if there exists a rectangular predicate φ such that $B = [\![\varphi]\!]$. Given a rectangle B, let $b_i^{min}, b_i^{max} \in \mathbb{Q} \cup \{-\infty, \infty\}$ denote the bounds on the individual variables such that $B = \Pi_{i=1..k}[b_i^{min}, b_i^{max}]$.

A *rectangular automaton* A is a system (X, V, *inv*, *flow*, *init*, E, *guard*, *reset_vars*, *reset*) consisting of the following components [9]:

Variables: A finite set $X = \{x_1, \ldots, x_n\}$ of variables.

Control modes: A finite set V of control modes.

Invariant conditions: A function *inv* that maps every control mode to an invariant condition in $\mathcal{R}(X)$. Control of the automaton may remain in a control mode only when its invariant is satisfied.

Flow conditions: A function *flow* that maps every control mode to a flow condition in $\mathcal{R}(\dot{X})$, where $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ with $\dot{x}_i$ representing the first derivative of x_i with respect to time. While control remains in a given mode, the variables evolve according to the differential inclusion specified by the mode's flow condition.

Initial conditions: A function *init* that maps every control mode to an initial condition in $\mathcal{R}(X)$.

Control switches: A finite multiset E of control switches in $V \times V$. For a control switch (v, v'), we say that v denotes the *source mode* and v' the *target mode*.

Guard conditions: A function *guard* that maps every control switch to a guard condition in $\mathcal{R}(X)$. Intuitively, the guard must be satisfied before the mode switch can be taken.

[1] For simplicity, we consider only nonstrict inequalities.

Resets: A function *reset_vars* that maps every control switch to an *update set* in 2^X, and a function *reset* that maps every control switch e to a reset condition in $\mathcal{R}(X)$. We require that for every control switch e and for every $x \in X$, if $x \in \mathit{reset\_vars}(e)$, then $\mathit{reset}(e)$ implies $x = c$ for some constant c, which, by abuse of notation, will be denoted $\mathit{reset}(e)(x)$. Intuitively, after the mode switch, the variables must satisfy the reset condition. Variables that appear in the update set must be reset to the fixed value indicated by the reset condition. Furthermore, all other variables must be unchanged.

2.2 Semantics

A *configuration* of a rectangular automaton is a pair $(v, \mathbf{a})$ that consists of a control mode v, together with a point $\mathbf{a} \in \mathbb{R}^n$. A configuration $(v, \mathbf{a})$ is *admissible* if $\mathbf{a} \in [\![\mathit{inv}(v)]\!]$. Let $\mathit{Config}(A)$ denote the admissible configurations of A. A configuration $(v, \mathbf{a})$ is *initial* if it is admissible and $\mathbf{a} \in [\![\mathit{init}(v)]\!]$. Let $\mathit{Init}(A)$ denote the set of initial configurations of A. There are two kinds of transitions between admissible configurations: jump transitions, which correspond to instantaneous control switches, and flow transitions, which correspond to the variables continuously evolving while time elapses. For every control switch $e \in E$, we define the binary *jump transition* relation $\xrightarrow{e}$ by $(v, \mathbf{a}) \xrightarrow{e} (v', \mathbf{a}')$ iff the following four conditions hold: (1) $e = (v, v')$, (2) $\mathbf{a} \in [\![\mathit{guard}(e)]\!]$, (3) $\mathbf{a}' \in [\![\mathit{reset}(e)]\!]$, and (4) for every $x_i \in X \setminus \mathit{reset\_vars}(e)$, $a_i = a_i'$. For every nonnegative real $\delta \geq 0$ and for every rectangular flow condition $\varphi \in \mathcal{R}(\dot{X})$, we define the binary flow relation $\xrightarrow{\delta}_{\varphi} \subseteq \mathbb{R}^n \times \mathbb{R}^n$ over states in $\mathbb{R}^n$ by $\mathbf{a} \xrightarrow{\delta}_{\varphi} \mathbf{a}'$ iff either (a) $\delta = 0$ and $\mathbf{a} = \mathbf{a}'$ or (b) $\delta > 0$ and $(\mathbf{a}' - \mathbf{a})/\delta \in [\![\varphi]\!]$. From this, we derive a relation expressing the flows of the automaton. For every nonnegative real $\delta \geq 0$, we define the binary *flow transition* relation $\xrightarrow{\delta} \subseteq \mathit{Config}(A) \times \mathit{Config}(A)$ by $(v, \mathbf{a}) \xrightarrow{\delta} (v', \mathbf{a}')$ iff (1) $v = v'$, and (2) $\mathbf{a} \xrightarrow{\delta}_{\mathit{flow}(v)} \mathbf{a}'$. The second condition states that the continuous variables evolve at a rate consistent with the flow condition.

The transition relation $\rightarrow_A$ for the rectangular automaton A is $\bigcup_{e \in E} \xrightarrow{e} \cup \bigcup_{\delta \in \mathbb{R}_{\geq 0}} \xrightarrow{\delta}$. A *trajectory* is a finite sequence $q_0, q_1, \ldots, q_m$ of admissible configurations such that q_0 is an initial configuration, and $q_i \rightarrow_A q_{i+1}$ for $i = 0..m-1$. A configuration is *reachable* if it appears on some trajectory. The set of reachable configurations of A is denoted $\mathit{Reach}(A)$.

2.3 Reachability analysis

Reachability analysis consists of determining the set of reachable configurations. It is commonly used to validate the correctness of a system. Safety properties intuitively assert that nothing bad happens during system execution. Violations of safety properties can be expressed via a designated set of unsafe configurations. Safety properties can then be verified by performing reachability analysis and checking whether any unsafe configurations are reached. We review a familiar procedure for computing $\mathit{Reach}(A)$. We define the successor operator

$Post : 2^{Config(A)} \to 2^{Config(A)}$ by $Post(W) = Post_{time}(W) \cup Post_{evt}(W)$, where $Post_{time}(W) = \{q' \mid \exists q \in W. \exists \delta \in \mathbb{R}_{\geq 0}. q \xrightarrow{\delta}_A q'\}$ and $Post_{evt}(W) = \{q' \mid \exists q \in W. \exists e \in E. q \xrightarrow{e}_A q'\}$. Let $Post^i$ represent the composition of i $Post$ operations. Then $Reach(A) = \bigcup_{i=0}^{\infty} Post^i(Init(A))$. Iterating $Post$ until convergence yields a semialgorithm for computing $Reach(A)$. If the procedure terminates, one obtains $Reach(A)$, but in general, termination is not guaranteed.

2.4 Simple rectangular automata

A rectangular predicate φ is *bounded* if $[\![\varphi]\!]$ is bounded. A rectangular automaton is *simple* if the following two properties hold:

1. Its invariant, initial, flow, guard, and reset conditions are bounded.
2. For every control switch $e = (v, v')$, if $[\![inv(v)]\!] = B = \Pi_{i=1..n}[b_i^{min}, b_i^{max}]$, and if $[\![inv(v')]\!] = B' = \Pi_{i=1..n}[b_i^{min'}, b_i^{max'}]$, then there exists a variable $x_i \in X$ such that either
   (a) (i) $x_i \in reset\_vars(e)$, and (ii) the guard condition $guard(e)$ implies $x_i = b_i^{min}$ or $x_i = b_i^{max}$, and (iii) $reset(e)(x_i) \in \{b_i^{min'}, b_i^{max'}\}$,
   (b) (i) $x_i \notin reset\_vars(e)$, and (ii) the guard condition $guard(e)$ implies $x_i = b_i^{min}$, and (iii) $b_i^{min} \in \{b_i^{min'}, b_i^{max'}\}$, or
   (c) (i) $x_i \notin reset\_vars(e)$, and (ii) the guard condition $guard(e)$ implies $x_i = b_i^{max}$, and (iii) $b_i^{max} \in \{b_i^{min'}, b_i^{max'}\}$.

The second condition states that guard conditions include tests for equality for one of the variables' bounding values in the source mode's invariant. Furthermore, if the variable is reset, then it is reset to a bounding value for the target mode's invariant (condition 2a). Finally, if it is not reset, then its value in the guard must be a bounding value in the target mode's invariant (conditions 2b and 2c).

Simple rectangular automata often arise naturally when approximating more complex hybrid systems. In order to conservatively overapproximate the flow field of nontrivial continuous dynamics, one may partition the state space into rectangular blocks, and for each variable provide constant lower and upper bounds on the flow within each block [7]. A control mode is split into several control modes, one for each block of the partition. Crossing from one block in the state space to another is modeled by mode switches among the blocks, with the guards being tests for equality across common boundaries. For example, a mode v with the invariant $1 \leq x \leq 3$ may be split into two modes — v_1 with the invariant $1 \leq x \leq 2$ and v_2 with the invariant $2 \leq x \leq 3$ — with mode switches between them having the guard $x = 2$.

3 Approximative Analysis Algorithm

In this section we define an algorithm for the approximative reachability analysis of simple rectangular automata. The algorithm is built on top of a conservative $\widehat{Post}$ operator and based on some other concepts that are defined first.

3.1 Face-regions

A *face* is a rectangular predicate with one dimension fixed to a certain value. Our rationale for introducing faces is to use rectangular faces to represent non-rectangular sets. A *face-region* $\mathcal{F}$ is a set $\{F_1, \ldots, F_k\}$ where each F_i is a face. The semantics of $\mathcal{F}$ is the convex hull over its k faces, i.e. $[\![\mathcal{F}]\!] = \mathit{convexhull}\{[\![F_1]\!], \ldots, [\![F_k]\!]\}$. This is shown for an example in Fig.1 where a face-region $\mathcal{F}$ is represented by the two faces F_1 and F_2. In practice, the faces of a face-region over n variables are derived from $2n$ constraints of the form $x_i = l_i$ or $x_i = u_i$. In the example, the face F_1 corresponds to $x_1 = 1$ and the face F_2 to $x_2 = 7$, with the empty faces for $x_1 = 7$ and $x_2 = 1$ being omitted.

3.2 The operator $Post^{x_d=b}$

Our algorithm is based on the fact that face-regions are bounded by their faces. We present an operator $Post^{x_d=b}$ that takes a face F and a flow φ of the variables and returns a conservative approximation $Post^{x_d=b}(F, \varphi)$ of the states for which $x_d = b$ that are reachable from F by following φ.

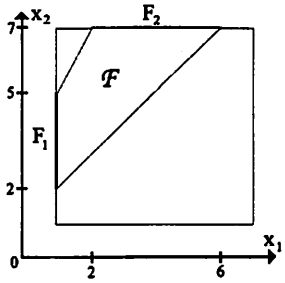


Fig. 1. Face-Region

Let F be the face $\bigwedge_{i=1}^{n}(x_i^{min} \leq x_i \leq x_i^{max})$ and let φ be the flow condition in $\mathcal{R}(\dot{X})$ defining the possible rates of the x_i as $\bigwedge_{i=1}^{n}(\dot{x}_i^{min} \leq \dot{x}_i \leq \dot{x}_i^{max})$. We want to compute a new face $\bigwedge_{i=1}^{n}(x_i^{min'} \leq x_i \leq x_i^{max'})$ that defines the reachable states where $x_d^{min'} = x_d^{max'} = b$. The idea behind the operator is simple. We look at the time that x_d needs at least and at most to reach the bound b from its possible values in F and then see how much the values of the other dimension's variables possibly change in these times, thereby computing the new values for x_i^{min} and x_i^{max}.

So at first we compute an interval $T = [t_{min}, t_{max}]$ of the times in which x_d can reach the bound b according to its current value and its possible rates. The maximal time of the interval can also be set to the symbol '*infinite*' to express that a bound can be reached within any arbitrarily large amount of time. The interval is empty if the bound b can not be reached at all. This is the case if b is less than x_d^{min} and no negative rate is possible or if b is greater than x_d^{max} and no positive rate is possible. Formally:

$$T = \emptyset \quad \text{if} \quad (b < x_d^{min} \wedge \dot{x}_d^{min} \geq 0) \vee (b > x_d^{max} \wedge \dot{x}_d^{max} \leq 0)$$

If T is not empty its interval bounds t_{min} and t_{max} are given as follows.

$$t_{min} = \begin{cases} \frac{x_d^{min} - b}{-\dot{x}_d^{min}} & \text{if } b < x_d^{min} \wedge \dot{x}_d^{min} < 0 \\ \frac{b - x_d^{max}}{\dot{x}_d^{max}} & \text{if } b > x_d^{max} \wedge \dot{x}_d^{max} > 0 \\ 0 & \text{if } x_d^{min} \leq b \leq x_d^{max} \end{cases}$$

$$t_{max} = \begin{cases} \textit{infinite} & \text{if } \begin{array}{l}(b < x_d^{min} \wedge \dot{x}_d^{min} < 0 \wedge \dot{x}_d^{max} \geq 0) \\ \vee(b > x_d^{max} \wedge \dot{x}_d^{max} > 0 \wedge \dot{x}_d^{min} \leq 0) \\ \vee(x_d^{min} \leq b \leq x_d^{max} \wedge \dot{x}_d^{min} \leq 0 \leq \dot{x}_d^{max})\end{array} \\ \frac{x_d^{max} - b}{-\dot{x}_d^{max}} & \text{if } (b \leq x_d^{max} \wedge \dot{x}_d^{max} < 0) \\ \frac{b - x_d^{min}}{\dot{x}_d^{min}} & \text{if } (b \geq x_d^{min} \wedge \dot{x}_d^{min} > 0). \end{cases}$$

We now give an intuitive explanation of these formulas. To compute T, we first check on which side of the current values of x_d the new bound b lies. Three cases are possible:

1. $b < x_d^{min}$: If no negative rate is possible, i.e. $\dot{x}_d^{min} \geq 0$, then T is empty. So consider the case that a negative rate is possible, i.e. $\dot{x}_d^{min} < 0$. The maximal time is either '*infinite*', if a positive or zero rate is also possible, or else computed by using the slowest rate possible to clear the greatest possible distance to b. The minimal time is computed by using the fastest rate possible to clear the smallest possible distance to b.
2. $b > x_d^{max}$: If no positive rate is possible, i.e. $\dot{x}_d^{max} \leq 0$, then T is empty. So consider the case that a positive rate is possible, i.e. $\dot{x}_d^{max} > 0$. The maximal time is either '*infinite*', if a negative or zero rate is also possible, or else computed by using the slowest rate possible to clear the greatest possible distance to b. Again, the minimal time is computed by using the fastest rate possible to clear the smallest possible distance to b.
3. $x_d^{min} \leq b \leq x_d^{max}$: In this case the bound b is already part of the old face. So we have $t_{min} = 0$. The maximal time is '*infinite*', if a zero rate is possible. Otherwise there are only positive or only negative rates and the maximal time is computed using the slowest rate possible to clear the greatest possible distance to b.

We use the computed interval T to compute the new bounds $x_i^{min'}$ and $x_i^{max'}$ of the other dimensions' variables that give the new face where $x_d^{min'} = x_d^{max'} = b$. If T is empty, then no face can be computed and *Post* returns an empty face. Otherwise, let us first assume that the computation of t_{max} did not yield '*infinite*'. To compute $x_i^{min'}$ we take the old x_i^{min} and subtract as much as possible, if there are negative rates for x_i, or add as little as possible, if there are only positive rates. To compute $x_i^{max'}$ we take the old x_i^{max} and add as much as possible, if there are positive rates for x_i, or subtract as little as possible, if there are only negative rates. Then we obtain:

$$x_i^{min'} = \begin{cases} x_i^{min} + \dot{x}_i^{min} \cdot t_{max} & \text{if } i \neq d \wedge \dot{x}_i^{min} < 0 \\ x_i^{min} + \dot{x}_i^{min} \cdot t_{min} & \text{if } i \neq d \wedge \dot{x}_i^{min} \geq 0 \\ b & \text{if } i = d \end{cases}$$

$$x_i^{max'} = \begin{cases} x_i^{max} + \dot{x}_i^{max} \cdot t_{max} & \text{if } i \neq d \wedge \dot{x}_i^{max} > 0 \\ x_i^{max} + \dot{x}_i^{max} \cdot t_{min} & \text{if } i \neq d \wedge \dot{x}_i^{max} \leq 0 \\ b & \text{if } i = d \end{cases}$$

If t_{max} has previously been found to be '*infinite*', then the computations with t_{max} also yield $\pm$'*infinite*'. For the variable x_d, constant $b \in \mathbb{Q}$, face F, and flow condition φ, we define the new face as

$$Post^{x_d=b}(F, \varphi) = \bigwedge_{1 \leq i \leq n} x_i^{min'} \leq x_i \leq x_i^{max'}.$$

Considering the example in Fig.1, the operator could be used to compute the face F_2 from F_1. Let $F_1 = \{x_1 = 1 \wedge 2 \leq x_2 \leq 5\}$, $F_2 = \{2 \leq x_1 \leq 6 \wedge x_2 = 7\}$ and $\varphi = \{\dot{x}_1 = 1 \wedge 1 \leq \dot{x}_2 \leq 2\}$. Then $F_2 = Post^{x_2=7}(F_1, \varphi)$ with $T = [1, 5]$.

In two dimensions, the operator $Post^{x_d=b}$ is exact. For any number of dimensions, the operator is conservative.

Lemma 1. *Let x_d be a variable and b a constant in $\mathbb{Q}$. For every face F and flow condition φ, the operator $Post^{x_d=b}$ computes a superset of the points at which $x_d = b$ that are reachable according to φ from the points given by F, i.e.* $[\![Post^{x_d=b}(F, \varphi)]\!] \supseteq \{\mathbf{x}' | \exists \mathbf{x} \in [\![F]\!]. \exists \delta \in \mathbb{R}_{\geq 0}. \mathbf{x} \xrightarrow{\delta}_{\varphi} \mathbf{x}'\} \cap [\![x_d = b]\!]$.

Proof. We outline the proof. Assume $\mathbf{x}'$ satisfies $x_d = b$ and there exists an $\mathbf{x} \in [\![F]\!]$ and $\delta \in \mathbb{R}_{\geq 0}$ such that $\mathbf{x} \xrightarrow{\delta}_{\varphi} \mathbf{x}'$. By a tedious check on the construction of interval T in the definition of the $Post^{x_d=b}$ operator, we see that $\delta \in T$, and hence by construction of the bounds $x_i^{min'}$ and $x_i^{max'}$, each x_i lies in $[x_i^{min'}, x_i^{max'}]$.

3.3 An algorithm for approximative reachability analysis

The algorithm described here makes use of the fact that the invariants in a control mode of a rectangular automaton form a rectangular region. So, a reachable face-region within the invariants can be represented by faces that lie on the invariant's bounds. Let $\mathcal{F}^v$ be a reachable face-region in a control mode v. Now we want to compute the new $\mathcal{F}^k$ in another control mode k, for which there is a mode switch from v. The algorithm performs an event-step evolution from each face of $\mathcal{F}^v$, checking the guard and possibly applying a reset operation to compute each face F_i^k lying on one of the faces of the invariant of k. Then the algorithm does a time-step evolution from each F_i^k using the $Post^{x_d=b}$ operator for each of the bounds defining the invariant of k, thereby computing $\mathcal{F}^k$. Being able to compute a successor face-region for a face-region, we can apply standard fixpoint analysis to find the set of all reachable face-regions in a given automaton.

We now give a more formal description of the algorithm. Let the invariant in a control mode v be of the form $\bigwedge_{i=1}^{n}(x_i^{min} \leq x_i \leq x_i^{max})$. Let a *region* R be a set of pairs $(v, \mathcal{F})$ of a mode v and a face-region $\mathcal{F}$ in v. The semantics of a region

R is a set of configurations defined by $[\![R]\!] = \{(v, \mathrm{a}) | \exists (v, \mathcal{F}) \in R.\mathrm{a} \in [\![\mathcal{F}]\!]\}$. We define $\widehat{Post}(R)$ as follows:

$$\widehat{Post}(R) = \bigcup_{(v,\mathcal{F})\in R} \widehat{Post}_{time}(v, \mathcal{F}) \cup \widehat{Post}_{evt}(v, \mathcal{F})$$

where

$$\widehat{Post}_{time}(v, \mathcal{F}) = \{(v, \bigcup_{F\in\mathcal{F}} \begin{pmatrix} \bigcup_{1\le i\le n}\{[\![Post^{x_i=x_i^{min}}(F, flow(v))]\!] \cap [\![inv(v)]\!]\} \cup \\ \bigcup_{1\le i\le n}\{[\![Post^{x_i=x_i^{max}}(F, flow(v))]\!] \cap [\![inv(v)]\!]\} \end{pmatrix})\}$$

and

$$\widehat{Post}_{evt}(v, \mathcal{F}) = \bigcup_{e=(v,k)\in E} \bigcup_{F\in\mathcal{F}} \{(k, \left\{ \left[\!\!\left[\begin{array}{l} (\exists x_i \in reset\_vars(e).(guard(e) \wedge F)) \\ \wedge\ reset(e) \wedge inv(k) \end{array} \right]\!\!\right] \right\})\}.$$

The way in which the operator $\widehat{Post}_{time}$ is built on top of the conservative operator for faces insures that it conservatively overapproximates $Post_{time}$.

Lemma 2. *For all regions R, $[\![\widehat{Post}_{time}(R)]\!] \supseteq Post_{time}([\![R]\!])$.*

The operator $\widehat{Post}_{evt}$ is exact, since it simply requires existential quantification and intersection of faces with guards, reset conditions, invariants, and hyperplanes of the form $x = b$.

Lemma 3. *For all regions R, $[\![\widehat{Post}_{evt}(R)]\!] = Post_{evt}([\![R]\!])$.*

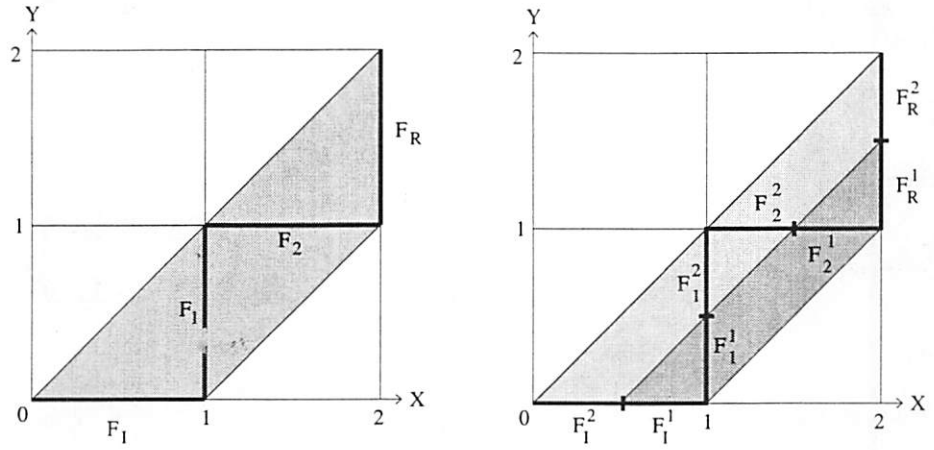
The operator $\widehat{Post}$ can be used in the procedure in Section 2.3 in place of the exact operator $Post$. Since $\widehat{Post}(R)$ contains $Post(R)$, the resultant fixpoint contains the set of reachable configurations.

Theorem 1. *Let A be a rectangular automaton, and let the region R_{init} represent the initial configurations of A,* i.e. *$[\![R_{init}]\!]$=Init(A). Then the fixpoint computed by $\widehat{Post}$ from R_{init} is a superset of reachable configurations of A, i.e.*

$$[\![\bigcup_{0\le i\le\infty} \widehat{Post}^i(R_{init})]\!] \supseteq \bigcup_{0\le i\le\infty} Post^i(Init(A)) = Reach(A).$$

3.4 Tighter approximations

The operator $\widehat{Post}$ is exact in two dimensions. In higher dimensions, however, it can compute vast overapproximations. Consider a box $[0,2] \times [0,2] \times [0,4]$ in the three dimensions x, y, and t. Let $F_I = \{0 \le x \le 1 \wedge y = t = 0\}$ and $\varphi = \{\dot{x} = \dot{y} = \dot{t} = 1\}$. Computing $F_R = Post^{x=2}(F_I, \varphi)$ results in $F_R = \{x = 2 \wedge 1 \le y \le 2 \wedge 1 \le t \le 2\}$. Now we divide the original box according to the hyperplanes $x = 1$ and $y = 1$. Fig.2a displays the partitioned box projected on the plane spanned by x and y. From an initial face F_I the algorithm would now compute via F_1 and F_2 the face $F_R = Post^{x=2}(Post^{y=1}(Post^{x=1}(F_I, \varphi), \varphi), \varphi)$. This F_R gives a vast overapproximation for the value of t. The faces are:



(a) Computation without splitting

(b) Computation using face splitting

Fig. 2. Face splitting

F_I: $(0 \leq x \leq 1, y = t = 0)$ $\quad$ F_1: $(x = 1, 0 \leq y \leq 1, 0 \leq t \leq 1)$
F_2: $(1 \leq x \leq 2, y = 1, 0 \leq t \leq 2)$ $\quad$ F_R: $(x = 2, 1 \leq y \leq 2, 0 \leq t \leq 3)$

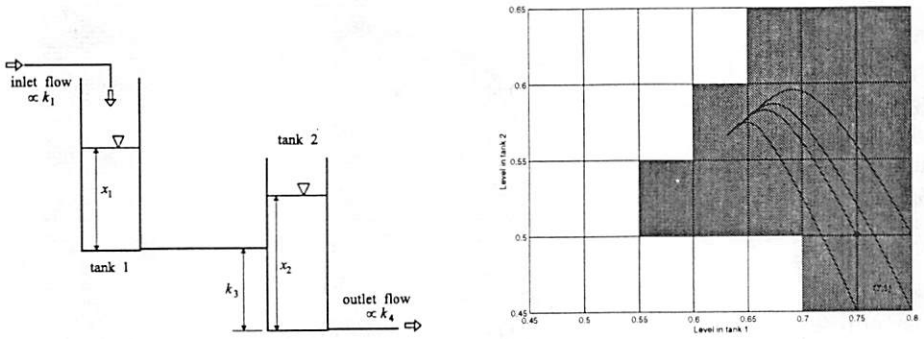
An attempt to solve this problem is *face splitting*. The idea of face splitting is, not to compute the successor faces from a whole face, but to split a face into several parts, and then to compute the successors for these parts. In Fig.2b the face F_I is split into two pieces F_I^1 and F_I^2. Assuming the same partitioning as before, this leads to a better approximation, namely the faces F_R^1 and F_R^2, instead of F_R. The faces are:

F_I^1: $(0.5 \leq x \leq 1, y = t = 0)$ $\quad$ F_I^2: $(0 \leq x \leq 0.5, y = t = 0)$
F_1^1: $(x = 1, 0 \leq y \leq 0.5, 0 \leq t \leq 0.5)$ $\quad$ F_1^2: $(x = 1, 0.5 \leq y \leq 1, 0.5 \leq t \leq 1)$
F_2^1: $(1.5 \leq x \leq 2, y = 1, 0.5 \leq t \leq 1.5)$ $\quad$ F_2^2: $(1 \leq x \leq 1.5, y = 1, 0.5 \leq t \leq 1.5)$
F_R^1: $(x = 2, 1 \leq y \leq 1.5, 0.5 \leq t \leq 2)$ $\quad$ F_R^2: $(x = 2, 1.5 \leq y \leq 2, 1 \leq t \leq 2.5)$

Note that the same result could not be achieved by simply choosing a finer grid. A finer grid may be useful to obtain rates that are better approximations for the underlying differential equations but as we have shown in this example, somewhat unintuitively, there are cases in which finer grids lead to worse overapproximations in the analysis procedure. However, the drawback of face splitting is that it contributes to state explosion in larger systems.

4 Two-Tank Example

The analysis method is demonstrated on a small laboratory plant [12]. First, the dynamics of the system are approximated by a rectangular automaton. Then this



(a) Scheme of the two-tank plant

(b) Reachability analysis for the two-tank example

Fig. 3. Two-Tank example

automaton is analyzed using the approximative analysis algorithm. The plant consists of two tanks which are connected as illustrated in Fig.3a. The first tank is fed by an inflow characterized by the parameter k_1. The inlet stream of the second tank is the outflow of the first one. The stream from tank 1 to tank 2 depends on the difference between the tanks' liquid levels, the geometric properties of the connecting pipe (characterized by parameter k_2) and its height k_3 above the bottom of tank 2. The liquid level in this tank and the dimensions of its outlet pipe (characterized by constant k_4) determine the flow out of tank 2. The resulting model for the dynamics of the two tank levels is given by equation (1), where $\mathbf{x} = (x_1, x_2)$ is the continuous state vector of the system. Equation (1) defines a flow $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, k_1, ..., k_4)$ which moves the system to an equilibrium point $\mathbf{x_s}$ for all $k_j > 0$, $x_i \geq 0$, and $x_1 \geq x_2 - k_3$.

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{cases} \begin{pmatrix} k_1 - k_2\sqrt{x_1 - x_2 + k_3} \\ k_2\sqrt{x_1 - x_2 + k_3} - k_4\sqrt{x_2} \end{pmatrix} & \text{if} \quad x_2 > k_3 \\ \begin{pmatrix} k_1 - k_2\sqrt{x_1} \\ k_2\sqrt{x_1} - k_4\sqrt{x_2} \end{pmatrix} & \text{if} \quad x_2 \leq k_3 \end{cases} \tag{1}$$

A MATLAB script was developed to generate an approximative rectangular automaton of the dynamics given by (1) for arbitrary partition grids. By refining the partition grid, arbitrarily accurate automata can be generated. The script estimates upper and lower bounds for the derivatives of x_1 and x_2 in each partition cell by evaluating equation (1) for a finite number of grid points. While the automata generated by this procedure are not conservative approximations, they are sufficiently accurate for testing the approximative analysis algorithm on realistic systems.

The MATLAB script calls HYTECH to analyze the rectangular automaton and then graphically displays HYTECH's output of the reachable state space. The finest grid we were able to run on the exact version of HYTECH can be seen in Fig.3b. Using our approximative algorithm for the analysis more exact models with much finer grids could be analyzed, as is shown in Fig.4.

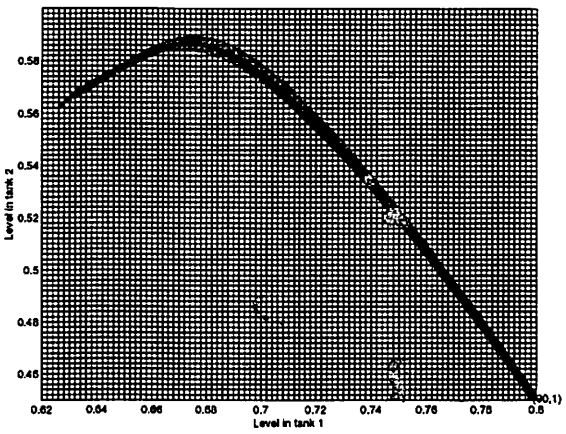


Fig. 4. Reachability analysis for the two-tank example on a finer grid

The main purpose of our example was the demonstration of the approximation technique. From the analysis point of view, the example is not very challenging, since simulation can achieve more accurate results rather easily. The application of the algorithm to a more realistic example can be found in [13].

5 Conclusion and Related Work

We have introduced an algorithm for the approximative analysis of rectangular hybrid systems. Though the approximations are accurate in two dimensions, the approximation technique seems to be of limited use in higher dimensions.

The idea of computing reachable regions only from the bounds of the predecessor regions, instead of looking at the whole regions, is not new. It also appears in [4,5], where the bounding edges of a region are 'bloated' outward with an integration routine on the original differential equations of a system to compute the edges of the successor region. Reachable sets may be represented by a set of non-convex polyhedral 2-dimensional projections. The idea of 'bloating' is referred to as 'face lifting' in [3]. The state space is partitioned a priori into hypercubes, and a face is then 'lifted' to a successor region, by computing successors for all hypercubes associated with that face. The current methodology uses a fixed partition which can lead to computational inefficiency. The authors have also identified the problem of potentially large overapproximations during the analysis.

There are a number of extensions that could be made to our algorithm. Instead of allowing only rectangular bounds for the derivatives of continuous variables the $\widehat{Post}$ operator could also deal with differential equations. In a different direction, the algorithm could be made accurate over higher dimensions at the cost of increased computation. Each face could be represented as the convex hull of a set of points instead of a rectangle. Then we could compute

the successor faces F^i of a given face F^k pointwise, i.e. to compute F^i at $x_d = b$, we collect from each point in F^k the successor points for which $x_d = b$. This would avoid unwanted overapproximation during the analysis. However, from a computational standpoint, we confront two problems. First, we compute redundant points, and have to minimize face representations (i.e. solve the convex hull problem). Second, intersection with face invariants is no longer as efficient.

Acknowledgement. We thank Tiziano Villa for many helpful comments and Olaf Stursberg for his help with the MATLAB script.

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*, 138:3–34, 1995.
2. P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors. *Hybrid Systems IV.* Lecture Notes in Computer Science 1273. Springer-Verlag, 1997.
3. T. Dang and O. Maler. Reachability analysis via face lifting. In T.A. Henzinger and S. Sastry, editors, *HSCC 98: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1386, pages 96–109. Springer-Verlag, 1998.
4. M.R. Greenstreet. Verifying safety properties of differential equations. In R. Alur and T.A. Henzinger, editors, *CAV 96: Computer Aided Verification*, Lecture Notes in Computer Science 1102, pages 277–287. Springer-Verlag, 1996.
5. M.R. Greenstreet and I. Mitchell. Integrating projections. In T.A. Henzinger and S. Sastry, editors, *HSCC 98: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1386, pages 159–174. Springer-Verlag, 1998.
6. N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximation. In B. LeCharlier, editor, *SAS 94: Static Analysis Symposium*, Lecture Notes in Computer Science 864, pages 223–237. Springer-Verlag, 1994.
7. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):540–554, 1998.
8. T.A. Henzinger, P.H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1,2):110–122, 1997.
9. T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995. Full version to appear in *Journal of Computer and System Sciences.*
10. T.A. Henzinger and S. Sastry, editors. *HSCC 98: Hybrid Systems—Computation and Control.* Lecture Notes in Computer Science 1386. Springer-Verlag, 1998.
11. T. Stauner, O. Müller, and M. Fuchs. Using HYTECH to verify an automotive control system. In O. Maler, editor, *HART 97: Hybrid and Real-Time Systems*, Lecture Notes in Computer Science 1201, pages 139–153. Springer-Verlag, 1997.
12. O. Stursberg, S. Kowalewski, I. Hoffmann, and J. Preußig. Comparing timed and hybrid automata as approximations of continuous systems. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems IV*, Lecture Notes in Computer Science 1273, pages 361–377. Springer-Verlag, 1996.
13. T. Villa, H. Wong-Toi, A. Balluchi, J. Preußig, A. Sangiovanni-Vincentelli, and Y. Watanabe. Formal verification of an automotive engine controller in cutoff mode. 1998. Submitted.