Verification of linear hybrid systems by means of convex approximations*†

Nicolas Halbwachs, Pascal Raymond
Verimag‡
Montbonnot, France

Yann-Eric Proy
Schneider-Electric
Grenoble, France

Abstract

We present a new application of the abstract interpretation by means of convex polyhedra, to a class of hybrid systems, i.e., systems involving both discrete and continuous variables. The result is an efficient automatic tool for approximate, but conservative, verification of reachability properties of these systems.

1 Introduction

Timed automata [AD90] have been recently introduced to model real-time systems. A timed automaton is a finite automaton associated with a finite set of clocks, each clock counting the continuous elapsing of time. Each transition of the automaton can be guarded by a simple linear condition on the clock values, and can result in resetting some clocks to zero. A nice feature of this model is that it can be abstracted into a finite state system, and that all the standard verification problems (reachability, TCTL model-checking [ACD90, HNSY92]) are decidable. However, many interesting extensions of this model have been shown to lose this decidability property: for instance, as soon as clocks may be frozen in some states, to count cumulated durations, reachability becomes undecidable [KPSY93].

Hybrid automata [MMP91, ACHH93] are a much more general model, which has been proposed to model systems involving both discrete and continuous variables. A hybrid automaton is a finite automaton associated with a finite set of continuous variables. Each transition of the automaton can be guarded by a condition on these variables, and can perform an action modifying their values (discrete change). In each state of the automaton, the variables continuously vary, according to a system of differential equations associated with the state. So, timed automata are a special kind of hybrid automata, where the guards are simple linear conditions, discrete actions are only "reset" actions, and in each state, the variable derivatives are equal to 1. If clocks may be frozen, the derivatives belong to $\{0,1\}$.

In this paper, we consider a class of *linear* hybrid automata (which has been identified elsewhere [KPSY93, ACHH93, AHH93]), where guards are (general) linear conditions, actions are assignments of linear expressions, and derivatives are (symbolic) constants subject to linear

*This work has been partly supported ESPRIT-BRA action "REACT" and by a grant from Schneider-Electric.
†Authors'address: Verimag, Miniparc-Zirst, 38330 - Montbonnot, France.
e-mail: {Nicolas.Halbwachs,Pascal.Raymond,Yann-Eric.Proy}@imag.fr
‡Verimag is a joint laboratory of CNRS, Institut National Poly technique de Grenoble, Université Joseph Fourier and Verilog SA associated with IMAG.

inequalities. This class is general enough to handle many practical problems, and the linear restrictions allow easy symbolic analysis.

Since verification problems are undecidable, we apply an approximate reachability analysis technique, proposed a long time ago [CH78], and based on abstract interpretation. This technique consists in approximating sets of numerical states by their convex hull — a convex polyhedron, i.e., the set of solutions of a system of linear inequalities — and by enforcing the convergence of iterative computations by extrapolation. This technique is particularly suitable for the analysis of linear hybrid systems, on one hand because it copes with continuous variables, and on the other hand, because the behavior of these variables is naturally linear.

2 Hybrid automata

Hybrid automata [MMP91, ACHH93] have been introduced to model hybrid systems. They involve discrete transitions between states (called locations) and continuous evolution within each location.

2.1 Definition

A **hybrid automaton** $H = \{Loc, Var, Init, Trans, Act, Inv\}$ consists of 6 components:
– A finite set *Loc* of **locations**
– A finite set *Var* of **real-valued variables**. A **valuation** ν is a function which assigns a real value $\nu(x) \in \mathbb{R}$ to each variable $x \in Var$. A **configuration** is a pair (ℓ, ν) where ℓ is a location and ν is a valuation.
– A set *Init* of **initial configurations**.
– A finite set *Trans* of **transitions**. Each transition $\tau = (\ell, \gamma, \alpha, \ell')$ consists of a **source location** $\ell \in Loc$, a **target location** $\ell' \in Loc$, a **guard** γ which is a function from valuations to $\{true, false\}$, and an **action** α which is a function from valuations to valuations. A transition $\tau = (\ell, \gamma, \alpha, \ell')$ is **enabled** in a configuration (ℓ, ν) if and only if $\gamma(\nu) = true$. The configuration $(\ell', \alpha(\nu))$ is then the **transition successor** of (ℓ, ν) by τ.
– A labeling function *Act* which assigns to each location a set of **activities**. An activity is a function from nonnegative reals $\mathbb{R}^{\geq 0}$ to valuations, whose value at $t = 0$ is the null valuation $\lambda x.0$. An activity expresses a continuous change of the valuation along the time: if the automaton reaches a location ℓ with valuation ν and chooses the activity $f \in Act(\ell)$, after staying in ℓ for a delay δ, the valuation will be $\nu + f(\delta)$.
– A labeling function *Inv* which assigns to each location an **invariant**, which is a function from valuations to $\{true, false\}$.

At any instant, the configuration of the system is given by a control location and values for all variables. The configuration can change in two ways:

- an enabled **discrete** transition can instantaneously change both the control location and the current variable valuation;

- a **time delay** can change only the valuation according to an activity associated with the current location; such a delay can only take place as long as the valuation satisfies the invariant associated with the location.

More formally, a **run** of the automaton is a finite or infinite sequence $\sigma_0 \rightarrow^{t_0}_{f_0} \sigma_1 \rightarrow^{t_1}_{f_1} \sigma_2 \rightarrow^{t_2}_{f_2} \dots$ of configurations $\sigma_i = (\ell_i, \nu_i)$, nonnegative reals t_i and activities $f_i \in Act(\ell_i)$ such that $\sigma_0 \in Init$ and, for all $i \geq 0$,

1. for all $t \in [0, t_i[$, $Inv(\ell_i)(\nu_i + f_i(t)) = true$
2. the configuration σ_{i+1} is a transition successor of the configuration $\sigma'_i = (\ell_i, \nu_i + f_i(t_i))$.

An infinite run diverges if the infinite sum $\sum_{i \geq 0} t_i$ diverges. With such a divergent run, we can associate a **behavior**, which is a total function β from time to valuations defined as follows:

$$\beta(t) = \nu_{i(t)} + f_{i(t)}(t - \sum_{j<i(t)} t_j), \quad \text{where } i(t) = \min\{\, k \mid \sum_{j=0}^{k} t_j > t \,\}$$

A configuration (ℓ, ν) is **reachable** iff there exists a divergent run and an instant t such that $\ell = \ell_{i(t)}$ and $\nu = \nu_{i(t)} + f_{i(t)}(t - \sum_{j<i(t)} t_j)$. Let *Reach* denote the set of reachable configurations.

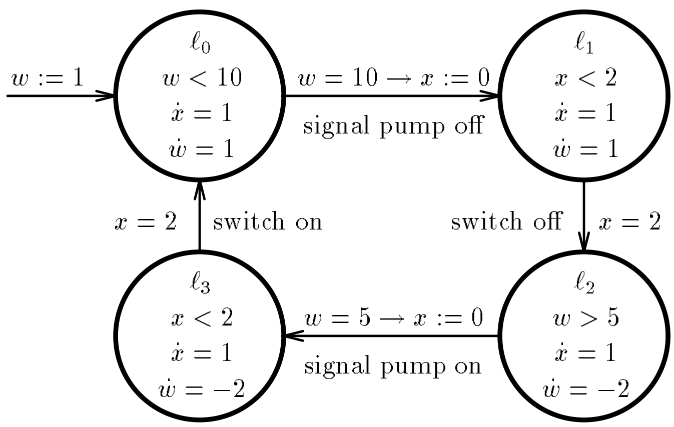
We will represent a hybrid automaton by means of a directed graph, whose nodes represent the locations and edges represent transitions. Let us illustrate the use of the model by means of some classical examples.

2.2 Examples

2.2.1 A water-level monitor:

The water level in a tank is controlled through a monitor, which continuously senses the water level and turns a pump on and off. The water level changes as a piecewise-linear function over time: when the pump is off, the water level, denoted by the variable w, falls by 2 inches per second; when the pump is on, the water level rises by 1 inch per second. Suppose that initially the water level is 1 inch and the pump is turned on. We wish to keep the water between 1 and 12 inches. But from the time that the monitor signals to change the status of the pump to the time that the change becomes effective, there is a delay of 2 seconds. Thus the monitor must signal to turn the pump on before the water level falls to 1 inch, and it must signal to turn the pump off before the water level reaches 12 inches.

The hybrid automaton shown besides describes a water level monitor that signals whenever the water level passes 5 and 10 inches, respectively. The automaton has four locations: in locations ℓ_0 and ℓ_1, the pump is turned on; in locations ℓ_2 and ℓ_3, the pump is off. The clock x is used to specify the delays: whenever the automaton control is in location ℓ_1 or ℓ_3, the signal to switch the pump off or on, respectively, was sent x seconds ago.



On each transition, we give the guard and the action (if any). In each location, we give the label, the invariant, and the activities, expressed by their derivatives. For instance, the invariant associated with location ℓ_3 is $x < 2$, and the derivatives of x and w are 1 and -2, respectively. It means that the only activity in $Act(\ell_3)$ is $\lambda t.(x \mapsto t, w \mapsto -2t)$.

2.2.2 Fischer mutual exclusion protocol:

This example describes a parameterized multirate timed system. It is a timing-based algorithm that implements mutual exclusion for a distributed system with skewed clocks. Consider an asynchronous shared-memory system that consists of two processes P_1 and P_2 with atomic read and write operations. Each process has a critical section and at each instant, at most one of the two processes is allowed to be in its critical section. Mutual exclusion is ensured by a version of Fischer's protocol [Lam87], which we describe first in pseudocode. The code executed by process $P_i (i = 1, 2)$ is shown beside.

repeat
  repeat
    await $k = 0$
    $k := i$
    delay b
  until $k = i$
  Critical section
  $k := 0$
forever

The two processes P_1 and P_2 share a variable k, and process P_i is allowed to be in its critical section iff $k = i$. Each process has a private clock. The statement "**delay** b" delays a process for at least b time units as measured by the process's local clock. Furthermore, each process takes at most a time units, as measured by the process's clock, for a single write access to the shared memory (i.e., for the assignment $k := i$). The values of a and b are the only information we have about the timing behavior of processes. Clearly, the protocol ensures mutual exclusion only for certain values of a and b. If both private processor clocks proceed at precisely the same rate, then mutual exclusion is guaranteed iff $a < b$.

To make the example more interesting, we assume that the private clocks of the processes P_1 and P_2 proceed at different rates, namely, the rate of the local clock of P_2 is between 0.9 and 1.1 times the rate of the clock of P_1.

The resulting system could be modeled by the product of the two hybrid automata, each of which modeling one process. Instead, for clarity in the further treatment of this example, we give an abstract view of the system, where the behavior of P_2 is abstracted. Fig. 1 gives the behavior of P_1 with only the relevant interactions with P_2. Variables x and y are used to count delays, with respect to P_1's and P_2's local clock, respectively. In location ℓ_0, P_1 is idle, in ℓ_1, it has read $k = 0$. On the transition from ℓ_1 to ℓ_2, P_1 is supposed to set k to 1, so it is the last time P_2 can read $k = 0$. In ℓ_2, P_1 waits for b; two transitions may occur: either the delay b expires, and P_1 enters the critical section (ℓ_4), or P_2 sets k to 2 (ℓ_3) thus forbidding P_1 to enter the critical section. In ℓ_4, P_1 is in the critical section; if P_2 can set k to 2, it can also enter the critical section, and the mutual exclusion is violated (location ℓ_5).

2.2.3 A scheduler:

Our last example is a task scheduler. We consider two classes of tasks, activated by interrupts. Interrupt I_1 (resp. I_2) occurs at most once each 10 (resp. 20) time units and activates a task of the first (resp. second) class, which takes 4 (resp. 8) time units. Tasks of the second class have priority, and can preempt other tasks. We want to show that a task of the second class never waits.

We use two timers, $c_i (i = 1, 2)$, to count the delay elapsed since the last interrupt I_i. The assumptions about interrupt frequencies can be expressed by the automaton of Fig. 2.(a). Concerning tasks, (see Fig. 2.(b)), we use two other timers, $x_i (i = 1, 2)$, to count the execution time of tasks, and two counters $k_i (i = 1, 2)$, to count the number of pending tasks in each class (these counters are discrete variables, their derivative is supposed to be 0 in any location).

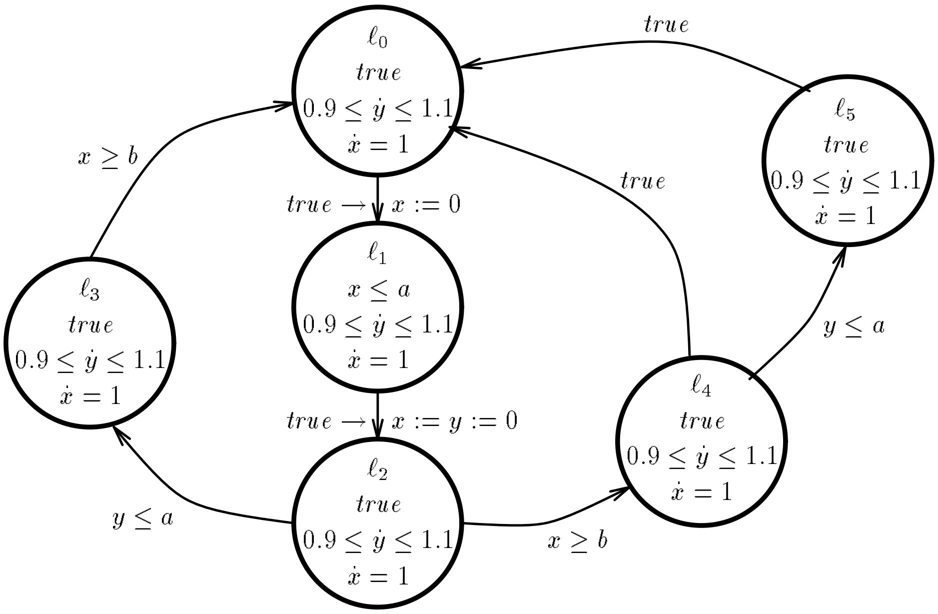


Figure 1: Fischer protocol (simplified)

A typical behavior of this automaton is to start in location ℓ_0 (idle), then receiving an interrupt I_1 and activating a task 1 (in location ℓ_1) for 4 time units — counted by the timer x_1. If, during this execution, an interrupt I_2 occurs, the task 1 is suspended, and the scheduler executes a task 2 (in location ℓ_2) for 8 time units — counted by x_2. Notice that the timer x_1 is only frozen in ℓ_2, so, on termination of all the pending tasks 2, the suspended task 1 can be completed. The occurrence of an interrupt which does not have priority upon the active task only results in incrementing the corresponding counter.

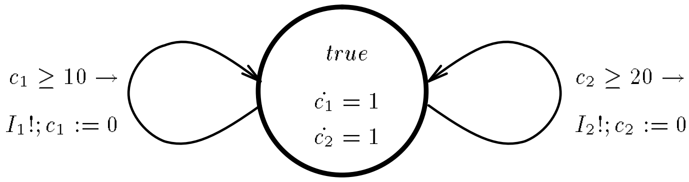
2.3 Linear hybrid automata

Of course, hybrid automata are much too general to allow other analysis techniques than simulation. Interesting subclasses have been identified:

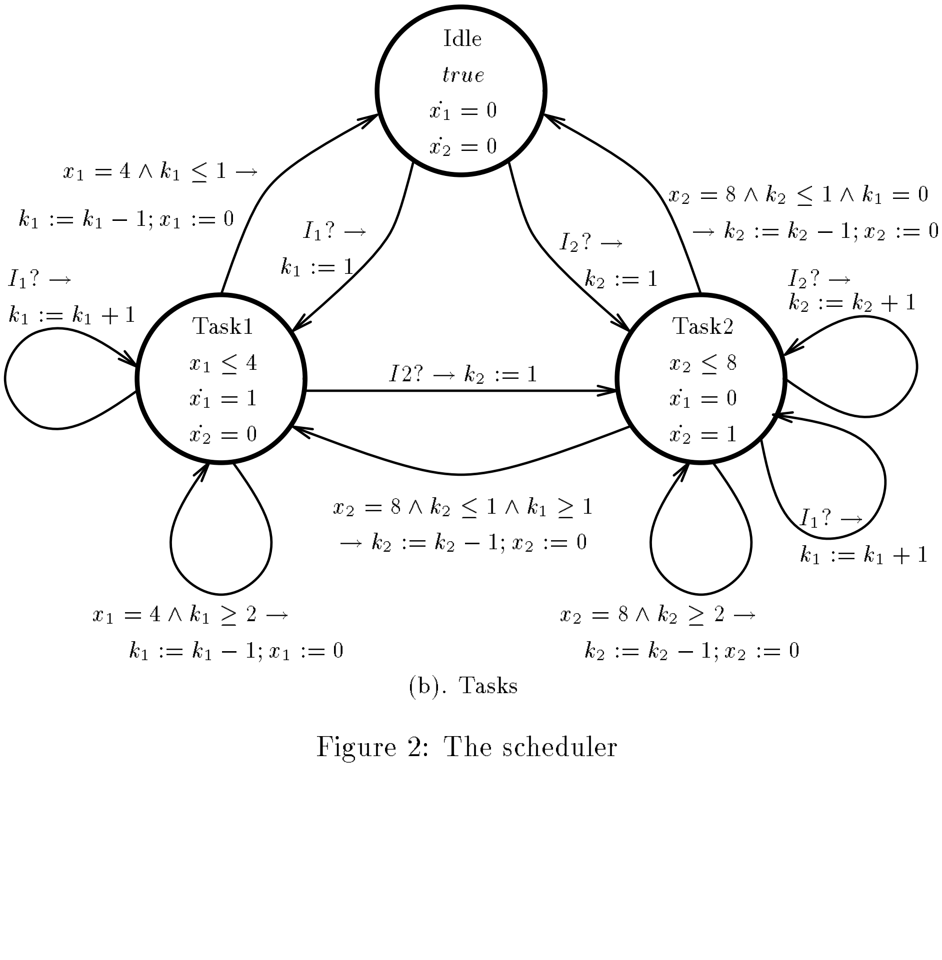
Timed automata are hybrid automata where there is only one allowed activity: each variable has a derivative equal to 1. Moreover, the only actions that can be performed on variables are reset actions. So variables are only timers which count the elapsed time since they have been reset. Finally, invariants and guards may only be comparison of variables with integer constants.

Integration graphs obey the same restrictions as timed automata, but derivatives may belong to $\{0, 1\}$. So, timers may be "frozen" in some locations. Our scheduler (§2.2.3) is an integration graph.

Linear hybrid automata are less restrictive: derivatives are supposed to be constants submitted to linear constraints. Actions are assignments of linear expressions; invariants and guards are systems of linear constraints. All the examples given above belong to this class.



(a). Interrupts



(b). Tasks

Figure 2: The scheduler

In the remainder of this paper, we will restrict ourselves to linear hybrid automata. First, we make precise some notions.

Let $Var = \{x_1, \ldots, x_n\}$ be the set of variables. A *linear constraint* is given by a triple (a, ρ, b) where $a \in \mathbb{Z}^n$, $b \in \mathbb{Z}$ and $\rho \in \mathcal{R} = \{<, \leq, =, \geq, >\}$. It characterizes the subset of $\mathbb{R}^n$ made of all the valuations ν satisfying $a\nu\ \rho\ b$. Notice that, for obvious computational reasons, we restrict ourselves to integer coefficients. A conjunction of m linear constraints, given by a triple $(A, R, B) \in (\mathbb{Z}^n)^m \times \mathcal{R}^m \times \mathbb{Z}^m$, will be simply called a *linear system*. Notice that the set of solutions of a linear system is *convex*. We will often assimilate a linear system with the set of its solutions. A *linear assignment* is given by a pair (A, B), $A \in \mathbb{Z}^{n\times n}$, $B \in \mathbb{Z}^n$ and defines the function $\lambda\nu.A\nu + B$.

In a linear hybrid automaton, with each location ℓ are associated two linear systems: the invariant Inv_ℓ, and the system D_ℓ constraining variable's derivatives. With each transition τ is associated a linear system γ_τ (the guard) and a linear assignment α_τ (the action). The set of initial configurations is supposed to be given as a (possibly unsatisfiable) linear system $Init_\ell$ associated with each location ℓ ($Init_\ell = \{\nu \mid (\ell, \nu) \in Init\}$).

2.4 Forward collecting semantics of linear hybrid automata

Let us recall that *Reach* denotes the set of reachable configurations. For any location ℓ, we note $Reach_\ell$ the set of reachable valuations at location ℓ:

$$Reach_\ell \ = \ \{\nu \mid (\ell, \nu) \in Reach\}$$

In this section, we will characterize the tuple $(Reach_\ell)_{\ell \in Loc}$ by means of a system of fixpoint equations. This system will be constructed in a "forward" way, in the sense that, for each location ℓ, $Reach_\ell$ will be defined as a function of the sets $Reach_{\ell'}$, where ℓ' runs over the source location of transitions incoming to ℓ:

$$Reach_\ell = F_\ell\left(\{Reach_{\ell'} \mid (\ell', \gamma, \alpha, \ell) \in Trans\}\right)$$

We first introduce some operations on sets of valuations. Let $\alpha = (A, B)$ be a linear assignment, and S be a set of valuations. We note $\alpha(S)$ the image of S by α: $\alpha(S) \ = \ \{A\nu + B \mid \nu \in S\}$

Let D be a linear system (supposed to be a domain of derivatives), and S be a set of valuations. We note $S \nearrow D$ the set of valuations that can be obtained by letting the variables continuously evolve, according to a constant derivative belonging to D, and starting from a valuation belonging to S:

$$S \nearrow D \ = \ \{\nu + td \mid \nu \in S,\ d \in D\ , t \in \mathbb{R}^{\geq 0}\}$$

This operator will be called the *time elapse operator*.
Now, we are able to define the set $Reach_\ell$:

$$Reach_\ell \ = \ \left(\left(Init_\ell \cup \bigcup_{(\ell', \gamma, \alpha, \ell) \in Trans} \alpha(Reach_{\ell'} \cap \gamma) \cap Inv_\ell\right) \nearrow D_\ell\right) \cap Inv_\ell \tag{1}$$

This equation expresses that a reachable valuation in location ℓ satisfies the invariant Inv_ℓ, and is obtained by letting the time elapse from either an initial valuation or from an incoming valuation satisfying the invariant (notice that, since the invariant defines a convex domain, and since the continuous time-elapsing transformation is linear, any time-elapsing behavior starting from a valuation satisfying the invariant and leading to a valuation satisfying the invariant, continuously satisfies the invariant). An incoming valuation is obtained from a valuation associated with the source location of an incoming transition, satisfying the guard of the transition, as the result of the action of the transition.

3 Linear analysis

The iterative resolution of the system (1) raises two problems:

(i) How to represent sets of valuations, which are subsets of $\mathbb{R}^n$, and perform computations on such sets?

(ii) How to deal with infinite iterations?

In this section, we apply an abstract interpretation proposed a long time ago [CH78] to compute automatically an upper approximation of the solution. This technique is especially well-suited to this problem, since it approximates a set of numerical vectors by a convex polyhedron, i.e., the set of solutions of a system of linear constraints.

3.1 Convex approximation

In the convex approximation technique proposed in [CH78], a set S of vectors of $\mathbb{R}^n$ is approximated by its *convex hull*, i.e., the least convex polyhedron containing S. Well-known algorithms are available to compute over convex polyhedra. An abstract system of equations can be associated with the system (1), whose least solution is a tuple $(P_\ell)_{\ell \in Loc}$ of polyhedra, such that $\forall \ell$, $Reach_\ell \subseteq P_\ell$. The abstract system of equations is almost the same as the system (1), since almost all the involved operations (intersection, linear assignment, time-elapse) transform polyhedra into polyhedra. Only the set union must be replaced by the convex-hull operation, noted "$\sqcup$":

$$P_\ell \;=\; \left(\left(Init_\ell \sqcup \bigsqcup_{(\ell',\gamma,\alpha,\ell)\in Trans} \alpha(P_{\ell'} \cap \gamma) \cap Inv_\ell\right) \nearrow D_\ell\right) \cap Inv_\ell \tag{2}$$

We explain now the principles of computation over convex polyhedra.

3.2 Algorithms on polyhedra

We first consider the case of *closed polyhedra*, i.e., those which are defined by means of equations and loose inequalities. The case of strict inequalities is considered in the next subsection.

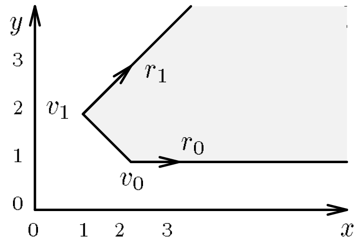
3.2.1 Closed convex polyhedra:

The following presentation is mainly borrowed from [Hal93]. Let us recall that a closed convex polyhedron P (a closed polyhedron, for short) has two representations (see Fig. 3):

- it is the set of solutions of a *system of loose linear inequalities* $P = \{X \mid AX \geq B\}$ where A is a $m \times n$-matrix and B is a m-vector.

- it is the convex closure of a *system of generators*, i.e., two finite sets V and R (respectively for "vertices" and "rays") of n-vectors such that

$$P = \left\{ \sum_{v_i \in V} \lambda_i . v_i + \sum_{r_j \in R} \mu_j . r_j \;\middle|\; \lambda_i \geq 0, \mu_j \geq 0, \sum_i \lambda_i = 1 \right\}$$



$$P = \left\{ (x,y) \mid \left(\begin{array}{rcl} y & \geq & 1 \\ x+y & \geq & 3 \\ -x+y & \leq & 1 \end{array} \right) \right\}$$

$$V = \left\{ v_0 \begin{pmatrix} 2 \\ 1 \end{pmatrix}, v_1 \begin{pmatrix} 1 \\ 2 \end{pmatrix} \right\} \quad R = \left\{ r_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix}, r_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

Figure 3: The two representations of a closed convex polyhedron

These two representations are dual. There exist efficient algorithms [Che68, LeV92] for translating each representation into the other; these algorithms also minimize the representations. We will use the following basic operations on closed polyhedra:

Intersection: The intersection of two closed polyhedra P and Q is a closed polyhedron whose system of linear inequalities is the conjunction of those of P and Q.

Convex hull: Let $(V, R), (V', R')$ be the respective systems of generators of two closed polyhedra P and Q. Then $(V \cup V'\ ,\ R \cup R')$ is a system of generators of their convex hull $P \sqcup Q$.

Linear assignment: Let $\alpha = (A, B)$ be a linear assignment, and (V, R) be the system of generators of a closed polyhedron P. Then $(V' = \{Av + B \mid v \in V\}\ ,\ R' = \{Ar \mid r \in R\})$ is a system of generators of $\alpha(P)$.

Time elapse: Let $(V, R), (V', R')$ be the respective systems of generators of two closed polyhedra P and D. Then $(V\ ,\ R \cup V' \cup R')$ is a system of generators of $P \nearrow D$.

Test for emptyness: A closed polyhedron is empty if and only if it has no vertices.

Test for inclusion and equality: A closed polyhedron P, with system of generators (V, R), is included in a closed polyhedron Q, defined by the system of inequalities $AX \geq B$, if and only if $\forall v \in V,\ Av \geq B$ and $\forall r \in R,\ Ar \geq 0$. The equality of two closed polyhedra is decided by showing the double inclusion.

3.2.2 Strict inequalities:

The case of strict inequalities was not considered in previous applications of the method [CH78, Hal93], since only discrete integer variables were considered. In fact, strict inequalities can easily be handled, by adding an auxiliary variable, say ε, with $0 \leq \varepsilon \leq 1$, replacing any strict inequality $aX > b$ by $aX - \varepsilon \geq b$, and checking that ε is not null when checking for polyhedron emptyness and inclusion. More precisely, if $AX \geq B \ \wedge\ A'X > B'$ is the system of inequalities of a polyhedron P of $\mathbb{R}^n$, let $\widehat{P}$ be the polyhedron of $\mathbb{R}^{n+1}$ defined by the system $AX \geq B \wedge A'X - \varepsilon \geq B' \wedge 0 \leq \varepsilon \leq 1$. We note $\widehat{X}$ the extended vector of variables, and $\widehat{A}\widehat{X} \geq \widehat{B}$ the system of constraints of $\widehat{P}$. Let $\widehat{V}$ be the set of vertices of $\widehat{P}$, and $\widehat{V}^{>0}$, the subset of those vertices whose ε-component is strictly positive. Then,
Test for emptyness: A polyhedron P is empty if and only if $\widehat{V}^{>0}$ is empty.

Test for inclusion: We note $\overset{\frown}{P}$ the closure of P, i.e., the polyhedron defined by the same system of constraints as P where all the inequalities are considered to be loose. Let $\widehat{V}$ be the set of vertices of $\widehat{P}$. If $v \in \widehat{V}^{>0}$, we note $v{\downarrow}$ its projection onto $\mathbb{R}^n$ according to ε (i.e., the result of removing the ε component of v). Then P is included in another polyhedron Q if and only if $\overset{\frown}{P} \subseteq \overset{\frown}{Q} \ \wedge\ \forall v \in \widehat{V}^{>0},\ v{\downarrow} \in Q$.

3.3 Extrapolation

Let us come back to the fixpoint system (2) and rewrite it

$$P_\ell \;=\; F_\ell(\vec{P}), \ell \in \mathit{Loc} \tag{3}$$

where $\vec{P}$ stands for the tuple $\{P_\ell \,|\, \ell \in \mathit{Loc}\}$. Such a system can be iteratively solved by computing the limit of the sequence $\vec{P}^{(0)}, \ldots, \vec{P}^{(k)}, \ldots$, where

$$\vec{P}_\ell^{(0)} \;=\; \emptyset \quad \forall \ell \in \mathit{Loc} \quad , \qquad \vec{P}_\ell^{(k+1)} \;=\; F_\ell(\vec{P}^{(k)}) \quad \forall \ell \in \mathit{Loc}, \forall k \in \mathbb{N}$$

The last problem is that this sequence may diverge. So, we apply the well-known "widening" technique [CC77, CC92], to enforce the convergence of the iterations by extrapolating an upper approximation of the limit. We have to define a *widening operator* on polyhedra, traditionally noted ∇, and satisfying the following properties:

- For any polyhedron P, $\emptyset \nabla P \;=\; P$;
- For each pair (P_1, P_2) of polyhedra, $P_1 \sqcup P_2 \;\subseteq\; P_1 \nabla P_2$;
- For any increasing chain $(P_0 \subseteq P_1 \subseteq \ldots)$ of polyhedra, the increasing chain defined by $P'_0 = P_0$, $P'_{i+1} = P'_i \nabla P_{i+1}$, is not strictly increasing (i.e., stabilizes after a finite number of terms).

Then, let $\mathcal{W}$ be a set of locations "cutting" each loop of the hybrid automaton (i.e., each loop of the locations–transitions graph contains at least one location in $\mathcal{W}$[1]). The iterative resolution of the system:

$$P_\ell \;=\; P_\ell \nabla F_\ell(\vec{P}), \quad \forall \ell \in \mathcal{W}' \quad , \qquad P_\ell \;=\; F_\ell(\vec{P}), \quad \forall \ell \in \mathit{Loc} \setminus \mathcal{W} \tag{4}$$

converges after a finite number of steps towards an upper approximation of the least solution of the system (3).

We use the standard widening operator defined in [Hal79]. Let P and Q be two polyhedra. Roughly speaking, the widening $P \nabla Q$ is obtained by removing from the system of P all the inequalities which are not satisfied by Q. Here is an example (see Fig. 4.a):

$$P = \{(x,y) \mid 0 \le y \le x \le 1\}, \quad Q = \{(x,y) \mid 0 \le y \le x \le 2\}$$

$$P \nabla Q = \{(x,y) \mid 0 \le y \le x\}$$

The intuition is clear: whenever a constraint is translated or rotated, it can do so infinitely many times, so it is removed. This operator clearly satisfies the properties of a widening: the result contains both the operands, and since the system of inequalities of $P \nabla Q$ is a subset of the one of P, the widening cannot be infinitely iterated without convergence.

The actual operator is a bit more complicated: first, whenever P is empty, $P \nabla Q \;=\; Q$. Moreover, if P is included in a strict subspace of $\mathbb{R}^n$, its minimal system of inequalities is not canonical. It can be first rewritten into an equivalent system maximizing the number of inequalities satisfied by Q, and thus kept in the result. For instance, consider:

$$P = \{(x,y) \mid x = 0 \,\wedge\, y = 0\} \quad , \quad Q = \{(x,y) \mid 0 \le y \le x \le 1\}$$

The system of inequalities of P can be first rewritten into $\{(x,y) \mid 0 \le y \le x \le 0\}$ before performing the widening, which evaluates to $P \nabla Q = \{(x,y) \mid 0 \le y \le x\}$ (see Fig. 4.b) instead of $\{(x,y) \mid 0 \le y \,\wedge\, 0 \le x\}$, which would be obtained without rewriting. This optimization preserves the widening properties. A simple algorithm has been proposed for it [Hal79].

[1] The determination of a minimal set of such "cuts" is NP-complete, but we don't need $\mathcal{W}$ to be minimal.

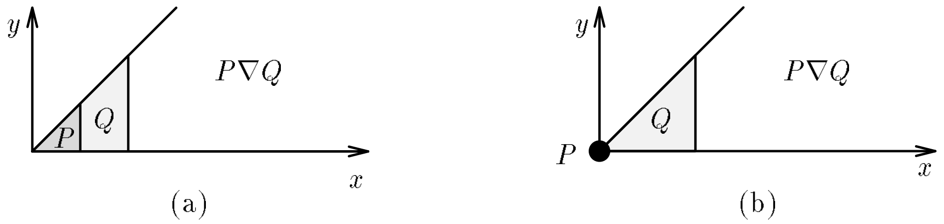


Figure 4: Widening operation

4 Applications

Let us illustrate the use of our method on the examples given in section 2.2. We will give some details about the analysis of the water-level monitor, and simply the results obtained for the other two examples.

4.1 Water-level monitor

The system of equations corresponding to the water-level monitor is the following:

$$\begin{aligned}
P_{\ell_0} &= ((\{w=1\} \sqcup (P_{\ell_3} \cap \{x=2\} \cap \{w<10\})) \nearrow \{\dot{x}=\dot{w}=1\}) \cap \{w<10\} \\
P_{\ell_1} &= (((P_{\ell_0} \cap \{w=10\})[x:=0] \cap \{x<2\}) \nearrow \{\dot{x}=\dot{w}=1\}) \cap \{x<2\} \\
P_{\ell_2} &= (((P_{\ell_1} \cap \{x=2\} \cap \{w>5\})) \nearrow \{\dot{x}=1, \dot{w}=-2\}) \cap \{w>5\} \\
P_{\ell_3} &= (((P_{\ell_2} \cap \{w=5\})[x:=0] \cap \{x<2\}) \nearrow \{\dot{x}=1, \dot{w}=-2\}) \cap \{x<2\}
\end{aligned}$$

We choose $\mathcal{W} = \{\ell_0\}$, thus replacing P_{ℓ_0}'s definition by

$$P_{\ell_0} = (P_{\ell_0} \nabla ((\{w=1\} \sqcup (P_{\ell_3} \cap \{x=2\} \cap \{w<10\})) \nearrow \{\dot{x}=\dot{w}=1\})) \cap \{w<10\}$$

The resolution converges after 3 iterations, needs 0.3 seconds of CPU time on a SUN4-SPARC, and provides the following results:

$$\begin{aligned}
P_{\ell_0} &= \{1 \leq w < 10\} & P_{\ell_1} &= \{w = x+10 \wedge 0 \leq x < 2\} \\
P_{\ell_2} &= \{2x+w=16 \wedge 4 \leq 2x < 11\} & P_{\ell_3} &= \{2x+w=5 \wedge 0 \leq x < 2\}
\end{aligned}$$

from which we can easily conclude that

$1 \leq w < 10$ in location ℓ_0 $\quad$ $10 \leq w < 12$ in location ℓ_1

$5 < w \leq 12$ in location ℓ_2 $\quad$ $1 < w \leq 5$ in location ℓ_3

4.2 Fischer mutual exclusion protocol

We give the results of the analysis of the mutual exclusion protocol of §2.2.2:

$$
\begin{aligned}
P_{\ell_0} &= \{a \geq 0 \wedge b \geq 0\} \\
P_{\ell_1} &= \{b \geq 0 \wedge 0 \leq x \leq a\} \\
P_{\ell_2} &= \{a \geq 0 \wedge b \geq 0 \wedge 9x \leq 10y \leq 11x\} \\
P_{\ell_3} &= \{a \geq 0 \wedge b \geq 0 \wedge 9x \leq 10y \leq 11x\} \\
P_{\ell_4} &= \{a \geq 0 \wedge b \geq 0 \wedge 9x \leq 10y \leq 11x \wedge b \leq x\} \\
P_{\ell_5} &= \{0 \leq b \leq x \wedge 9x \leq 10y \leq 11x \wedge 9b \leq 10a \wedge 10a + 11x \geq 10y + 11b\}
\end{aligned}
$$

These results are obtained after two iterations, and the analysis takes 0.4 seconds of CPU time. Remember that ℓ_5 is the location where the mutual exclusion can be violated. Since, in P_{ℓ_5}, we have the constraint $10a \geq 9b$, this location is not reachable if $9b > 10a$. So, the analysis shows that $9b > 10a$ is a sufficient condition for the mutual exclusion to hold. In fact, this condition is also necessary, which shows that the analysis is precise. This example shows how, by dealing with symbolic constants (here the delays a and b), the analysis can be used to adjust some parameters to insure a desired property.

4.3 The scheduler

For the scheduler (§2.2.3), the analysis gives (in 4 iterations and 0.6 sec.) the following results:

- in the "idle" location: $c_1 \geq 0 \wedge c_2 \geq 0 \wedge x_1 = x_2 = k_1 = k_2 = 0$
- "task 1" running: $k_1 \geq 1 \wedge 0 \leq x_1 \leq 4 \wedge c_1 \geq 0 \wedge c_2 \geq 0 \wedge k_2 = x_2 = 0$
- "task 2" running: $0 \leq c_2 = x_2 \leq 8 \wedge 0 \leq x_1 \leq 4 \wedge c_1 \geq 0 \wedge x_1 \leq 4k_1 \wedge k_2 = 1$

So the analysis succeeds in showing that $k_2 \leq 1$, which means that a task of the second class never waits. But it fails to show that $k_1 \leq 2$, a fact that clearly appears from a straightforward simulation. Notice that this imprecision is not due — as it often happens — to the widening, but comes from the convex hull approximation[2].

5 Related works and conclusion

The most deeply studied class of hybrid systems is the one of timed systems. The *region graph* technique proposed in [ACD90, Alu91] can be viewed as an abstract interpretation giving exact results. This technique works on a special kind of linear systems, sometimes called *bound matrices*, which consist of relations of the form "$x \rho k$" or "$(x - y)\rho k$", where x and y are variables, k is an integer constant and ρ is an equality or inequality relation. The algorithmic of bound matrices is notably simpler than the one of general polyhedra. A region is a pair made of a location and a bound matrix, such that all the valuations satisfying the bound matrix are time-bisimilar. The key result is that any timed automaton can be abstracted into a finite graph of regions. Several verification tools have been based on this approach [HNSY92, ACD+92]. Because of the

[2]For the same example, [HH94] solves the problem by changing the control structure of the automaton, by distinguishing the locations according to the values of k_1 and k_2. They get exact results, but of course, such a change in the structure could not be found automatically, and may involve an explosion of the size of the automaton.

exponential proliferation of regions, approximate methods converging towards exact results have been proposed in this field [WTD93]. Concerning linear hybrid systems, semi-decision procedures have been proposed [ACHH93, AHH93, NOSY93]. The closest work to our method is [HH94], where an extrapolation operation is proposed, which does not force the convergence. They use also a backward computation technique, which we could apply in our approach.

The convex analysis method presented in this paper has been already applied [CH78, Hal93], but only to discrete systems. Hybrid systems are a good application field for approximation techniques, first because interesting classes are on the border of decidable problems, and also because, even in the decidable case, approximation techniques are often interesting because of the very high cost of decision procedures. Our convex analysis is particularly suitable, because it fits well with continuous domains, and, of course, it provides accurate results in the case of linear automata.

Acknowledgements: We are indebted to Hervé Leverge for his very efficient program computing the convex hull, which is the basis of our prototype implementation. Alain Kerbrat helped us in the implementation of the polyhedra package. We thank also David Dill, Tom Henzinger and Pei-Hsin Ho for helpful discussions.

References

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model checking of real-time systems. In *Fifth IEEE Symposium on Logic in Computer Science*, Philadelphia, 1990.

[ACD+92] R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *13th IEEE Real-Time Systems Symposium*, Phoenix (Az), December 1992.

[ACHH93] R. Alur, C. Courcoubetis, T. A. Henzinger, and Pei-Hsin Ho. Hybrid automata: an algorithmic approach to the specification and analysis of hybrid systems. In *Workshop on Theory of Hybrid Systems*, Lyngby, Denmark, October 1993. LNCS 736, Springer Verlag.

[AD90] R. Alur and D. Dill. Automata for modeling real-time systems. In *ICALP'90*, 1990.

[AHH93] R. Alur, T. A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. In *RTTS93*, 1993.

[Alu91] R. Alur. Techniques for automatic verification of real-time systems. Phd thesis, Stanford University, August 1991.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages*, Los Angeles, January 1977.

[CC92] P. Cousot and R. Cousot. Comparing the Galois connection and widenning/narrowing approaches to abstract interpretation. Research Report LIX/RR/92/09, Ecole Polytechnique, June 1992.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages*, Tucson (Arizona), January 1978.

[Che68] N. V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

[Hal79] N. Halbwachs. Détermination automatique de relations linéaires vérifiées par les variables d'un programme. Thèse de 3e cycle, University of Grenoble, March 1979.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Fifth Conference on Computer-Aided Verification*, Elounda (Greece), July 1993. LNCS 697, Springer Verlag.

[HH94] T. A. Henzinger and P.-H. Ho. Model checking strategies for hybrid systems. In *Conference on Industrial Applications of Artificial Intelligence and Expert Systems*, 1994.

[HNSY92] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *LICS'92*, June 1992.

[KPSY93] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: a class of decidable hybrid systems. In *Workshop on Theory of Hybrid Systems*, Lyngby, Denmark, October 1993. LNCS 736, Springer Verlag.

[Lam87] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.

[LeV92] H. LeVerge. A note on Chernikova's algorithm. Research Report 635, IRISA, February 1992.

[MMP91] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In REX *Workshop on Real-Time: Theory in Practice*, DePlasmolen (Netherlands), June 1991. LNCS 600, Springer Verlag.

[NOSY93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In *Workshop on Theory of Hybrid Systems*, Lyngby, Denmark, October 1993. LNCS 736, Springer Verlag.

[WTD93] H. Wong-Toi and D. Dill. Using iterative approximations for timing verification. In *First AMAST International Workshop on Real-Time Systems*, Iowa City (Iowa), November 1993.