Task-System Analysis Using Slope-Parametric Hybrid Automata

Augusto Burgueño[1] * and Vlad Rusu[2]

[1] ONERA-CERT, Département d'Informatique,
2 av. E. Belin, BP4025, 31055 Toulouse Cedex 4, France.
a.burgueno@acm.org

[2] IRCyN (UMR CNRS N. 6597, Ecole Centrale de Nantes, Université de Nantes),
1 rue de la Noë, BP92101, 44321 Nantes Cedex 3, France.
Vlad.Rusu@lan10.ec-nantes.fr

Abstract. Slope-parametric hybrid automata (SPHA) are hybrid automata whose variables can have parametric slopes. SPHA are useful, in particular, for modeling task-control systems in which the task speeds can be adjusted for meeting some safety requirement. In this paper, we present an example of parametric analysis for a simple task system. We introduce a prototype verification tool that fully automates the analysis.

Keywords: real-time systems, hybrid automata, parametric polyhedra.

1 Introduction

The verification of real-time properties is nowadays a well-known problem, and its most successful resolution techniques [ACH+95] have been automated and applied to real-size systems [DY95, HH94, BGK+96]. It consists, clasically, in verifying a given (timed) property on a given model of the system, and thus obtaining a binary answer: 'the system satisfies/does not satisfy the property'. However, many problems arising in the field of verification are *parametric*. Indeed, for a system designer it is often more important to obtain quantitative information such as: (α) 'for protocol safety, the messages should arrive at destination in no more than 1 second' or (β) 'for the task system to operate correctly, task 1 should run at least 2 times faster than task 2'.

Parametric analysis is the subject of some study and application, although mainly dealing with (α)-like analysis: finding the possible values of *delays* for some property to be satisfied. In a short paper [BBRR97] we presented a first approach to (β)-like analysis: the parameters to be computed are *speeds* rather than delays.

* Partially supported by Research Grant of the Spanish Ministry of Education and Culture. This research was carried out in part while the author was visiting the IRCyN (formerly LAN).

In this paper, we present an application of this approach. The goal is to have a situation in which the correct parameter values are hard to find by classical, 'try and fail until success' verification, since the correct values lie in a broad interval. We find these values automatically, using a prototype tool that we have implemented using MAPLE V and Prolog IV.

Related work. Parametric analysis of hybrid and timed automata has also been treated from other view points: [AHV93, HH94] focus on delays, that is, parameters appear on guards; [CY91] follows a similar approach; [Wan96] defines Parametric TCTL and redefines the classical model checking algorithm; [KS97] and [HLM97] deal with the somehow similar problem of controller synthesis; finally, the only other work to our knowledge combining constraint solving and model checking is [CABN97].

2 Slope-Parametric Hybrid Automata

Slope-Parametric Hybrid Automata (SPHA) are a generalization of Multirate Automata [ACH+95] in which the rates (slopes) of variables can be parameters. Let $\mathbb{R}$ be the set of real numbers.

Syntax. A SPHA is a tuple $(\mathcal{L}, \mathcal{E}, \mathcal{V}, \mathcal{K}, invar, diff, guard, reset)$ where

- $\mathcal{L}$ is a finite set of *vertices*
- $\mathcal{E} \subseteq \mathcal{L} \times \mathcal{L}$ is a finite set of *edges*
- $\mathcal{V} = \{x_1, .., x_n\}$ is a finite set of *variables*
- $\mathcal{K} = \{k_1, .., k_m\}$ is a finite set of *parameters*
- *invar* is a function that associates to each vertex an *invariant* i.e. a predicate of the form $(\bigwedge_i x_i \sim c_i)$, where $\sim \in \{<, \leq, >, \geq\}$ and $c_i \in \mathbb{R}$
- *diff* is a function that associates to each vertex, a *parametric differential law* for each variable, i.e an expression of the form $dx/dt = \sum_{j=1}^{m} a_j \cdot k_j + b$, where $a_j, b \in \mathbb{R}$ and $k_j \in \mathcal{K}$ are parameters
- *guard* is a function that associates to each edge a *guard* i.e. a predicate of the form $(\bigwedge_i x_i \sim c_i)$, where $\sim \in \{<, \leq, >, \geq\}$ and $c_i \in \mathbb{R}$
- *reset* is a function that associates to each edge a *reset expression* i.e. an expression of the form $(\bigwedge_i x_i := 0)$.

The SPHA in figure 1 has five vertices (L_1, L_2, L_3, L_4, L_5), three variables (a_1, a_2, t) and one parameter K. Some evolution laws are parametric and some are constant. For example, at vertex L_2 the evolution laws of a_1 and a_2 are parametric ($\dot{a_1} = K, \dot{a_2} = 1 - K$) while the evolution law for t is constant ($\dot{t} = 1$). The edges are labeled with guards ($a_2 = 40$ for the edge from L_2 to L_4) and variable resets ($a_2 := 0$ for the same edge). The invariant for each vertex is also shown (e.g. $t \leq 100 \wedge a_1 \leq 30 \wedge a_2 \leq 40$ for vertex L_2).

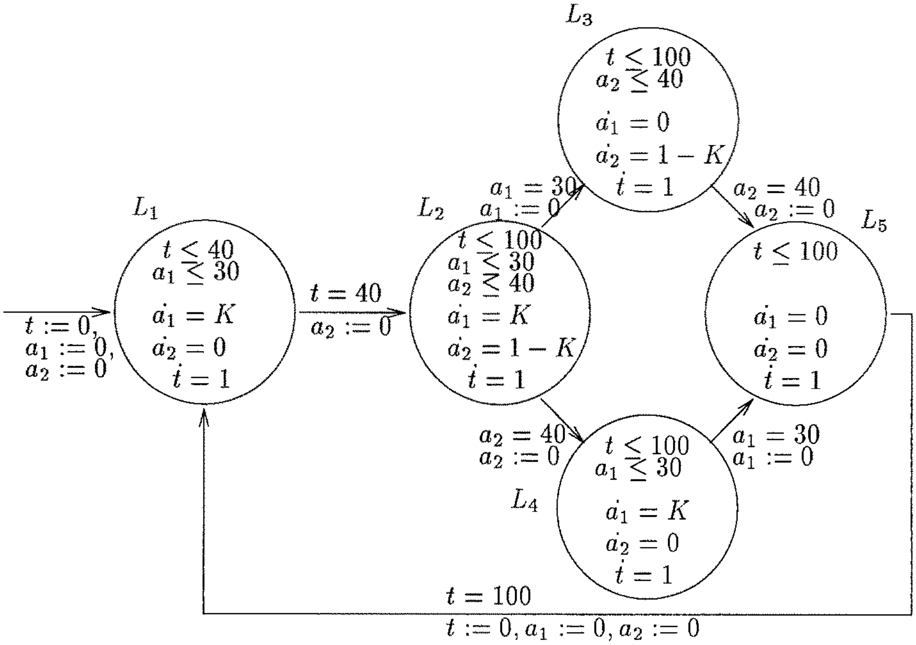

Figure 1. Example of slope-parametric hybrid automaton

Semantics. A *state* of a SPHA is defined, as in the case of Multirate Automata [ACH+95], by a couple (L, v) where $L \in \mathcal{L}$ is a vertex and v is a variable valuation, assigning a value $v(x)$ to each variable $x \in \mathcal{V}$. But in SPHA, variables can have *parametric* values i.e. a variable's value can be not only a real number, but also an expression on the parameters, as for example $v(x) = K^2 - 3 \cdot K + 1$.

A *run* of a SPHA consists in a sequence of states, obtained by letting the variables continuously evolve by their differential laws in some vertex, such that the vertex invariant is continuously satisfied; in crossing some outgoing edge, when the edge's guard is satisfied; and finally resetting the corresponding variables. The process is pursued in the newly reached vertex. For instance, in the SPHA represented in figure 1, consider a run starting at vertex L_1 with variables $t, a_1, a_2 = 0$. Then, evolution at vertex L_1 is given by $(\dot{a_1}, \dot{a_2}, \dot{t}) = (K, 0, 1)$, while the invariant $t \leq 40 \wedge a \leq 30$ holds. Vertex L_1 will be left, by crossing the edge from L_1 to L_2, when $t = 40$. But a_1, whose value is $K \cdot 40$ at the crossing moment, cannot exceed 30 (as stated by the vertex L_1 invariant); thus we have a condition on the parameter, in order to continue the run: $K \leq 3/4$.

The previous remark showed that *any run of a SPHA defines a sequence of conditions on the parameters.* Indeed, any edge guard must be satisfied by the variable values when that edge is crossed; since these values are expressions on the parameters, the guard satisfaction translates to a condition on the parameters. Thus, the analysis of SPHA focuses not only on the existence of a run between states (like in the case of plain hybrid automata), but also on computing and solving the associated sequence of conditions on the parameters. This is the goal of parametric analysis.

Parametric Analysis. The problem of parametric analysis is formulated as follows: given a *slope-parametric hybrid automaton*, a *temporal logic formula expressing a reachability property* and a *set of intervals in* $\mathbb{R}$ (one for each parameter), find the relations among parameters (in the corresponding intervals) such that the formula is true. For instance, in the automaton of figure 1, find values for parameter $K \in]0,1[$, for the TCTL formula $\varphi = \{(L_1 \wedge a_1 = 0 \wedge a_2 = 0 \wedge t = 0) \Rightarrow \neg[(\neg L_5)\exists\mathcal{U}_{=100}(\neg L_5)]\}$ to be true. This formula means that, starting from the initial set of states defined by vertex L_1 with $a_1 = 0, a_2 = 0, t = 0$, it is not possible to avoid vertex L_5 for a period of 100 units of time. In other words, vertex L_5 is inevitably reached within 100 time units.

3 Example

Consider the problem of assigning CPU time to processes that share a single processor. There are several ways to cope with this problem. One of them is to allocate resources statically, such that every time a process must execute, it will have its *time slice* reserved to do it. This is an approach followed in reactive programming, when processes must be launched as a reaction to an observed event: if the process has to compete at run time for resources, it may fail to get them and, therefore, it may fail its goal; if, on the contrary, resources are reserved in advance, it will always be able to run. In our example, processes a_1 and a_2 will share a CPU in fixed proportions: a percentage of time of $100 * K$ for a_1 and of $100 * (1 - K)$ for a_2.

Apart from these allocation considerations, there are other facts in the example that interest us. In particular, processes a_1 and a_2 must both be executed within a period of 100 time units. Total execution times are 30 time units for a_1 and 40 time units for a_2 (measured on the processor when each process takes all resources). Process a_2 must wait 40 time units from the begining of the period ($t = 0$) to be ready to execute, therefore, during the first 40 time units ($t \in [0, 40]$) only a_1 runs. From that instant on ($t \in]40, 100]$) a_1 and a_2 run concurrently (once one of them has finished, the other will continue alone until completion). As we said before, the processor allocation is fixed for each process disregarding whether they run alone or concurrently. We assume that these processes don't have any interaction and the waiting times (for I/O, for example) are null.

The slope-parametric hybrid automaton of figure 1 represents this schema. The expression $\dot{a_1} = K$ indicates that process a_1 executes using $100 * K$ per cent of CPU time. Similarly, the expression $\dot{a_2} = 1 - K$ indicates that process a_2 executes using $100 * (1 - K)$ per cent of CPU time. The expression $\dot{a_1} = 0$ says that the process a_1 does not execute at all. The fact of having a parameter (K) in our automaton allows us to define parametric analysis problems. One could be: determine the possible values of $K \in]0, 1[$ such that processes a_1 and

a_2 are both executed, once every 100 seconds. On the SPHA of figure 1, this translates to the property (already stated in section 2) that, starting from the initial situation, vertex L_5 is inevitably reached within 100 time units. A non trivial hand calculation shows that, for the above property to hold, K must lie in a broad interval: $K \in]0.3, \frac{1}{3}[$. We recall how to automate this calculus [BBRR97].

4 Computing parameter values

Parametric analysis comes to operating with parametric polyhedra. The latter are described by sets of linear equations and inequations whose coefficients can be either *constants* or *symbolic expressions on the parameters*; for instance, $0 \leq a_1 \leq 30 \ \wedge 0 \leq a_2 \leq 40 \wedge 40 \leq t \wedge a_1 - K \cdot t = 0$, where K is a parameter. The operations involved in parametric analysis are *extension*, *restriction* and *projection*, that incrementally generate conditions on the parameters, at each passage from a vertex to the next one.

Example. Consider the vertex L_2 of the automaton in figure 1 and an initial parametric polyhedron $0 \leq a_1 \leq 30 \wedge 0 \leq a_2 \leq 40 \wedge 40 \leq t \wedge a_1 - K \cdot t = 0$. We want to find the conditions on the parameter K, for the control to go from L_2 to L_4. For this we *extend* the initial polyhedron in the parametric direction given by $\dot{a_1} = K, \dot{a_2} = 1 - K$ and $\dot{t} = 1$; as we shall see, this extension has a finite number of different forms, under different conditions on K. Then, we *restrict* the values of parameter K such that the previous extension intersects the guard $a_2 = 40$ and the invariant $t \leq 100 \ \wedge a_1 \leq 30 \wedge a_2 \leq 40$: the intersection polyhedron is non-empty iff parameter K satisfies some further conditions. Once we have obtained these conditions, we continue at vertex L_4 by *projecting* the intersection polyhedron on plane $a_2 = 0$ (this corresponds to reinitializing a_2 on the transition). We should of course, apply the same sequence of operations in L_4 to find the conditions on K to cross the edge from L_4 to L_5.

4.1 Extension

Extension means: given a parametric polyhedron $P = \bigwedge_{j=1}^{m} (\Sigma_{i=1}^{n} a_{i,j} \cdot x_i \succ b_j)$ where $a_{i,j}, b_j$ can be constants or symbolic expressions and $\succ \in \{>, \geq\}$, find the polyhedron: $\overrightarrow{P} = \exists \tau \geq 0. \left[\bigwedge_{j=1}^{m} (\Sigma_{i=1}^{n} a_{i,j} \cdot (x_i - k_i \cdot \tau) \succ b_j)\right]$ where each k_i is the slope for variable x_i (constant or symbolic expressions on parameters). This is *forward continuous simulation* [ACH+95] except that we consider parametric polyhedra and directions. This imposes to consider several cases for eliminating the $\exists$ quantifier in the equivalent expression $\overrightarrow{P} = \exists \tau \geq 0. \left[\bigwedge_{j=1}^{m} (\Sigma_{i=1}^{n} a_{i,j} \cdot x_i - \tau \cdot (\Sigma_{i=1}^{n} a_{i,j} \cdot k_i) \succ b_j)\right]$.

The cases to consider are, for each sum-of-products $\Sigma_{i=1}^{n} a_{i,j} \cdot k_i$, the possibility that it is negative, positive or 0. As in [AHH93] we eliminate the existential quantifier by dropping the inequations that correspond to negative sums-of-products ($\Sigma_{i=1}^{n} a_{i,j} \cdot k_i < 0$), keeping the inequations that correspond to positive or zero sums-of-products ($\Sigma_{i=1}^{n} a_{i,j} \cdot k_i \geq 0$), and linearly combining pairs of inequations that correspond to one negative and one positive sum-of-products. The point here is that the sums-of-products are symbolic, so in general we will not be able to tell at sight the sign of the expressions $\Sigma_{i=1}^{n} a_{i,j} \cdot k_i$. We must then consider all the possible cases for these signs, so the extended polyhedron $\overrightarrow{P}$ has at most 3^m different forms, following the possible signs of the m expressions $\Sigma_{i=1}^{n} a_{i,j} \cdot k_i$.

For the example of figure 1, vertex L_2, the extension of the polyhedron $P = 0 \leq a_1 \leq 30 \wedge 0 \leq a_2 \leq 40 \wedge 40 \leq t \wedge a_1 - K \cdot t = 0$ by $\dot{a_1} = K, \dot{a_2} = 1 - K, \dot{t} = 1$ would give three cases, depending on the sign of expression $1 - K$. For instance, when $1 - K > 0$, the extended polyhedron (as automatically generated by our verification tool described in section 5) is:

$$\overrightarrow{P_1} = [0 \leq a_1 \wedge 0 \leq a_2 \wedge 40 \leq t \wedge 0 = a_1 - K \cdot t \wedge 30 \cdot K - 30 \leq (K-1) \cdot a_1 + K \cdot a_2 \wedge -30 \leq K \cdot t - a_1 \wedge 40 \cdot K - 40 = a_2 + (K-1) \cdot t \wedge 0 \leq (1-K) \cdot a_1 - K \cdot a_2 \wedge -40 \cdot K \leq a_1 - K \cdot t \wedge 40 \cdot K - 30 \leq K \cdot t - a_1 \wedge 0 \leq (1-K) \cdot t - a_2].$$

The extended polyhedra have different forms when $1 - K = 0$ and $1 - K < 0$. In practice, we do not have to generate all 3^m cases, since most of them lead to unsatisfiable conditions on the parameters (for instance, in the above example, the cases $1 - K = 0$ and $1 - K < 0$ are eliminated from start since we are looking for parameters in the interval $]0, 1[$ cf. section 3). We will come back to this point in section 5.

Note also that the parametric slopes have become coefficients in the inequations of the extended polyhedron e.g. $40 \cdot K - 30 \leq K \cdot t - a_1$.

4.2 Restriction

Restriction is: given a parametric polyhedron $P = \bigwedge_{j=1}^{m} (\Sigma_{i=1}^{n} a_{i,j} \cdot x_i \succ b_j)$ (remember that $a_{i,j}, b_j$ are constants or symbolic expressions on the parameters) find the possible values of parameters such that P is non-empty.

For the parametric polyhedron $\overrightarrow{P_1}$ obtained in the previous step, we compute its intersection with the guard $a_2 = 40$ and the invariant $t \leq 100 \wedge a_1 \leq 30 \wedge a_2 \leq 40$, and find the conditions on K such that this intersection is not empty. We obtain several cases from which only one is satisfiable (with respect to the values of K):

- Condition: $0 < K \wedge K < 1 \wedge (K-1) \cdot K < 0 \wedge K^2 \cdot (K-1) < 0$
- Intersection: the same as for extension plus three additional inequations ($\overrightarrow{P_1} \wedge t \leq 100 \wedge a_1 \leq 30 \wedge a_2 = 40$).

The general case can be treated as follows: the polyhedron P is non-empty iff the expression $\exists x_1 . \exists x_2 \ldots \exists x_n . P$ is 'true'. The formal elimination of all variables in the previous expression generates a symbolic condition on the parameters, *that precisely constitutes the condition for P to be non-empty.* Variables $x_1, .., x_n$ can be eliminated one by one by successively applying the Fourier-Motzkin elimination algorithm (see for instance chapter 1 of [Zie95]) that we now describe.

The Fourier-Motzkin elimination algorithm. This algorithm computes, given a system of linear inequations $P = \bigwedge_{j=1}^{m} (\sum_{i=1}^{n} a_{i,j} x_i \succ b_j)$, the system obtained by eliminating a variable say x_k : $P \updownarrow_k = \exists x_k . \bigwedge_{j=1}^{m} (\sum_{i=1}^{n} a_{i,j} \cdot x_i \succ b_j)$. The idea is to consider the possible signs of the coefficients $a_{k,j}$: as in the case of extension, eliminating variable x_k leads to at most 3^m possible forms of the result, depending on the signs of the m coefficients $a_{k,j}$. For a given combination of signs, denote $J_>$ (respectively, $J_=$, $J_<$) the subsets of indices of $\{1, \ldots, m\}$ such that $a_{k,j} > 0$ (respectively $= 0$, < 0). Then, $P \updownarrow_k$ is obtained by keeping the inequations indexed by $J_=$, by eliminating the inequations indexed by $J_<$ and $J_>$ (i.e. keep only the inequations where x_k does not occur), and by linearly combining pairs of inequations (one indexed by some $j_> \in J_>$, the other indexed by some $j_< \in J_<$) to eliminate variable x_k. For all $j_> \in J_>$ and $j_< \in J_<$, generate the following linear combination: $a_{k,j_>} [\sum_{i=1}^{n} a_{i,j_<} x_i \succ b_{j_<}] - a_{k,j_<} [\sum_{i=1}^{n} a_{i,j_>} x_i \succ b_{j_>}]$ which is equivalent to $\sum_{i=1}^{n} [a_{k,j_>} a_{i,j_<} - a_{k,j_<} a_{i,j_>}] x_i \succ [a_{k,j_>} b_{j_<} - a_{k,j_<} b_{j_>}]$. In this last inequation the coefficient of x_k is 0 so variable x_k has been eliminated.

4.3 Projection

Projection on the $x_k = 0$ plane means: given a parametric polyhedron $P = \bigwedge_{j=1}^{m} (\Sigma_{i=1}^{n} a_{i,j} \cdot x_i \succ b_j)$, find the parametric polyhedron $P|_{x_k=0}$ obtained by projecting P on the plane $x_k = 0$. Consider the polyhedron $P = \overrightarrow{P_1} \cap \{t \leq 100 \wedge a_1 \leq 30 \wedge a_2 = 40\}$ of the previous example. The projection of P on $a_2 = 0$ for conditions $0 < K \wedge K < 1 \wedge (K-1) \cdot K < 0 \wedge K^2 \cdot (K-1) < 0$ is

$$\begin{aligned}
P|_{a_2=0} = & 0 \leq a_1 \wedge 0 = a_2 \wedge a_1 \leq 30 \wedge 40 \leq t \wedge a_1 - K \cdot t = 0 \wedge \\
& t \leq 100 \wedge -40 \cdot K \leq a_1 - K \cdot t \wedge \\
& -30 \leq K \cdot t - a_1 \wedge 40 \cdot K - 30 \leq K \cdot t - a_1 \wedge \\
& 30 \cdot K - 30 \leq (K-1) \cdot a_1 - (K-1) \cdot K \cdot t \wedge \\
& -10 \cdot K - 30 \leq (K-1) \cdot a_1 \wedge \\
& 40 \cdot (K-1) \cdot K \leq (1-K) \cdot a_1 + (K-1) \cdot K \cdot t \wedge \\
& 30 \cdot K - 30 \leq (K-1) \cdot a_1 \wedge \\
& 70 \cdot K - 40 \cdot K^2 - 30 \leq (K-1) \cdot a_1 - (K-1) \cdot K \cdot t \wedge \\
& 0 \leq (1-K) \cdot t \wedge 40 - 40 \cdot K \leq (1-K) \cdot t \wedge \\
& 0 \leq (1-K) \cdot a_1 \wedge 40 \leq (1-K) \cdot t \wedge \\
& 40 \cdot K - 40 \leq (K-1) \cdot t \wedge 40 \cdot K \leq (1-K) \cdot a_1 \wedge \\
& 80 - 40 \cdot K \leq (1-K) \cdot t \wedge 40 \cdot K - 80 \leq (K-1) \cdot t
\end{aligned}$$

In general, to obtain the projected parametric polyhedron we use the Fourier-Motzkin elimination algorithm and the identity $P|_{x_k=0} = \exists x_k . P \wedge (x_k = 0)$.

4.4 The operations at work

To obtain the conditions on the parameters for a reachability formula to be true, combine the three operations as follows. First, choose a vertex path that links a vertex from the initial region to a vertex from the final region (as defined by the reachability formula). Then, iterate the three operations on that vertex path, to generate a *tree* whose nodes are pairs (vertex, parametric polyhedron), and whose edges are labeled by symbolic conditions on the parameters.

Starting from the pair (initial vertex, polyhedron defined by the initial values of variables) as the root, apply the *extension* procedure to the initial polyhedron to generate all the possible extended polyhedra, and associate a node (successor of the root) to each one. The branch leading to a node is labeled with the condition under which the node's extended polyhedron was obtained. Likewise, for each new node, generate its successors by applying the *restriction* procedure, and label the new branches correspondingly. For the lastly obtained successors, continue with the *projection* procedure. Iterate the sequence of procedures in this order until the final vertex is reached, and terminate by a restriction to intersect the final region.

At this point, any sequence of branches of the constructed tree, from the root to a leaf, defines a sufficient condition on the parameters, for the final region to be reachable from the initial one (it is the conjunction of the conditions on all branches). The *disjunction* of these sufficient conditions, for all the sequences of branches in the tree (from the root to a leaf), constitute the necessary and sufficient condition for the final region to be reachable from the initial one, just by the particular vertex path *in the automaton* that was initially chosen.

But, in general, there are an infinity of vertex paths in the automaton, from a formula's initial vertex to a final one; this means that in order to generate the necessary and sufficient conditions on the parameters for the reachability to hold, we might need to iterate the above operations on an infinite number of vertex paths, making our problem undecidable. However, we have noted in [BBRR97] that, for a restricted class of SPHA and formulas, the problem remains decidable: these are *uniformly low-bounded SPHA* and *time-bounded reachability formulas*.

Uniformly low-bounded SPHA are characterized by the fact that any cyclic run has a duration at least equal to some strictly positive ϵ, which does not depend on the run or the parameter values. For instance, the 2 cycles of the automaton in figure 1: $L_1 - L_2 - L_3 - L_5 - L_1$ and $L_1 - L_2 - L_4 - L_5 - L_1$, always have a duration 100 whatever the parameters, so this SPHA falls into the uniformly low-bounded case. Now for such SPHA and for time-bounded reachability formulas, it is necessary to propagate extension, restriction, and projection, only on a finite number of vertex paths [BBRR97].

Furthermore, some conditions on the parameters might be unsatisfiable. Therefore, every time a condition is generated it is important to test if it is satisfiable; this will prevent us from analyzing branches that would lead us nowhere. As it will be described in section 5, there exist automatic means to do it.

4.5 Solving our example problem

Consider the above automaton and the time-bounded reachability formula $\varphi = \{(L_1 \wedge a_1 = 0 \wedge a_2 = 0 \wedge t = 0) \Rightarrow \neg[(\neg L_5)\exists\mathcal{U}_{=100}(\neg L_5)]\}$. This means that, starting from the initial state, vertex L_5 is inevitably reached within 100 time units. Alternatively, this can be written as: $\neg\{(L_1 \wedge a_1 = 0 \wedge a_2 = 0 \wedge t = 0) \wedge [(L_1 \vee L_2 \vee L_3 \vee L_4)\exists\mathcal{U}_{=100}(L_1 \vee L_2 \vee L_3 \vee L_4)]\}$ meaning that, starting from the initial region $(L_1 \wedge a_1 = 0 \wedge a_2 = 0 \wedge t = 0)$, it is not possible to stay within vertices L_1, L_2, L_3 and L_4 for 100 time units.

In order to solve our initial parametric problem, we shall find values for K such that (α): *it is possible to stay vertices L_1, L_2, L_3 and L_4 for 100 time units*, and then take the complement for K (in $]0,1[$) .

To solve (α) we have to iterate the extension, restriction and projection operations on two vertex paths ($L_1 - L_2 - L_3$ and $L_1 - L_2 - L_4$). At each vertex L_i ($i \in \{1,2,3,4\}$), we find a set $\mathcal{P}_i$ of parametric polyhedra with their associated conditions on K; they constitute the result of propagating the initial region up to that vertex. In order to see if it is possible to be at vertex L_i at time $= 100$, we *restrict* each polyhedron in $\mathcal{P}_i$ with $t = 100$, that is, we find the conditions on parameter K such that their intersection with $t = 100$ is non-empty.

For vertices L_1 and L_2, these final conditions are unsatisfiable. For vertex L_4, the condition is $K \leq 0.3$. For vertex L_3, it is $K \in [1/3, 3/4]$. The complement of these conditions in interval $]0,1[$ is: $K \in]0.3, 1/3[\cup]3/4, 1[$. But for $K > 3/4$, the system is so-called *Zeno* (cf. section 4.6). Thus, the answer to our parametric problem is: $K \in]0.3, 1/3[$.

4.6 Non-Zenoness and backwards reachability

Consider again the SPHA of figure 1. For some values of the parameter $K \in]0,1[$, the automaton is affected by so-called *Zeno* behaviours [HNSY94]. We have seen an example of such behaviour in section 2: if $K > 3/4$, the run starting from initial state $L_1 \wedge a_1 = 0 \wedge a_2 = 0 \wedge t = 0$ is unable to leave vertex L_1 but it is also unable to stay in L_1 forever, since L_1's invariant eventually becomes false (after at most 40 time units). The system is so-called *Zeno* [HNSY94], meaning that *values $K > 3/4$ are intrinsically bad* for the system, whatever further properties we might want it to satisfy (e.g. tasks terminating within 100 time units). So a new problem of parametric analysis is to find the possible values of parameters of a SPHA, for the SPHA to be non-Zeno. To solve this problem, we use an existing technique [HNSY94]: a hybrid automaton is non-Zeno iff from any state of the automaton, it is possible to have a run of duration 1. In our parametric context, checking non-Zenoness means *finding the possible values of the parameters, such that from any state of the automaton, it is possible have a run of duration 1.*

Parametric backward analysis. The existence of a run of duration 1 from each state cannot be checked directly by the *forward* parametric analysis as presented in section 4, because one would have to perform the analysis starting from all the (continuously infinite number) states of the SPHA. Instead, the analysis should proceed *backwards*. For example, consider the SPHA in figure 1; to measure the global time, we enrich the SPHA with a new variable z behaving like a *clock* (constant slope 1 at all vertices), which is never reset[1]. Let us start indicating how to compute, for instance, the states from which it is possible to reach vertex L_2 in 1 time unit. For this, we first consider all the states of the automaton at vertex L_2 and at instant 1, given by L_2's invariant: $P_0 = t \leq 100 \wedge a_1 \leq 30 \wedge a_2 \leq 40 \wedge z = 1$. Then, we compute the set of states P_1 from which it is possible to reach some state in P_0 by *continuously remaining in vertex* L_2: this can be done by the *backwards extension* of P_0. Next, we compute the states P_2 from which it is possible to reach some state in P_1 by some *discrete edge crossing* i.e. by crossing the edge from L_1 to L_2: this is done by *backwards projection* on the edges's *reset* ($a_2 := 0$) followed by *restriction* to the edges's *guard* ($a_2 = 40$).

Restriction has been defined in section 4.2, and backwards extension/projection are similar to the restriction/projection described in sections 4.1 and 4.3. For instance, backwards extension of polyhedron $P_0 = t \leq 100 \wedge a_1 \leq 30$ $\wedge a_2 \leq 40 \wedge z = 1$ in direction $(\dot{a_1}, \dot{a_2}, \dot{t}, \dot{z}) = (K, 1-K, 1, 1)$ coincides with the forward extension of the polyhedron in the *opposite* direction $(-K, K-1, -1, -1)$; and backwards projection of polyhedron P_1 following the reset ($a_2 := 0$) is just $\exists a_2.P_1$ i.e. the Fourier-Motzkin elimination defined in section 4.2.

These operations should be iterated backwards on all the vertex paths of the SPHA, and at each step i, by restricting the obtained set of parametric polyhedra $\mathcal{P}_i$ with condition $z = 0$, we obtain states from which it is possible to have a run of duration 1. The iteration should continue until for the lastly obtained set of states, *the restriction to* $z = 0$ *is empty.* This last condition becomes true after a finite number of iterations, if SPHA has uniformly low-bounded cycles [BBRR97]. For example, in the hybrid automaton of figure 1, the above operations should be iterated only once on each cycle, because each cycle is guaranteed to last more than 1 time unit.

Thus, after a finite number of steps, the algorithm terminates, and one obtains the whole set of states $\mathcal{P}$ from which it is possible to have a run of duration 1, under the form of a set of parametric polyhedra. By imposing the condition that $\mathcal{P}$ contains *all* the states of the automaton, on obtains the necessary and sufficient condition on the parameters, for the SPHA to be non-Zeno.

5 The tool

We have implemented the *extension, restriction* and *projection* procedures in MAPLE V, a tool for symbolic computation. As we have seen, these operations generate conditions on the parameters that, in general, take the form of systems of non-linear inequalities whose unkowns are the variables of the hybrid automaton and the parameters. Such systems of inequalities are not solvable by the classical numerical methods but, instead, *propagation* methods [Sac87, Bro81]

[1] in section 4.5, we used directly t as a clock to measure global time, since it was never reset on the paths that interested us.

must be used to calculate a "rectangular envelope" of the solution. This envelope defines upper and lower bounds on the unknowns and its meaning is as follows: (α) if the envelope is empty then the system of inequalities has no solution, and (β) if the envelope is non-empty and if the system of inequalities has a solution, then the envelope covers the solution. Note that, as we are interested in those values of parameters that make an unsafe state reachable, in order to discard them, an over approximation is always valid (as long as its complement is not empty). We use in our prototype the propagation features of Prolog IV.

Furthermore, we have implemented the following optimizations:

- **Consistent case generation.** Considering all possible combinations of signs of sum-of-products, in the extension procedure, and of coeficients, in Fourier-Motzkin, may lead to generation of unsatisfiable cases as $K \wedge -K \wedge K \neq 0$. The prototype is aware of this, and does not generate unsatisfiable cases.
- **On the fly branch pruning.** The conditions associated with a case may be unsatisfiable in conjunction with the corresponding parametric polyhedron, leading to a useless branch in the tree. The tool prunes all such branches.
- **Alternative restriction procedure.** The aim of the restriction procedure is to generate conditions on the parameters for a given parametric polyhedron to be non-empty. This is a very expensive procedure whose result can be approximated as follows: approximate the conditions on the parameters by the envelope obtained *by propagating* the system of inequalities defined by the parametric polyhedron.

Table 1 shows the execution times for the example of section 4.5 measured on a Sun SPARCclassic. Times are in seconds.

Path	Tree generation (Maple)	Branch pruning (Prolog)	Envelope calculation (Prolog)	Total
$L_1 - L_2 - L_3$	3.750	0.350	0.060	4.16
$L_1 - L_2 - L_4$	3.333	0.380	0.060	3.773

Table 1. Execution times

The maximum number of cases generated is 9. This extremely low number of cases and the low executuion times are due to the application of the three optimizations described above.

6 Conclusion

In this paper we have presented an example of system analysis using the slope-parametric hybrid automata presented in [BBRR97]. The method focuses on computing slopes of variables (rather than delays) for some safety requirement to be respected, and it may be seen as an extension of the polyhedra-based symbolic analysis [ACH$^+$95] of hybrid automata. The main operation is the

Fourier-Motzkin's algorithm to deal with parametric polyhedra, which imposes, in theory, to consider a large number of cases. In practice, however, many of these cases are either redundant or unsatisfiable and can be discarded, making the problem computationally tractable. We have presented a prototype verification tool which fully automates the analysis, and shown its applicability to a simple example.

Acknowledgments. We thank Gérard Verfaillie for suggesting us to use Prolog IV, and Olivier Roux and Frédéric Boniol for following this work.

References

ACH+95. 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. *TCS*, 138:3–34, 1995.

AHH93. R. Alur, T. A. Henzinger, and P-H. Ho. Automatic symbolic verification of embedded systems. In *Proc. IEEE RTSS'93*, pages 2–11, 1993.

AHV93. R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proc. ACM STOC'93*, pages 592–601, 1993.

BBRR97. F. Boniol, A. Burgueño, O. Roux, and V. Rusu. Analysis of slope-parametric hybrid automata. In *Proc. HART'97*, volume 1201 of *LNCS*, pages 75–80, 1997.

BGK+96. J. Bengtsson, D. Griffioen, K. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using UPPAAL. In *Proc. CAV'96*, volume 1102 of *LNCS*, 1996.

Bro81. R. A. Brooks. Symbolic reasoning among 3-D models and 2-D images. *Art. Int.*, 17:285–348, 1981.

CABN97. W. Chan, R. Anderson, P. Beame, and D. Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In *Proc. CAV'97*, 1997.

CY91. C. Courcoubetis and M. Yannakakis. Minimum and maximun delay problems in real-time systems. In *Proc. CAV'91*, volume 575 of *LNCS*, pages 399–409, 1991.

DY95. C. Daws and S. Yovine. Two examples of verification of multirate timed automata with Kronos. In *Proc. IEEE RTSS'95*, 1995.

HH94. T. A. Henzinger and P.-H. Ho. HyTech: the Cornell HYbrid TECHnology tool. In *Hybrid Systems II*, volume 999 of *LNCS*, pages 265–294, 1994.

HLM97. M. Heymann, F. Lin, and G. Meyer. Control synthesis for a class of hybrid systems subject to configuration-based constraints. In *Proc. HART'97*, volume 1201 of *LNCS*, pages 376–390, 1997.

HNSY94. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Inf. and Comp.*, 111(2):193–244, 1994.

KS97. D. Kapur and R. K. Shyamasundar. Synthesizing controllers for hybrid systems. In *Proc. HART'97*, volume 1201 of *LNCS*, pages 361–375, 1997.

Sac87. E. Sacks. Hierarchical reasoning about inequalities. In *Proc. AAAI'87*, volume 2, pages 649–654, 1987.

Wan96. F. Wang. Parametric timing analysis for real-time systems. *Inf. and Comp.*, 130(2):131–150, 1996.

Zie95. G. M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer-Verlag, 1995.