Int J Softw Tools Technol Transfer (2008) 10:263–279
DOI 10.1007/s10009-007-0062-x

REGULAR CONTRIBUTION

PHAVer: algorithmic verification of hybrid systems past HyTech

Goran Frehse

Published online: 8 January 2008

Abstract In 1995, HyTech broke new ground as a potentially powerful tool for verifying hybrid systems. But due to practical and systematic limitations it is only applicable to relatively simple systems. We address the main problems of HyTech with PHAVer, a new tool for the exact verification of safety properties of hybrid systems with piecewise constant bounds on the derivatives, so-called linear hybrid automata. Affine dynamics are handled by on-the-fly overapproximation and partitioning of the state space based on user-provided constraints and the dynamics of the system. PHAVer features exact arithmetic in a robust implementation that, based on the Parma Polyhedra Library, supports arbitrarily large numbers. To force termination and manage the complexity of the polyhedral computations, we propose methods to conservatively limit the number of bits and constraints of polyhedra. Experimental results for a navigation benchmark and a tunnel diode circuit demonstrate the effectiveness of the approach.

Keywords Hybrid systems · Verification · Tools · Polyhedra

1 Introduction

Systems with interacting discrete as well as continuous dynamics, so-called hybrid systems, are notoriously complex to analyze. Their algorithmic verification remains a challenging problem, from a theoretical point of view because of decidability problems, and from the implementation side because the slightest numerical errors can void the results. We present PHAVer (Polyhedral Hybrid Automaton Verifier), a verification tool that aims at overcoming the limitations of predecessors such as as HyTech [1]. Although PHAVer uses in principle the same algorithm as HyTech, we not merely present a new implementation of known algorithms, but propose heuristics to overcome inherent and fundamental problems of reachability computations, namely excessive complexity, lack of termination and loss of accuracy due to overapproximations. As a result, we are able to analyze systems previously beyond the reach of verification tools.

In this paper we consider the reachability problem, which consists of computing the set of reachable states of a system or a conservative overapproximation thereof. While reachability is a relatively simple property, more complex safety or bounded liveness properties can be formulated as reachability problems by adding monitoring components to the system [2]. Computations in PHAVer are based on linear hybrid automata (LHA) [3], which are defined by linear predicates and piecewise constant, convex, bounds on the derivatives. They stand out from other classes of hybrid automata because one knows how to compute the successor states of discrete and timed transitions exactly [4]. PHAVer is based on such a reachability algorithm for LHA, but can also compute conservative overapproximations for hybrid automata with more complex dynamics thanks to an on-the-fly implementation of phase-portrait approximation [5].

Our reachability computations face three fundamental problems: excessive complexity, slow convergence, and insufficient accuracy when overapproximations are used. In addition to the state explosion problem, well-known from the domain of discrete systems, one has to deal with expensive

A preliminary version of this paper appeared in the *Proceedings of Hybrid Systems: Computation and Control (HSCC 2005), Lecture Notes in Computer Science 3414, Springer-Verlag, 2005, pp. 258–273.*

G. Frehse (✉)
VERIMAG, 2, ave de Vignate, 38610 Gieres, France
e-mail: goran.frehse@imag.fr
URL: http://www-verimag.imag.fr/~frehse

polyhedral computations whose complexity increases, often exponentially, not only with the number of variables in the system but also in each iteration of the analysis. Inevitably, one has to resort to overapproximations for all but the most simple problems. Termination is not guaranteed in general, but it may be induced by overapproximation such as limiting the number of bits per constraint, which we will introduce later. However, *convergence* might be prohibitively slow, and overapproximations may have a positive or negative effect that is hard to predict. Overapproximations forcibly decrease the *accuracy* of the results, so the obtained set of reachable states might contain forbidden states that are not actually reachable by the system.

In applications, complexity and convergence problems manifest themselves in terms of computation time, memory consumption and degradation of accuracy, and the challenge is to find an acceptable trade-off between them. We propose the following heuristic to enable such a trade-off: The complexity of polyhedral computations is reduced by imposing a limit on the number of bits and constraints in polyhedral representations, overapproximating them in a manner that is guaranteed to be conservative. These overapproximations immediately imply termination, since only a finite number of constraints exist for a given limit. However, they may lead to severe convergence problems, which we attempt to counter by applying them only above a certain threshold. Experimental results indicate that the proposed overapproximations are sufficiently accurate in practice.

In addition to the reachability algorithm, PHAVer includes a separate engine for computing simulation relations between hybrid automata. It can be used to verify equivalence or refinement between different models, and in assume-guarantee-style proofs. The reader is referred to [6,7] for a description of the approach and some experimental results. Using PHAVer, we successfully verified analog and mixed-signal circuits that were previously beyond the reach of verification tools, see [8,9]. PHAVer has been used by others to verify safety properties of hybrid Chi specifications [10], and to verify a type of stability in hybrid systems [11]. It was also used in abstraction-refinement schemes based on rectangular automata [12] and based on LHA [9]. A performance comparison between PHAVer and HSOLVER in [13] turned out in PHAVer's favor.

Hybrid automata. Our hybrid automata are based on the classic model in [14]. We added a distinction between input and output variables, comparability and compatability definitions from [15]. In addition, we adopted from [3] that the set of variables may be different for each automaton in a composition, and there is a set of initial states.

From HyTech to PHAVer. In 1995, Henzinger et al. presented the tool HyTech [1]. It features a powerful input language permitting to specify fixpoint algorithms based on post- and pre-operators, but suffers from a major flaw: It uses exact arithmetic with a data structure of limited digits, which can quickly lead to overflow errors. It was successfully used to analyze a number of relatively small examples [16–21], but the overflow problem prohibits any application to more complex systems. Nonetheless, the numerous valuable experiences with HyTech have spawned important suggestions for improvement [17], most of which we incorporated in PHAVer. The basic operations in PHAVer, i.e., discrete and timed post-operators based on polyhedral computations, are identical to those in HyTech, and use exact arithmetic with unlimited precision. Our implementation is immune to overflow errors thanks to its use of the Parma Polyhedra Library (PPL) [22] and the GNU Multiple Precision Arithmetic Library (GMP) [23], which can handle computations with hundreds of thousands of bits on a standard PC. The first HyTech prototype was implemented in Mathematica and did not have any numerical restrictions, but it was also 50–1000 times slower than the later version written in C++ [24].

Overapproximating dynamics. We use an on-the-fly overapproximation that is a variation of the phase-portrait approximation from [5]. It differs in that the invariants of the partitioned locations do not have an open cover because we split along hyperplanes and the invariants only overlap on the plane, which is valid for affine dynamics [12]. Similar variations of partitioning the state space are widely used, e.g., in [25,26], although we're not aware of any work using the angular spread of the derivatives. Earlier attempts to improve over HyTech include the use of interval arithmetic [27], which can quickly lead to prohibitively large overapproximations. Recently, interval arithmetic was combined with abstraction refinement in a tool called HSOLVER, which can deal conservatively with nonlinear dynamics [28]. For a survey of verification tools for hybrid automata, see [29].

Managing complexity. In the context of timed systems, zones are *rounded* in [30] by dropping constraints or rounding to a constant lower bound. An algorithm specialized on rectangular automata was proposed in [31] and implemented based on the HyTech engine. It is able to use a limited number of bits through component-wise conservative rounding of the coefficients. However, the rectangular overapproximation can become too inaccurate. An improvement was proposed in [32] by allowing arbitrary convex polyhedra. It also incorporates a strategy to reduce the number of bits by component-wise overapproximation, but is based on a vertice representation of polyhedra and its complexity is exponential in the number of variables. For the simplification of polyhedra it has been suggested to use bounding boxes or oriented rectangular hulls [33]. Instead, we propose to simply drop

the least significant of the constraints, as this seems a good compromise in terms of accuracy and speed.

In the next section we introduce the hybrid automaton model used in PHAVer. In Sect. 3 we present the reachability analysis algorithm and its on-the-fly overapproximation of affine dynamics. Experimental results are provided for a navigation benchmark. Methods to manage the complexity of polyhedra by limiting the bits and constraints are proposed in Sect. 4, and illustrated with a tunnel diode circuit. We draw some conclusions in Sect. 5.

2 Hybrid automata with controlled variables

Hybrid automata [34] are widely recognized as an intuitive and expressive modeling paradigm for hybrid systems. Out of several different definitions present in literature we choose a slight variation of the hybrid automata in [14] because they admit compositional reasoning. To reason about equivalence between automata in a practical manner, we enrich them with a distinction between input and output variables, as in [15], although we forego such a distinction between synchronization labels for the sake of simplicity.

Preliminaries. Given a set $X = \{x_1, \ldots, x_n\}$ of variables, a *valuation* is a function $v : X \to \mathbb{R}$. We use $\dot{X}$ to denote the set $\{\dot{x}_1, \ldots, \dot{x}_n\}$ of dotted variables, and X' to denote the set $\{x'_1, \ldots, x'_n\}$ of primed variables. Let $V(X)$ denote the set of valuations over X. The *projection* of v is $v{\downarrow_{\bar{X}}} = \{x \to v(x) | x \in \bar{X}\}$. The *embedding* of a set $U \subseteq V(X)$ into variables $\bar{X} \supseteq X$ is $U|^{\bar{X}} = \{v \in V(\bar{X}) | v{\downarrow_X} \in U\}$. When a valuation u over X and a valuation v over $\bar{X}$ *agree*, i.e., $u{\downarrow_{X\cap\bar{X}}} = v{\downarrow_{X\cap\bar{X}}}$, we use $u \sqcup v$ to denote the valuation w defined by $w{\downarrow_X} = u$ and $w{\downarrow_{\bar{X}}} = v$. Arithmetic operations on valuations are defined in the straightforward way. An *activity* over X is a function $f : \mathbb{R}^{\geq 0} \to V(X)$. Let $Acts(X)$ denote the set of activities over X. The *derivative* $\dot{f}$ of an activity f is an activity over $\dot{X}$, defined analogously to the derivative in $\mathbb{R}^n$. The extension of operators from valuations to activities is done pointwise. Let $const_X(Y) = \{(v, v') | v, v' \in V(X), v{\downarrow_Y} = v'{\downarrow_Y}\}$.

Definition 1 A *hybrid automaton* $H = (Loc, (X, O, C), Lab, Edg, Flow, Inv, Init)$ consists of the following:

- A set $X = \{x_1, \ldots, x_n\}$ of continuous, real-valued *variables*. They are divided into *controlled* variables C and *input* variables $I = X \setminus C$, and a subset O of C is designated as the *output* variables.
- A set *Loc* of *locations*. A *state* is a pair (l, v) of a location l and a valuation v, which attributes a value to each of the variables.
- A set *Lab* of *synchronization labels* including the *stutter label* τ.
- A set *Edg* of *transitions* that describe instantaneous changes of location, in the course of which variables may change their values. Each transition $(l, a, \mu, l') \in Edg$ has a *source location* l, a *target location* l'. It is associated with a *synchronization label* a and with a *jump relation* μ that relates values of variables before the transition to values they take after the transition. The projection of the jump relation to the variables before the transition describes for which variable values the transition is enabled; this is often referred to as a *guard*. Each location l has a self-loop *stutter transition* $(l, \tau, const_X(C), l)$ that lets the input variables change arbitrarily within the invariant, while the controlled variables remain constant.
- A mapping *Flow* attributes to each location a set of valuations over the variables and their derivatives, which determines how variables can change over time.
- A set of states *Inv* called *invariant*. All behavior is constrained to the invariant at all times. In this paper we consider only invariants that are convex in each location.
- A set *Ini* of *initial states*, contained in the invariants, from which all behavior of the automaton originates.

A *linear constraint* over a set of variables X has the form

$$\sum_i \alpha_i x_i + \beta \bowtie 0, \tag{1}$$

where α_i and β are integer constants and $\bowtie$ is a sign $\bowtie$ is either $<$ or $\leq$. A *convex linear predicate* is a conjunction of linear constraints. We write $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ to denote derivatives, and using $X' = \{x'_1, \ldots, x'_n\}$ describe a relation $R \in X \times X$ with a predicate over $X \cup X'$. In a *Linear Hybrid Automaton* (LHA) [3], invariants and initial states are given by linear predicates over X, flow predicates by convex linear predicates over $\dot{X}$, and jump relations by linear predicates over $X \cup X'$. PHAVer can handle *affine hybrid automata*, which are defined like LHA except that flows may be given by linear constraints over $X \cup \dot{X}$. The use of inequalities in flow predicates allows us to model dynamics of the form $\dot{x} = Ax + b$ with the elements of A and b given by intervals.[1] Note that LHA can model discrete-time dynamics of the form $x_{k+1} = Ax_k + b$ by updating the variables in transitions with a jump relation $x'_i = \sum_j a_{ij} x_j + b_i$.

Complex systems can be constructed in a modular fashion using *parallel composition*. Our hybrid automata are *compositional* with respect to reachability properties [7,36], which means that a state that is not reachable in a given automaton

[1] In the literature, affine hybrid automata are sometimes also referred to as a *linear* hybrid automata, e.g., in [35], which may lead to confusion with LHA.

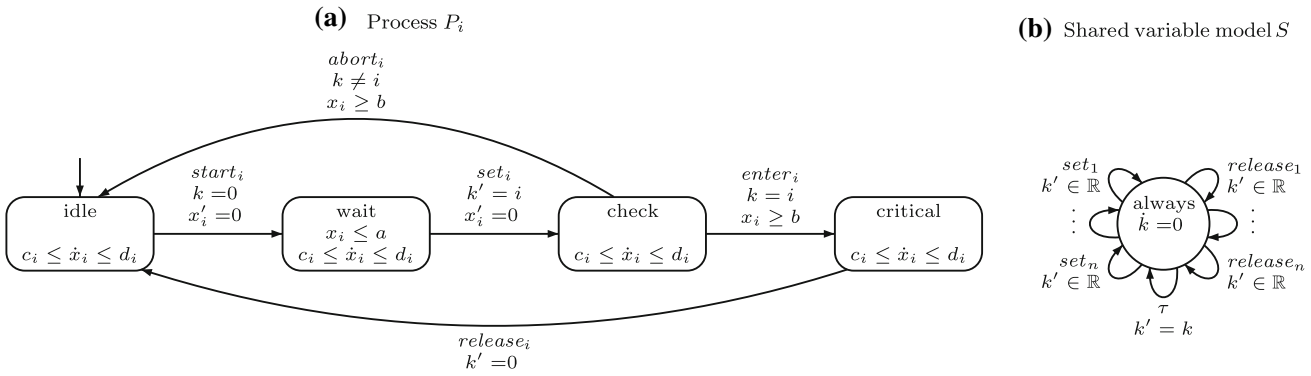

Fig. 1 Compositional model of timing based mutual-exclusion protocol in [4]

H will also not be reachable in a composition of H with any other automaton. If a system is modeled as collection of hybrid automata, then any non-reachability property of some subset of these automata will also hold for the entire collection. We use the standard definition of parallel composition:

Definition 2 (*Parallel composition*) [14] Hybrid automata H_1, H_2 are *compatible* if $C_1 \cap C_2 = \emptyset$, $X_1 \cap C_2 \subseteq O_2$ and $X_2 \cap C_1 \subseteq O_1$. The *parallel composition* of compatible hybrid automata H_1, H_2 is the hybrid automaton H with

- $Loc = Loc_1 \times Loc_2$,
- $X = X_1 \cup X_2$, $C = C_1 \cup C_2$, $O = O_1 \cup O_2$, $Lab = Lab_1 \cup Lab_2$
- $((l_1, l_2), a, \mu, (l_2', l_2')) \in Edg$ iff
  - $(l_1, a_1, \mu_1, l_1') \in Edg_1$ and $(l_2, a_2, \mu_2, l_2') \in Edg_2$
  - either $a = a_1 = a_2$, or $a = a_1 \notin Lab_2$ and $a_2 = \tau$, or $a_1 = \tau$ and $a = a_2 \notin Lab_1$,
  - $\mu = \{(v, v') | (v{\downarrow_{X_i}}, v'{\downarrow_{X_i}}) \in \mu_i\}$;
- $Flow(l_1, l_2) = Flow_1(l_1)|^{X \cup \dot{X}} \cap Flow_2(l_2)|^{X \cup \dot{X}}$;
- $Inv(l_1, l_2) = Inv_1(l_1)|^X \cap Inv_2(l_2)|^X$;
- $Init(l_1, l_2) = Init_1(l_1)|^X \cap Init_2(l_2)|^X$.

Example 1 A model of a timing based mutual-exclusion protocol, adapted from [4], is shown in Fig. 1. In every location l of P_i, there is a transition (l, τ, μ, l) with $\mu = \{(v, v') | v(x_i) = v'(x_i), v(k), v'(k) \in \mathbb{R}\}$ (omitted from the figure). Variables stay constant in transitions unless there is an explicit assignment shown. The system is considered *safe* if there are never two or more processes in the critical section at the same time. It is a compositional adaptation of the model given in [4], and parameterized to n processes with time constants c_i and d_i that represent the minimal, respectively maximal, skew of their clocks. The processes P_i have a controlled variable x_i to model their local clock and an input variable k that models a semaphore. A separate automaton S models write-access to k. S has k as a controlled variable and fixes its derivative to zero. It gives the processes access to k by synchronizing on transitions labeled with set_i or $release_i$. There are no restrictions on the jump relations of these transitions, so the processes can change k to an arbitrary value.

Semantics. The behavior of a hybrid automaton is based on two types of transitions: *Discrete* transitions are manifestations of the transitions in *Edg*, and change the location and variables instantaneously. *Timed* transitions describe the change of the variables over time.

Definition 3 (*Run, reachability*) For a given hybrid automaton H, there is a *discrete transition*

$$p \xrightarrow{a} p'$$

with *source and target states* p, p' and label $a \in Lab$ iff $p, p' \in Inv$ and there is a transition $loc(p) \xrightarrow{a,\mu}_H loc(p')$ with $(val(p), val(p')) \in \mu$. There is a *timed transition*

$$p \xrightarrow{\delta, f} p'$$

with $\delta \in \mathbb{R}^{\geq 0}$ and activity f over X iff $p, p' \in Inv$, f is differentiable, $f(0) = val(p)$, and $\delta = 0$ or $\forall t, 0 \leq t \leq \delta : f(t) \in Inv(loc(p))$, $f(t) \sqcup \dot{f}(t) \in Flow(l)$. A *run* of a hybrid automaton H is a finite or infinite sequence

$$\sigma = p_0 \xrightarrow{\delta_0, f_0, a_0} p_1 \xrightarrow{\delta_1, f_1, a_1} \cdots \xrightarrow{\delta_{n-1}, f_{n-1}, a_{n-1}} p_n$$

such that for all $i \geq 0$

$$p_i \xrightarrow{\delta_i, f_i} (loc(p_i), f_i(\delta_i)) \xrightarrow{a_i} p_{i+1}.$$

A state p' is *reachable* if there exists a run σ with $p_0 \in Init$ and $p_n = p'$.

3 Reachability analysis in PHAVer

We compute the set of reachable states as the fixpoint of two *Post*-operators, which yield the image states of discrete,

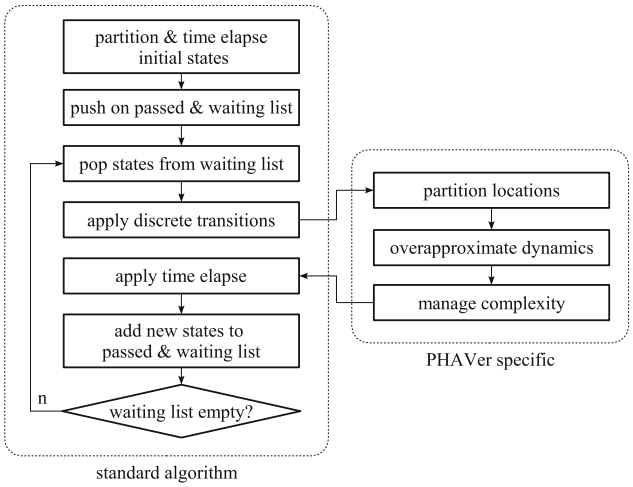

Fig. 2 Reachability algorithm in PHAVer

respectively timed, transitions. If the flow in a location is affine, it is overapproximated with LHA-dynamics, i.e., with constant bounds on the derivatives. The accuracy of this overapproximation depends on the size of the invariant, so locations are partitioned on the fly depending on several criteria. We give an overview of our algorithm, and then provide detailed descriptions of the operations involved.

The implementation in PHAVer processes a shared passed and waiting list as shown in Fig. 2. First, the locations of the initial states are partitioned, time elapse is applied, and they are put on the passed and waiting list. In the main loop, a set of states is taken from the waiting list, and the successor states of discrete transitions are computed. Then the locations of these successors are partitioned, and in each partition the dynamics are overapproximated with LHA-dynamics. The set of successors is subjected to complexity management, which may overapproximate it with a set of lower complexity. Time elapse is applied to this new set, and it is added to the passed list. The new states, i.e., those that were not previously on the passed list, are put onto the waiting list. The loop repeats until the waiting list is empty.

Remark 1 Note that to be sound, time elapse must be applied after any overapproximation, such as convex hull or complexity management, unless the overapproximation is guaranteed to be invariant with respect to time elapse.

The basic algorithm, shown on the left side of Fig. 2, is similar to the one used in HyTech, and summarized in the next section. Partitioning, overapproximation and complexity management are specific to PHAVer, and will be discussed in detail in the remainder of the paper.

3.1 Reachability of LHA

We summarize the standard reachability algorithm from [4,37] for LHA using polyhedra. To compute the set of reachable states, one repeatedly computes the successors of timed and discrete transitions. Given a set of states P, let

$$Post_t(P) = \left\{ p' \,\middle|\, \exists \delta, f, p \in P : p \xrightarrow{\delta, f} p' \right\}$$

be its timed successors and

$$Post_d(P) = \left\{ p' \,\middle|\, \exists a, p \in P : p \xrightarrow{a} p' \right\}$$

its discrete successors. The set of reachable states, denoted by *Reach*, is the smallest fixpoint of the sequence given by $P_0 = Init$ and

$$P_{k+1} = Post_d\left(Post_t(P_k)\right). \tag{2}$$

We recall the basic definitions and operations on polyhedra, and use them to define the *post* operators for LHA.

Definition 4 A *convex polyhedron* S is a set in $\mathbb{R}^n$ that can be represented as the conjunction of a finite number of linear constraints, i.e.,

$$S = \left\{ x \,\middle|\, \bigwedge_i a_i^T x \bowtie_i b_i \right\}, \tag{3}$$

where $a_i \in \mathbb{R}^n$, $b_i \in \mathbb{R}$ and $\bowtie_i \in \{<, \leq\}$. This is referred to as the *constraint representation* of the polyhedron. Strict inequalities $a_i^T x < b_i$ can be replaced with nonstrict inequalities $a_i^T x + \varepsilon \leq b_i$ by introducing an auxiliary variable ε that may take an arbitrary positive value [37]. In the following, we will therefore only consider nonstrict inequalities, which define closed polyhedra. Alternatively, a polyhedron can be described by a *generator representation* (V, R) as the convex hull of a finite set $V \subseteq \mathbb{R}^n$ of *vertices* and a finite set $R \subseteq \mathbb{R}^n$ of *rays*, i.e.,

$$S = \left\{ \sum_{v_i \in V} \lambda_i v_i + \sum_{r_i \in R} \mu_i r_i \,\middle|\, \lambda_i, \mu_i \geq 0, \sum_i \lambda_i = 1 \right\}. \tag{4}$$

One may switch between constraint and generator representations, although at a potentially exponential cost. Some operations are easier in constraint form, e.g., intersection, while others are easier in generator form, e.g., testing for emptiness. Therefore modern polyhedral libraries, such as the PPL, keep both represenations in what is referred to as the *double description method*. Projection and embedding operators are easily implemented in constraint form by removing elements or adding zeros to the vectors a_i.

Recall that in a discrete transition $(l, a, \mu, l') \in Edg$, the jump relation μ is a predicate over $X \cup X'$, where the primed variables specify the new values after the transition. The

discrete successors of a set of states P are

$$Post_d(P) = \bigcup_{(l,a,\mu,l')\in Edg} l' \times \left(\left(P(l)^{\uparrow X\cup X'} \cap \mu \right) \Big\downarrow_{X'\to X} \cap Inv(l') \right), \quad (5)$$

where $\downarrow_{X'\to X}$ denotes that we project onto the primed variables X', and rename the variables as unprimed. The timed successor operator is slightly more involved. For LHA with convex invariants there is an activity from one state in location l to another iff they are connected by a straight line whose derivative is in $Flow(l)$ [4]. This allows one to replace the passage of time by simple existential quantification as follows. Let $S \nearrow F$ be defined as

$$S \nearrow F = \left\{ v' \,\middle|\, \exists \delta \in \mathbb{R}^{\geq 0}, w \in F, v \in S : v' = v + w\delta \right\}. \quad (6)$$

This operation is easily implemented using the generator representation [37]. Let (V, R) be the generator representation of S, and (V', R') that of F. Then $S \nearrow F = (V, R \cup V' \cup R')$. For convex invariants, the timed successors are then given by

$$Post_t(P) = \bigcup_{l\in loc(P)} l \times (P(l) \nearrow Flow(l)) \cap Inv(l). \quad (7)$$

Example 2 Consider the mutual exclusion protocol from Example 1 for two processes, with $a = 3$, $c_1 = c_2 = 1$, $d_1 = 3$, $d_2 = 2$. We compute the time elapse for the set $S = \{x_1 = x_2 = 0\}$, with both processes in location *wait*. The flow in the location is given by $F = Flow((wait, wait)) = \{1 \leq \dot{x}_1 \leq 3 \wedge 1 \leq \dot{x}_2 \leq 2\}$, which translates in generator form to $F = (\{(1, 1), (3, 1), (3, 2), (1, 2)\}, \{\})$, see Fig. 3a. To apply the time elapse, the vertices of F in the derivative space are reinterpreted as rays in the state space, shown as dashed arrows in Fig. 3b. These rays are added to the generator representation of $S = (\{(0, 0)\}, \{\})$ and we obtain

$$S \nearrow F = (\{(0, 0)\}, \{(1, 1), (3, 1), (3, 2), (1, 2)\}).$$

Finally, one has to restrict the time elapse to the invariant $Inv((wait, wait)) = \{x_1 \leq 3, x_2 \leq 3\}$. The resulting set $Post_t((wait, wait) \times S) = (S \nearrow F) \cap Inv((wait, wait))$ is shown in Fig. 3b.

Remark 2 The time elapse operator implemented in the PPL (v0.9) computes the smallest convex set containing $S \nearrow F$. Strictly speaking, this is an overapproximation as the actual set may be nonconvex. Consider $S = \{x = 0 \wedge y = 0\}$ and $F = \{y > 0\}$. Then $S \nearrow F = \{(x = 0 \wedge y = 0) \vee y > 0\}$, while the PPL returns $S \nearrow_{PPL} F = \{y \geq 0\}$.

In the following example we compare the performance of HyTech and PHAVer. The results indicate that the overhead

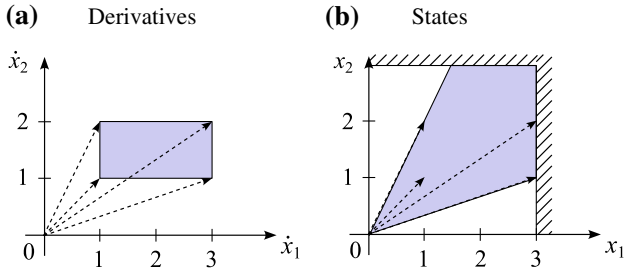

Fig. 3 Time elapse for LHA

introduced by using a multiprecision number representation is compensated by the efficiency of the PPL, and improvements such as the shared passed and waiting list. The comparison is limited to simple parameters and systems to avoid overflow problems with HyTech.

Example 3 Consider the mutual exclusion protocol from Example 1 for n processes, with $a = 1$, $b = 4$, $c_i = 1$, $d_i = 2$. The experimental results are shown for MEX1 with $n = 1$ to MEX5 with $n = 5$ in Table 1, as well as parametric versions MEXP1–4 in which the allowed range of a and b is computed. In these instances, PHAVer outperforms HyTech for nontrivial systems. The downside is its memory consumption, which is about twice that of HyTech. In the instance of a parametric analysis of an audio protocol AUDSP from [38], HyTech outperforms PHAVer. The experiments in this paper were carried out on a Pentium IV, 3.2 GHz with 1 GB RAM running Linux.

3.2 Overapproximation of dynamics

To deal with non-LHA dynamics, we apply a simple variant of *phase-portrait approximation* [5]. This method utilizes the fact that overapproximating any of the sets that define an automaton, in particular *Flow*, results in an overapproximation of its behavior. We can therefore apply our reachability algorithm to any hybrid automaton by overapproximating it with LHA-style sets, and obtain a conservative overapproximation of its reachable states. Because the accuracy of the overapproximation depends on the size of the invariant, the locations are adequatly partitioned before applying this transformation, as will be discussed in the next section.

Currently, this is implemented for affine dynamics, which have the form $M\dot{x} = Ax + b$, or more precisely a relaxed version thereof that may consist of inequalities. We describe two methods for overapproximating them with LHA-dynamics, i.e., for obtaining constant bounds on the derivatives in the form of linear constraints. In the following, we consider *Flow* to be given as *relaxed affine dynamics*, i.e., a conjunction of constraints

$$a_i^T \dot{x} + \hat{a}_i^T x \bowtie_i b_i, \quad (8)$$

Table 1 Tool performance, HyTech vs. PHAVer

Instance	HyTech			PHAVer			Reachable Set	
	Time (s)	Mem. (MB)	Iter.	Time (s)	Mem. (MB)	Iter.	Loc.	Poly.
MEX2	0.0	3.1	7	0.1	5.2	9	13	21
MEX3	0.7	6.0	17	1.0	10.3	15	39	139
MEX4	17.5	28.8	23	10.0	58.6	21	113	1169
MEX5	928.3	437.8	29	238.9	814.5	27	323	12001
MEX2P	0.1	4.5	12	0.2	5.9	9	16	37
MEX3P	10.5	27.4	23	2.9	17.9	19	64	564
MEX4P	2999.0	1660.6	34	114.0	443.1	29	256	12408
AUDSP	8.8	62.7	39	20.1	143.6	39	193	5849

where $a_i, \hat{a}_i \in \mathbb{Z}^n$, $b_i \in \mathbb{Z}$, $\bowtie_i \in \{<, \leq, =\}$, $i = 1, \ldots, m$.

Projection. Given a set of states S that constrains the possible values of x, we can obtain bounds on the derivatives by intersecting *Flow* with S and projecting onto the derivatives:

$$Flow_{pr}(S) = \left(Flow \cap S|^{X \cup \dot{X}}\right)\Big\downarrow_{\dot{X}}. \tag{9}$$

We may always choose $S = Inv$, but further refinement is possible, as discussed below. PHAVer's default time elapse operator is obtained by substituting (9) for the flow in (7):

$$Post_{t,pr}(P) = \bigcup_{l \in loc(P)} l \times (P(l) \nearrow Flow_{pr}(Inv(l)) \cap Inv(l). \tag{10}$$

When *Flow* is given as a conjunction of constraints of the form (8) and S is given by linear constraints, (9) involves only standard operations on polyhedra in the space of $X \cup \dot{X}$, and is therefore straightforward to implement.

Constraint-based. Sometimes, $Flow_{pr}(S)$ is very complex, e.g., when S is iteratively refined, as discussed below, or in abstraction-refinement schemes such as the one in [9]. If this is the case, the following solution may be advantageous. Assume that equalities in (8) are modeled using conjuncts of pairs of inequalities. We transform each constraint (8) into a linear constraint over $\dot{X}$ by finding a lower bound on $\hat{a}_i^T x$. If no such bound exists, we drop the constraint entirely. We obtain the same number of constraints as *Flow*, independent on how complex the invariant might be. On the downside, the correlations between the constraints are lost, which may result in gross overapproximations.

Given a set of states S that constrains the possible values of x, let

$$p/q = \inf_{x \in S} \hat{a}_i^T x, \quad p, q \in \mathbb{Z}. \tag{11}$$

If the infimum exists, the set of $\dot{x}$ that satisfy (8) is contained in the set defined by $a_i^T \dot{x} \bowtie_i b_i - p/q$. Multiplying both sides with q yields the LHA-style constraint

$$q a_i^T \dot{x} \bowtie_i q b_i - p. \tag{12}$$

Let $Flow_{con}(S)$ be obtained from $Flow(S)$ by replacing each constraint of the form (8) by its corresponding constraint (12) if the infimum in (11) exists, and otherwise removing the constraint.

A one-to-one correspondence of the original and the LHA-style constraints allows the user to specify the type of constraints he wishes to obtain on the dynamics. For example let the dynamics be given by $\dot{x} = f(x, y) \wedge \dot{y} = g(x, y)$, f and g being affine functions. Specifying the constraints in this form in PHAVer results in rectangular LHA-style constraints, i.e., of the form $\dot{x} \in [., .] \wedge \dot{y} \in [., .]$. Octagonal constraints, i.e., of the form $\dot{x} \in [., .] \wedge \dot{y} \in [., .] \wedge \dot{x} - \dot{y} \in [., .] \wedge \dot{x} + \dot{y} \in [., .]$ can be obtained by specifying $\dot{x} = f(x, y) \wedge \dot{y} = g(x, y) \wedge \dot{x} - \dot{y} = f(x, y) - g(x, y) \wedge \dot{x} + \dot{y} = f(x, y) + g(x, y)$. Octagonal constraints are useful because they preserve some of the correlation between variables, and thus may lead to drastically improved accuracy.

As the following example shows, the constraint based overapproximation can be very inaccurate, and does not even have to be tight, i.e., touch the original set.

Example 4 Let $Inv(l) = \{x_l \leq x \leq x_u\}$ and $Flow(l) = \{\dot{x} \leq ax + b_u, \dot{x} \leq -ax + b, \dot{x} \geq ax + b_l\}$ as shown in Fig. 4. With projection we obtain $Flow_{pr}(l) = \{ax_l + b_l \leq \dot{x} \leq (b - b_u)/(2a)\}$. In the constraint-based approach, we compute the infimum for each constraint separately, obtaining $Flow_{con}(l) = \{ax_l + b_l \leq \dot{x} \leq -ax_l + b\}$. As Fig. 4 illustrates, the overapproximation of the constraint-based approach can be considerable.

Iterative Refinement. The set of states S containing all possible values of x is initially chosen to be the invariant.

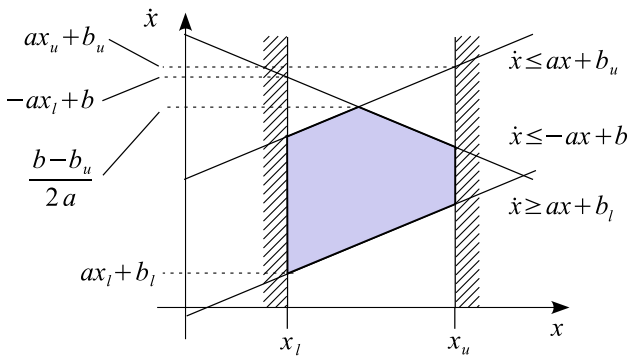

Fig. 4 Overapproximating affine dynamics with LHA dynamics

$Flow_{pr}(Inv)$, respectively $Flow_{con}(Inv)$, then yields the set of all derivatives possible inside this invariant, which is independent from the set that is actually reachable. We can further refine the set of derivatives—and the set of reachable states—by restricting S to the set of states that are reachable with $Flow_{pr}(Inv)$, possibly reiterating the process until convergence. This yields the following fixpoint computation for the operator $Post_t(P)$, where $Post^0_{t,pr} = P$:

$$Post^{k+1}_{t,pr} = \bigcup_{l\in loc(P)} l \times (P(l) \nearrow Flow_{pr}(Post^k_{t,pr})) \cap Inv(l). \tag{13}$$

Note that $Post^1_{t,pr} = Post_{t,pr}(P)$. We denote the fixpoint of the above sequence as $Post^\infty_{t,pr}(P)$, and the corresponding result of the constraint-based operator with $Post^\infty_{t,con}(P)$. The following example shall illustrate the qualitative difference that a single iteration of (13) can make.

Example 5[2] Consider $Flow(l) = \{\dot{x} = y \wedge \dot{y} = 0\}$ and $Inv`(l) = \{-1 \le y \le 1\}$. With $Flow_{pr}(Inv) = l \times \{\dot{x} \in [-1, 1], \dot{y} = 0\}$, the timed successors of a set of states $P = l \times \{x = 0 \wedge y = 1\}$ are $Post_{t,pr}(P) = l \times \{y = 1\}$. The overapproximation of $Flow$ with $Flow_{pr}$ results in negative values of x being reachable, even though $\dot{x} = y = 1$. Applying the fixpoint computation (13), we obtain using $Post^1_{t,pr} = Post_{t,pr}$ and $Flow_{pr}(Post^1_{t,pr}) = l \times \{\dot{x} = 1, \dot{y} = 0\}$ the fixpoint $Post^2_{t,pr}(P) = l \times \{x \ge 0 \wedge y = 1\}$, which is equal to the actual reachable set $Post_t(P)$.

In our examples, only the first 2–3 iterations yield a significant improvement. Instead of computing a fixpoint, PHAVer computes a fixed number of iterations given by the user.

3.3 Partitioning locations

When the dynamics are overapproximated as in the previous section, the error depends on the size of the invariant of the location. We partition each invariant into sufficiently small parts with a recursive splitting operator. Each splitting cuts the invariant in two along a hyperplane, and replaces the original location with two copies, each assigned a part of the original invariant. The hyperplanes for splitting are chosen from a user-defined set. The splitting recursion terminates if the size of the invariant or the spread of the derivatives fall below a given threshhold. By using small enough partitions, one is able to approximate the behavior of the original hybrid automaton arbitrarily close [5]. Note that this requires the introduction of the derivatives as auxiliary variables, as will be discussed at the end of this section.

The splitting of a location consists of duplicating the location, including incoming and outgoing transitions as well as flow, invariant and initial states. The invariants of the location duplicates may be restricted to subsets as long as they cover the original invariants. Formally, this corresponds to the following operation:

Definition 5 (*Invariant split*) (modified from [5][3]) A *split* $\mathcal{S}$ for a hybrid automaton H maps each location l to a finite set $\{S^l_1, \ldots, S^l_k\}$ of sets of valuations over X such that there exists a finite cover $\mathcal{O}^l = \{O^l_1, \ldots, O^l_k\}$ of $Inv(l)$ with $S^l_i = Inv(l) \cap O^l_i$ for $i = 1, \ldots, k$. The *split* of H along $\mathcal{S}$ is the hybrid automaton $split(H, \mathcal{S}) = (Loc_\mathcal{S}, (X, C, O), Lab, Edg_\mathcal{S}, Flow_\mathcal{S}, Inv_\mathcal{S}, Init_\mathcal{S})$ with

- $Loc_\mathcal{S} = \{(l, S) \mid l \in Loc, S \in \mathcal{S}(l)\}$,
- $Edg_\mathcal{S} = \{((l, S), a, \mu, (l', S')) \mid (l, a, \mu, l') \in Edg\}$,
- $Flow_\mathcal{S}((l, S)) = Flow(l)$,
- $Inv_\mathcal{S}((l, S)) = Inv(l) \cap S$, and
- $Init_\mathcal{S}((l, S)) = Init(l) \cap S$.

The behavior of the split automaton $H_\mathcal{S}$ is identical, i.e., bisimilar, to the behavior of the original automaton if the cover $\mathcal{O}$ is open [5], or the dynamics are affine [12].

Recall that a hyperplane h is defined by an equation $a^T_h x = b_h$, where the normal vector a_h determines its direction and the inhomogeneous term b_h its position. Let the *slack* of h in a location l be defined by

$$\Delta(a_h) = \sup_{x\in Inv(l)} a^T_h x - \inf_{x\in Inv(l)} a^T_h x.$$

In PHAVer, we recursively split one location l at a time along a hyperplane $a^T_h x = b_h$, i.e., we apply an invariant split with $S^l_1 = \{a^T_h x \le b_h\}$, $S^l_2 = \{a^T_h x \ge b_h\}$. Note that only reachable locations are split. The user provides a list of candidate normal vectors $a_{h,i}$ together with a minimum and maximum

[2] Many thanks to R. J. M. Theunissen for inspiring the example.

[3] We do not require the cover to be open.

value for the slack of each hyperplane:

$$Cand = \{(a_{h,1}, \Delta_{min,1}, \Delta_{max,1}), \ldots, (a_{h,m}, \Delta_{min,m}, \Delta_{max,m})\}.$$

This allows the user to include expert knowledge by choosing planes and location sizes suitable for the system. A trivial choice for the hyperplanes are the directions of the axes. Let P be the set of reachable states at the time of the splitting. The position b_h of the hyperplane is chosen to be the center of the location, i.e.,

$$b_h := \left(\sup_{x \in Inv(l)} a_h^T x + \inf_{x \in Inv(l)} a_h^T x \right) \Big/ 2,$$

if both supremum and infimum exist. Otherwise, we introduce a partition at distance Δ_{min} to the reachable states by setting

$$b_h := \Delta_{min} + \sup_{x \in P} a_h^T x, \quad \text{or} \quad b_h := -\Delta_{min} + \inf_{x \in P} a_h^T x,$$

whichever lies inside the invariant. In related work, a method for positioning of the hyperplane based on optimizing the set of derivatives was proposed in [12].

In each split, the best hyperplane is chosen according to a number of criteria. We provide an overview of the criteria, followed by a detailed description of the selection process. In principle, each location is split until the slack of every candidate hyperplane h_i satisfies $\Delta(a_{h,i}) \leq \Delta_{min,i}$. We account for the dynamics of the system using the spatial angle that is spanned by the derivatives in a location. Let the *spread* of a set of valuations X be

$$\sphericalangle(X) = \inf_{x,y \in X} x^T y \Big/ |x||y|.$$

The spread of the derivatives in a location l, constrained to the states S is then

$$\sphericalangle_{deriv}(l, S) = \sphericalangle((Flow(l) \cap S|^{X \cup \dot{X}}) \downarrow_{\dot{X}}).$$

The spread of the derivatives is used in two ways: The partitioning of a location is stopped once the spread is smaller than a given minimum $\sphericalangle_{min}$, and the constraints are sorted according to the spread of the derivatives in the location after the splitting. The slack $\Delta_{max,i}$ may be specified to split at least to that slack without regard to $\sphericalangle_{min}$.

For each splitting, the candidate hyperplanes $a_h^T x = b_h$ are sorted according to the following criteria (smallest is best):

1. slack

$$c_1 = \begin{cases} -\Delta(a_h)/\Delta_{min,h} & \text{if } \Delta(a_h) > \Delta_{min,h}, \\ \infty & \text{otherwise.} \end{cases}$$

2. reachable states only on one side

$$c_2 = \begin{cases} 1 & \text{if } \exists x, x' \in P : a^T x < b \wedge a^T x' > b \\ 0 & \text{otherwise.} \end{cases}$$

3. spread of the derivatives (discard constraint if a minimum spread $\sphericalangle_{min}$ is reached and the slack is smaller than $\Delta_{max,h}$)

$$c_3 = \begin{cases} -\sphericalangle_{deriv}(l, Inv) & \text{if } \sphericalangle_{deriv}(l, Inv) \geq \sphericalangle_{min} \\ & \quad \vee \Delta(a_h) > \Delta_{max,h}, \\ \infty & \text{otherwise.} \end{cases}$$

4. derivative spread after the constraint is applied

$$c_4 = -\max \Big\{ \sphericalangle_{deriv}(l, \{(l,x) \in Inv \mid a_h^T x \leq b_h\}), \sphericalangle_{deriv}(l, \{(l,x) \in Inv \mid a_h^T x \geq b_h\}) \Big\}.$$

By default, the candidate hyperplanes are sorted by evaluating the tuple (c_1, c_2, c_3) lexicographically. The best one is chosen that does not have a criterion evaluating to ∞. If no such hyperplane exists, the splitting recursion terminates. Note that c_3 can be deactivated by specifying $\sphericalangle_{min} = 1$. When c_4 is activated, the sorting is according to (c_4, c_1, c_2, c_3).

According to [5], the overapproximation of a hybrid automaton H with a linear hybrid automaton can be arbitrarily close to the original if the partition size is chosen sufficiently small. However, the splitting method in [5] relies on splitting not only the invariant, but also the flow predicate. This is necessary to approximate the flow aribtrarily close (consider a nonconvex flow). Any overapproximation by a convex flow will contain the convex hull no matter how small the invariant is partitioned, and can consequently not be arbitrarily close. When splitting the invariant is not sufficient, one may introduce the derivatives as auxiliary variables to implement the method of [5] by invariant split. Let $Y = \{y_1, \ldots, y_n\}$ be the auxiliary variables representing the derivatives. The transformed automaton is $H' = (Loc, (X \cup Y, C \cup Y, O), Lab, Edg', Flow', Inv', Init')$, where

- $Edg' = \{(l, a, \mu', l') \mid (l, a, \mu, l') \in \rightarrow, \mu' = \mu|^{X \cup Y \cup X' \cup Y'}\}$,
- $Flow'(l) = Flow(l)|^{X \cup \dot{X} \cup Y \cup \dot{Y}} \cap \{\bigwedge_{i=1,\ldots,n} \dot{x}_i = y_i\}$,
- $Inv'(l) = Inv(l)|^{X \cup Y}$, and
- $Init'(l) = Init(l)|^{X \cup Y}$.

The auxiliary variables are not restricted in transitions, invariants or initial states, and their derivatives are not restricted in the flow. They do not modify the behavior of the hybrid automaton except for the constraint $\dot{x}_i = y_i$ in the flow, which

has no effect on $\dot{x}_i$ since y_i is unrestricted. However, splitting the invariant will now also split the derivatives, therefore implementing the method in [5].

3.4 Example: navigation benchmark

We illustrate the reachability analysis of PHAVer with a navigation benchmark proposed in [39]. It models an object moving in a plane, and it must be shown that a set of bad states is not reachable, and that a set of target states is eventually reached. It is a challenging four-dimensional system because of two integrators (turning velocity into position), whose marginal stability contributes to the accumulation of errors.

The dynamics of the object is defined by "desired", or equilibrium, velocities that depend on its position in a plane. A map M attributes to each unit square in the plane a value $i \in \{0, \ldots, 7\}$, which determines the desired velocity $v_d(i) = (\sin(i\pi/4), \cos(i\pi/4))^T$. A special symbol A denotes the set of target states, and B denotes the set of forbidden states for the object. We verified that the forbidden states are not reachable for the instances shown in Fig. 5, whose maps are given by

$$M_{\text{NAV01}-03} = \begin{pmatrix} \mathtt{B}\ 2\ 4 \\ 2\ 3\ 4 \\ 2\ 2\ \mathtt{A} \end{pmatrix}, \quad M_{\text{NAV04}} = \begin{pmatrix} \mathtt{B}\ 2\ 4 \\ 2\ 2\ 4 \\ 1\ 1\ \mathtt{A} \end{pmatrix}.$$

The dynamics of the state vector $(x_1, x_2, v_1, v_2)^T$ are given by

$$\begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} 0 & I \\ 0 & A \end{pmatrix} \begin{pmatrix} x \\ v \end{pmatrix} - \begin{pmatrix} 0 \\ A \end{pmatrix} \begin{pmatrix} 0 \\ v_d(i) \end{pmatrix},$$
$$A = \begin{pmatrix} -1.2 & 0.1 \\ 0.1 & -1.2 \end{pmatrix}.$$

The initial states for for NAV01–NAV03 are defined by $x_0 \in [2, 3] \times [1, 2]$, for NAV04 by $x_0 \in [0, 1] \times [0, 1]$, and

$$v_{0,\text{NAV01}} \in [-0.3, 0.3] \times [-0.3, 0],$$
$$v_{0,\text{NAV02}} \in [-0.3, 0.3] \times [-0.3, 0.3],$$
$$v_{0,\text{NAV03}} \in [-0.4, 0.4] \times [-0.4, 0.4],$$
$$v_{0,\text{NAV04}} \in [0.1, 0.5] \times [0.05, 0.25].$$

As splitting constraints we use

$$Cand = \{(v_1, \delta_1, \infty), (v_2, \delta_2, \infty)\},$$

where appropriate δ_i were established by some trial-and-error runs, and (c_1) as splitting criterion. Note that x_1, x_2 need not be partitioned, since they depend only on v. The other analysis parameters were left at their default setting. While we need to specify bounds for the analysis region, we can handle the unbounded case by checking that the reachable state space is strictly contained in the analysis region. All instances shown were obtained with a-priori bounds of $[-2, 2]$ on the velocities, and the reachable velocities remained within an interval $[-1.1, 1.1]$, which confirms our a-priori bounds as valid. Figure 5 shows the set of reachable states computed by PHAVer as a result. For the instances NAV01–NAV04, the analysis was fairly straightforward, with $\delta_i = 1$. Higher instances require a much higher level of accuracy, and lead to large numbers of polyhedra. Applying a convex hull overapproximation would remedy this problem, but without complexity management the analysis did not terminate. We present results after introducing comlpexity mananagement in the next section. In comparison, for a predicate abstraction tool the following times were reported in [40]: For NAV01–NAV03 $34s$, $153s$ (68MB) and $152s$ (180MB), respectively, on a Sun Enterprise 3000 (4×250 MHz UltraSPARC) with 1 GB RAM. In [13] it is reported that HSOLVER was not able to show safety of instances similar to NAV01–NAV03.

4 Managing the Complexity of Polyhedra

In exact fixpoint computations with polyhedra, the size of numbers in the formula as well as the number of constraints typically increases unless the structure of the hybrid system imposes boundaries, for example with resets or invariants. To keep the complexity manageable, we propose to simplify polyhedra in a strictly conservative fashion by limiting the number of bits, that is the size of coefficients, and the number of constraints. While showing good performance in practice, both methods have the disadvantage that the approximation error may be unbounded, and is not localized. In practice, both simplifications are applied when the number of bits or constraints exceeds a given threshold that is significantly higher than the reduction level. The resulting hysteresis between exact computations and overapproximations gives cyclic dependencies time to stabilize.

4.1 Limiting the number of bits

We consider a convex polyhedron in constraint representation, i.e., given as the conjunction of constraints $a_i^T x \bowtie_i b_i$, where a_i is a vector of the coefficients $a_{ij} \in \mathbb{Z}, i = 1, \ldots, m$, $j = 1, \ldots, n$, $\bowtie_i \in \{<, \leq\}$, and inhomogeneous coefficients $b_i \in \mathbb{Z}$. We assume that the a_{ij} and b_i have no common factor and that there are no redundant constraints. Note that we usually do not simplify equalities to preserve the affine dimension, i.e., the inhabited subspace, of the polyhedron, but they can be simplified by converting each equality into the conjunction of two inequalities. The goal is to find, with as little overapproximation as possible, a new constraint $\alpha_i^T x \bowtie_i \beta_i$ that contains all solutions of $a_i^T x \bowtie_i b_i$, and whose coefficients α_{ij}, β_i have less than z bits, i.e., $|\alpha_{ij}|$, $|\beta_i| \leq 2^z - 1$.

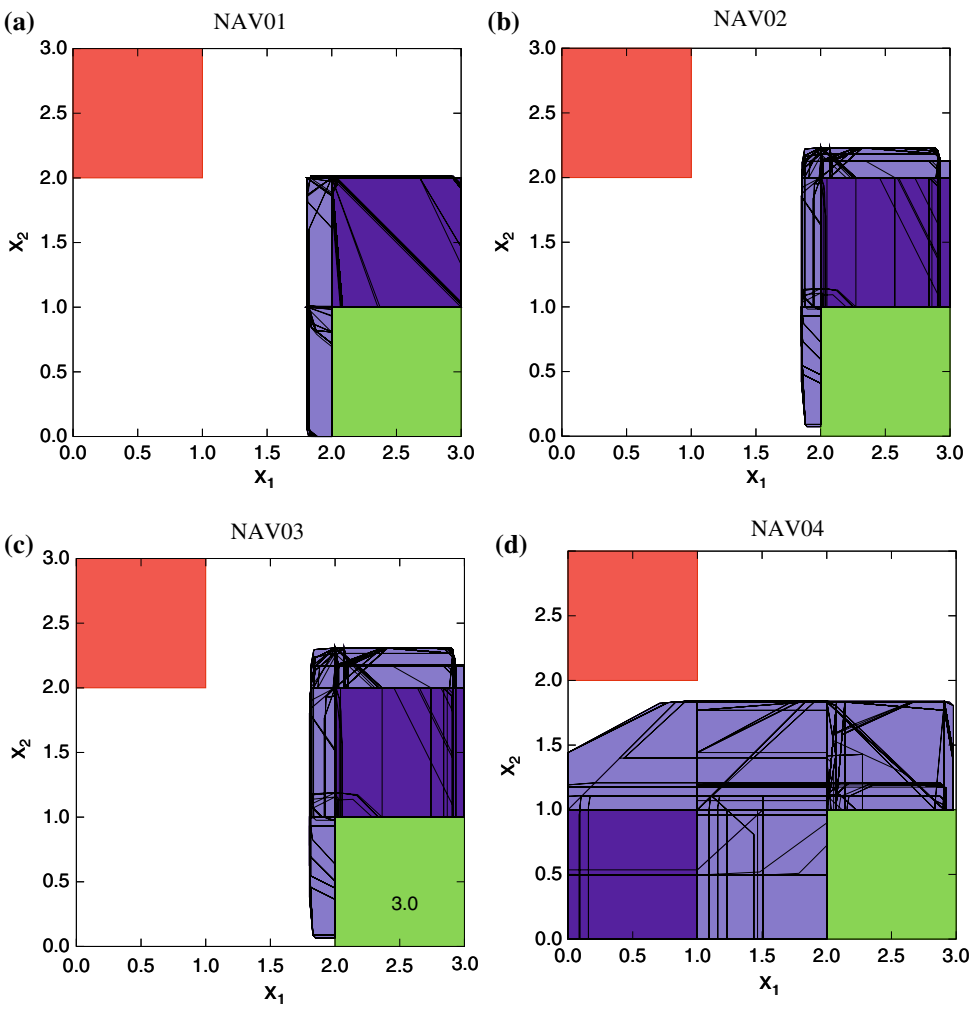

Fig. 5 Reachable states in the x_1, x_2-plane (initial states darkest)

Our approach is to scale down the a_{ij} so they are all smaller than $2^z - 1$, and then find a β_i that makes the constraint conservative. Because the new coefficients need to be integers, we need to account for rounding errors. We obtain an initial estimate of a scaling factor $f > 0$, and reduce it if it results in a β_i that is too large. We can represent the new coefficients as

$$\alpha_{ij} = f a_{ij} + r_{ij},$$
$$\beta_i = f b_i + r_i,$$

with rounding errors r_{ij}, $|r_{ij}| \leq 0.5$ and an unbounded error r_i for the inhomogeneous term. In this representation, our goal is to find a scaling factor f that is close to 1 and results in a small error r_i. There is no a-priori bound on r_i because it depends on the new direction α_i and the other constraints that define the polyhedron.

We obtain an initial estimate of f based on the assumption that β_i is close to $f b_i$, say the closest integer. Since β_i must be rounded upwards to guarantee conservativeness, we get $|r_i| \leq 1$ as an optimistic estimate. This gives us the following upper bounds for f:

$$f \leq (2^z - 3/2)/|a_{ij}| \quad \text{and} \tag{14}$$
$$f \leq (2^z - 2)/|b_i|. \tag{15}$$

To predict the effects of rounding on the new coefficients is difficult and would lead to a mixed integer linear program. We employ a heuristic algorithm, shown in Fig. 6. Let $round(x)$ be a function that returns the next integer between x and zero, and $ceil(x)$ be a function that rounds to the next larger integer. First, we estimate f based on (14) and (15), then we compute a new β_i using linear programming. If β_i has more than z bit, we decrease f and start over. The procedure is repeated until all $\alpha_{ij} = 0$, in which case the problem is infeasible. Note that even if it is feasible, the new polyhedron may not be bounded because the reduced constraints may be parallel. Figure 7 illustrates the basic scheme. The normal vector a_i of the constraint, shown in (a), is approximated by α_i, as shown in (b). Linear programming yields the inhomogeneous term q that makes the constraint tangent to

procedure *LimitConstraintBits*
Input: Polyhedron as a set of constraints
$P = \{a_k^T x \bowtie_k b_k | k = 1, \ldots, m\}$,
index i to constraint to be limited,
desired number of bits z
Output: new constraint $\alpha_i^T x \bowtie_i b_i$
$success := false$;
$f := \min\{(2^z - 3/2)/|a_{ij}|, (2^z - 2)/|b_i| \mid j = 1, .$
while $\neg success$ **do**
for $j = 1, \ldots, n$ **do** $\alpha_{ij} := round(f a_{ij})$ **od**;
$q := \inf_{x \in P} \alpha_i^T x$;
if $\alpha_i = 0$ **or** $q = -\infty$ **then abort fi**;
$\beta_i := ceil(q)$;
if $|\beta_i| \leq 2^z - 1$ **then** $success := true$
else $f := min\{f/2 - 3/(4|a_{ij}|), (2^z - 2)/|$
$\beta_i| \mid j = 1, \ldots, n\}$ **fi**;
od.

Fig. 6 Limiting the number of bits of a constraint

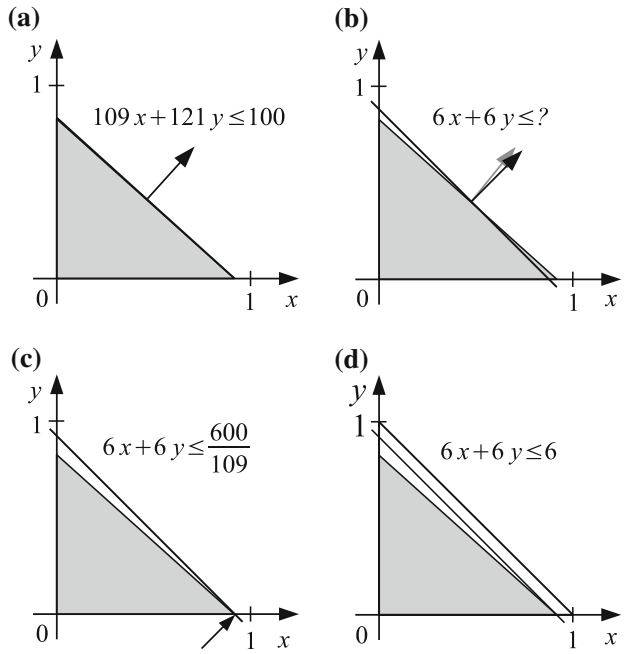

Fig. 7 Limiting the number of bits of a constraint

the polyhedron, as in (c). Rounding of q yields β_i, and the polyhedron outlined in (d).

4.2 Limiting the number of constraints

We use sets of convex polyhedra to describe the symbolic states computed by the reachability algorithm. The number of constraints of the polyhedra often increases with the number of iterations of a fixpoint computation. We limit this number to keep the analysis computationally feasible. Similar reductions have been proposed, for example bounding boxes [41], or oriented rectangular hulls [33] keep the number of constraints fixed to $2n$. Instead of computing an entirely new set of constraints, we propose to simply drop the least significant of the constraints. As with limiting the number of bits, we usually chose to not limit equalities in order to preserve the affine dimension of the polyhedron. If an equality is to be limited, it is simply replaced by two inequalities, which are then each limited.

We measure the significance of a constraint based on a criterion *crit* that measures the the difference between the polyhedron with and without the constraint. Let P be a set of linear constraints describing a convex polyhedron, and $P^{\setminus i} = P \setminus \{a_i^T x \bowtie_i b_i\}$ be the polyhedron without its ith constraint. Then the difference between the points contained in P and $P^{\setminus i}$ is the polyhedron $P^{\neg i} = P^{\setminus i} \cup \{-a_i^T x \overline{\bowtie}_i - b_i\}$, where $(\bowtie_i, \overline{\bowtie}_i) \in \{(<, \leq), (\leq, <)\}$, obtained by simply replacing the ith constraint with its complement. It has less non-redundant constraints than P and is therefore preferable in the formulations below. We consider three methods:

1. volumetric: Let $V(P)$ be the volume of the points contained in P. Then $crit = V(P^{\setminus i}) - V(P) = V(P^{\neg i})$. Requires $P^{\neg i}$ to be bounded.
2. slack: Let $b_{max} = \max_x a_i^T x$ s.t. $x \in P^{\neg i}$. Then $crit = (b_{max} - b_i)/||a_i||$, i.e., the distance, measured in the direction of the constraint, between the points farthest apart in $P^{\neg i}$. Requires $P^{\neg i}$ to be bounded in the direction of a_i.
3. angle: $crit = -\max_{j \neq i} a_j^T a_i$. Measures the negative cosine of the closest angle between the normal vector of the ith constraint and all others.

Our goal is to select the z most important out of m original constraints. It would be expensive to examine all $\binom{m}{z}$ possible combinations of constraints. Instead, we consider two heuristics:

1. deconstruction: Starting from the entire set of constraints, drop the $m - z$ constraints with the least effect according to *crit*.
2. reconstruction: Starting from an empty set of constraints, add the z constraints with the greatest effect according to *crit*.

The criteria based on volume and slack require the initial polyhedron to be bounded, for which one could use, e.g., the invariant of the location. The following example shall illustrate the difference between volumetric and angle criteria, and its potential unboundedness.

Example 6 Consider the polyhedron shown in Fig. 8a. It has 6 constraints A–F, whose angles with the neighbors are noted in the graph. In a volume based deconstruction with

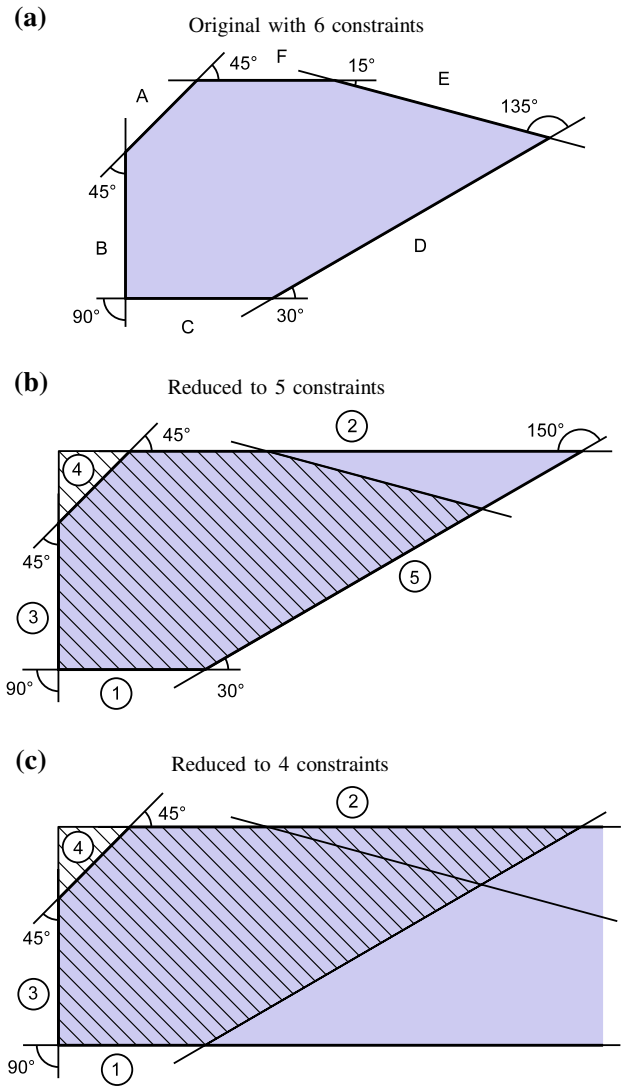

Fig. 8 Example for limiting the number of constraints by volumetric deconstruction (*hashed*) and angle based reconstruction (*shaded*)

5 constraints, constraint A is removed since that causes the smallest change in volume. The resulting polyhedron is shown hashed in Fig. 8b. The angle based reconstruction with 5 constraints results in the shaded polyhedron in Fig. 8b, where the constraints are labeled in the order they are chosen: First, an arbitrary initial constraint is chosen, say constraint C. The second choice is the constraint that has the largest angle with C, i.e., that is most opposed to it. In this case, this is constraint F, since it has an angle of 180° with C. The third choice is the one that is most opposed to both C and F, here constraint B because it has an angle of 90° with both C and F. The fourth constraint is A, with minimum angles of 45°, and the fifth is D with a minimum angle of 30°. Figure 8c shows the reduction to 4 constraints. Here the angle based method results in an unbounded polyhedron because constraint D is not chosen. An algorithm should take this possibility into account and test for boundedness.

procedure *LimitConstraintsByAngle*
Input: Polyhedron P as a set of constraints
$a_i^T x \bowtie_i b_i$, $i = 1, \ldots, m$,
desired number of constraints z
Output: Polyhedron H
for $i = 1, \ldots, m$, $j = 1, \ldots, m$, $j > i$ **do**
$\alpha(i, j) := a_i^T a_j$
od;
$H := \{a_k^T x \bowtie_k b_k \mid k = \text{argmin}_k(\max_j |a_{kj}|)\} \cup$
$\{a_i^T x \bowtie_i b_i \mid \bowtie_i \in \{=\}\}$;
$C := P \setminus H$;
while $(|C| > 0 \wedge (|H| < z \vee$
$(bounded(P) \wedge \neg bounded(H)))$ **do**
$j = \text{argmin}_j\ (max_i \alpha(i, j))$ **s.t.**
$a_i^T x \bowtie_i b_i \in H, a_j^T x \bowtie_j b_j \in C$;
$H := H \cup \{a_j^T x \bowtie_j b_j\}$;
$C := C \setminus \{a_j^T x \bowtie_j b_j\}$
od.

Fig. 9 Reconstructing a polyhedron with a limited number of constraints by angle prioritization

The construction method with an angle criterion was the fastest in our experiments. The angle calculations can be sped up by using a look-up table $\alpha(i, j)$ that maps an angle to every pair of constraints. This yields an algorithm of complexity $O(nm^2 + m^3)$, shown in Fig. 9, where C is the set of candidate constraints and H is the set of chosen constraints. It includes a test that preserves the boundedness of P. H is initialized with the set of equalities, which are not reduced to preserve the affine dimension of the polyhedron, and an arbitrary initial constraint. Here we choose the one with the smallest coefficients. In a while-loop, the constraint is chosen based on the best of the worst-cases, i.e., the smallest angle with the constraints in H. Since $a_j^T a_i$ is the cosine of the angle, choosing the smallest angle translates into maximizing $a_j^T a_i$. The constraint is added to H and removed from the candidates C, and the procedures is repeated until $|H| \geq z$ and the boundedness of P implies boundedness of H. This algorithm is in our implementation ∼1,000× faster than a slack based deconstruction for limiting 400 constraints down to 32 in 4 dimensions.

The following example shall illustrate that complexity management is useful to induce termination, and that the constraint-based overapproximation of dynamics may be preferable when a projection-based overapproximation leads to too complex sets of states. Still, the relationship between overapproximation method, complexity and quick convergence is not a trivial one.

Example 7 NAV01–NAV04 from Sect. 3.4 can be analyzed without complexity management, but the computed reachable set consists of a large number of polyhedra. If we compute the convex hull of the reachable states in each location,

Table 2 PHAVer performance, navigation benchmark

Instance	Time (s)	Mem. (MB)	Iter.	Checks	Automaton		Reachable Set			
					Loc.	Trans.	Loc.	Poly.	Con.	Bits
NAV01	8.7	29.0	11	386	72	1123	45	386	20	62
NAV02	14.5	38.2	10	803	72	1126	45	803	20	33
NAV03	14.4	38.0	10	795	72	1126	45	795	20	33
NAV04	13.6	47.6	8	562	122	2057	85	562	18	32

Table 3 PHAVer performance, navigation benchmark with convex hull overapproximations and complexity management

Instance	Time (s)	Mem. (MB)	Iter.	Checks	Automaton		Reachable Set			
					Loc.	Trans.	Loc.	Poly.	Con.	Bits
Projection-based overapproximation of dynamics										
NAV01	5.2	25.6	22	185	70	1090	43	43	30	308
NAV02	6.4	25.6	20	230	72	1126	45	45	42	282
NAV03	6.5	25.6	20	232	72	1126	45	45	48	290
NAV04	6.7	25.9	52	330	72	1116	54	54	18	32
Constraint-based overapproximation of dynamics										
NAV01	4.6	24.0	7	135	72	1126	45	45	16	33
NAV02	4.7	26.2	6	137	72	1126	45	45	16	37
NAV03	4.8	26.2	6	137	72	1126	45	45	16	40

we only get only polyhedron per location but the algorithm does not terminate. Limiting the number of bits remedies this problem. Limiting to 24 bits with a triggering threshold of 300 and to 48 constraints with a threshold of 96 halves the computation time and memory, see Table 3, compared to the exact computations used in Table 2. Note that the resulting number of bits and constraints may be larger than the threshold because the time elapse operator must be applied after limiting, see Remark 1. For constraint-based overapproximation NAV04 does not terminate within reasonable time due to very slow convergence.

4.3 Example: tunnel-diode oscillator circuit

We present experimental results for a tunnel-diode oscillator circuit taken from [26] with the parameters used in [42]. We model the current I_L through the inductor and the voltage drop V_d of a tunnel diode in parallel with the capacitor of a serial RLC circuit, which are in stable oscillation for the given parameters. The state equations are given by

$$\dot{V}_d = 1/C(-I_d(V_d) + I_L), \tag{16}$$

$$\dot{I}_L = 1/L(-V_d - 1/G \cdot I_L + V_{in}), \tag{17}$$

where $C = 1\ pF$, $L = 1\ \mu H$, $G = 5\ m\Omega^{-1}$, $V_{in} = 0.3\ V$, and the diode current

$$I_d = \begin{cases} 6.01V_d^3 - 0.992V_d^2 + 0.0545V_d \\ \quad \text{if } V_d \le 0.055, \\ 0.0692V_d^3 - 0.0421V_d^2 + 0.004V_d + 8.96 \cdot 10^{-4} \\ \quad \text{if } 0.055 \le V_d \le 0.35, \\ 0.263V_d^3 - 0.277V_d^2 + 0.0968V_d - 0.0112 \\ \quad \text{if } 0.35 \le V_d. \end{cases}$$

Following the procedure outlined in the previous section, a piecewise affine envelope was constructed for the tunnel diode characteristic $I_d(V)$. We choose 32 intervals for the range $V_d \in [-0.1, 0.6]$ to yield sufficient accuracy and so obtain a piecewise affine model for (16). It is modeled as a hybrid automaton with V_d as an output- and I_L as an input-variable, and consists of 32 locations, one for each interval. Equation (17) is affine, and is modeled as a hybrid automaton with V_d as an input and a single location. Both models are composed and analyzed in PHAVer. Figure 10a shows the computed reachable states for initial states given by $V_d \in [0.42V, 0.52V]$, $I_L = 0.6mA$. It also shows the invariants (dashed) generated by the partitioning algorithm using the constraints $Cand = \{(V_d, 0.7/128, \infty), (I_L, 1.5/128, \infty)\}$, i.e., max. 128 partitions in both directions, and splitting criterion (c_3, c_1) with $\lhd_{\min} = 0.99$. The analysis with PHAVer takes 17.1s and 68.0MB RAM, with the largest coefficient taking up 2508 bits and at most 5 constraints per polyhedron.

We now obtain bounds on the cycle time of the oscillator by composing the circuit model with a monitor automaton. The cycle time is the maximum time it takes any state to

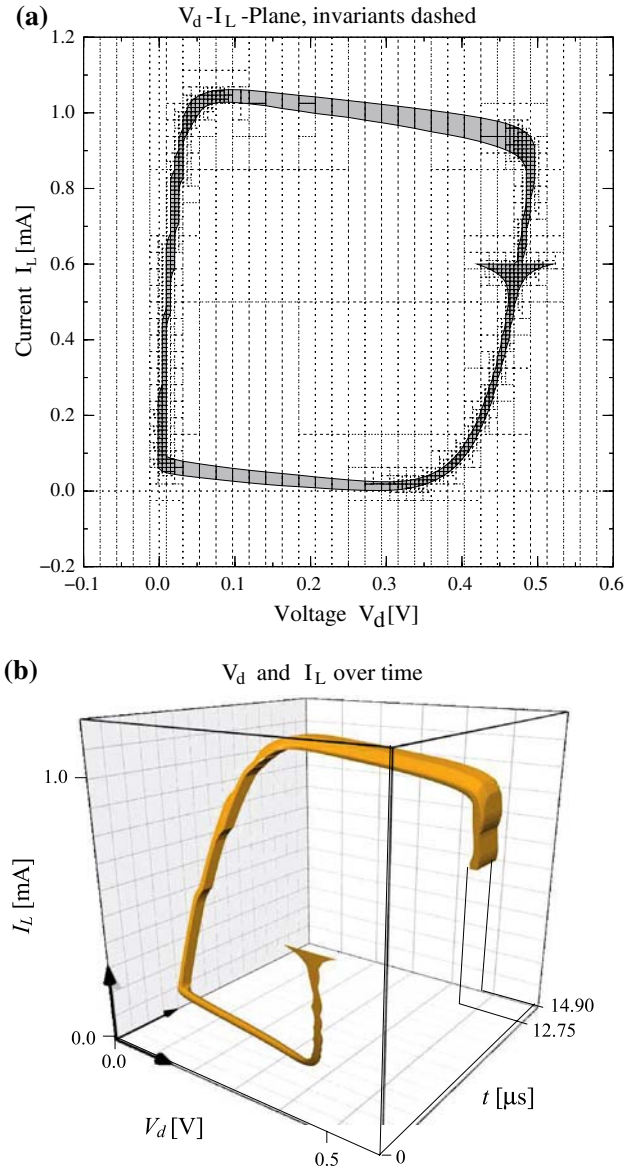

Fig. 10 Reachable states of Tunnel Diode Circuit

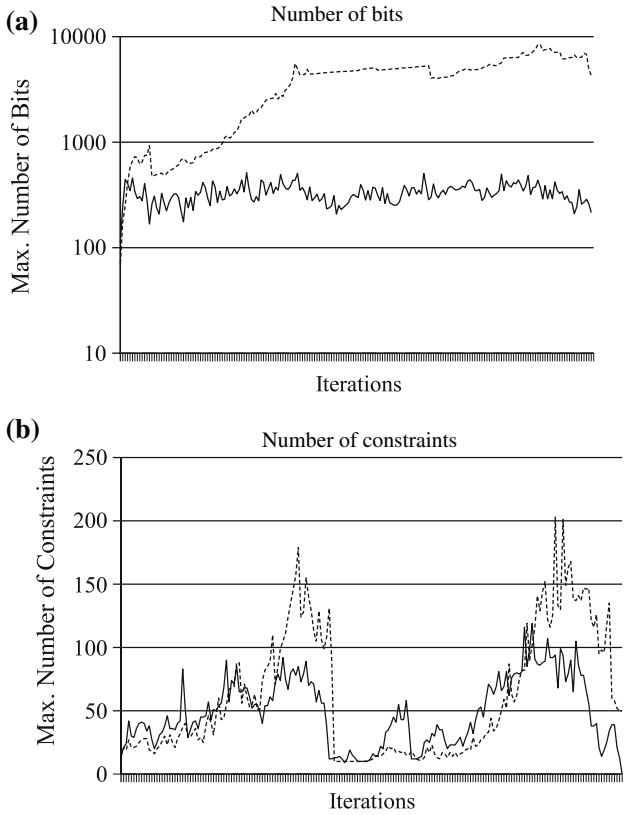

Fig. 11 Clocked Tunnel Diode Circuit, exact (*dashed*) and with limits on bits and constraints (*solid*)

cross the threshold $I = 0.6\mu A, V > 0.25V$ twice. For the clocked circuit, the number of bits and constraints grows rapidly and a more precise analysis, such as shown in Fig. 10b is only possible with limits on both. We compare the unlimited and limited analysis for constraint candidates $Cand = \{(V_d, 0.7/64, \infty), (I_L, 1.5/64, \infty)\}$ and using convex hull overapproximations for the flows. The bits are limited to 24 when a threshold of 300 bits is reached, and the constraints to 32 with a threshold of 200. Figures 11a, b shows a polynomial increase in the number of constraints, and an exponential increase of the number of bits in the new polyhedra found at each iteration. The analysis takes 1412 s (381 MB) when unlimited, and 90 s (132 MB) when limited and yields the reachable states shown in Fig. 10b. The bounds for on the cycle time are [12.75, 14.90] μs when unlimited. The relative error of the limited analysis is 0.006% for the lower bound and 0.08% for the upper bound. At a more than fifteenfold increase in speed, the overapproximation is negligible and results in a cycle time estimate that is practically identical. Note that a comparison was only possible at the lowest level of accuracy. Higher accuracy can not be achieved at all without limiting due to an drastic increase in computation time.

In comparison with CheckMate [42], PHAVer is able to analyze the circuit at higher accuracy, and obtains results for parameters where CheckMate does not [8].

5 Conclusions

PHAVer, a new tool for verifying safety properties of linear hybrid automata, provides infinite precision arithmetic in a robust implementation, on-the-fly overapproximation of affine dynamics, and supports compositional and assume/guarantee-reasoning. We propose heuristics to conservatively overapproximate polyhedra by limiting the number of bits and constraints, which often are indispensable for managing the complexity of polyhedral computations. Experimental results for a navigation benchmark and a tunnel diode circuit indicate their effectiveness, and PHAVer outperforms other

currently available verification tools. The results suggest that the benefits of exact polyhedral computations, such as exact testing for containment, can outweigh the costs incurred by the indispensible complexity management. Future research will focus on accelerating convergence and termination, e.g., using widening [37], and using the transition topology to improve the search algorithm. PHAVer is available at http://www-verimag.imag.fr/~frehse/.

Acknowledgments The author is most grateful for the numerous inspiring discussions with Prof. Bruce Krogh, whose insightful guidance was indispensable, and to Prof. Frits W. Vaandrager and Prof. Sebastian Engell for their generous support and supervision. This research was supported in part by the US ARO contract no. DAAD19-01-1-0485, the US NSF contract no. CCR-0121547, and the Semiconductor Research Corporation under task ID 1028.001. Many thanks go to Olaf Stursberg, who suggested working on a successor to HyTech, to R.J.M. Theunissen, Gabriela Marin, Laurent Doyen, Li Hong and Scott Little for bug reports and helpful suggestions, and to Flavio Lerda for his invaluable compilation scripts. Finally, a special thanks goes to the PPL development team and in particular to Roberto Bagnara for his wonderful help.

References

1. Henzinger, T.A., Ho, P.-H., Wong-Toi, H.: HYTECH: a model checker for hybrid systems. Int. J. Softw. Tools Technol. Transfer **1**(1–2), 110–122 (1997)
2. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS, pp. 332–344. IEEE Computer Society (1986)
3. Henzinger, T.A.: The theory of hybrid automata. In: Proc. 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96, New Brunswick, New Jersey, 27–30 July 1996, pp. 278–292. IEEE Computer Society Press (1996)
4. Alur, R., Henzinger, T.A., Ho, P.-H.: Automatic symbolic verification of embedded systems. IEEE Trans. Softw. Eng. **22**, 181–201 (1996)
5. Henzinger, T.A., Ho, P.-H., Wong-Toi, H.: Algorithmic analysis of nonlinear hybrid systems. IEEE Trans. Automat. Control **43**(4), 540–554 (1998)
6. Frehse, G., Han, Z., Krogh, B.H.: Assume-guarantee reasoning for hybrid i/o-automata by over-approximation of continuous interaction. In Proc. 43rd IEEE Conf. Decision and Control (CDC'04), December 14–17, 2004, Atlantis, Bahamas (2004)
7. Frehse, G.: Compositional verification of hybrid systems using simulation relations. PhD thesis, Radboud University Nijmegen (2005)
8. Frehse, G., Krogh, B.H., Rutenbar, R.A., Maler, O.: Time domain verification of oscillator circuit properties. In: Maler, O. (ed.) Workshop on Formal verification of Analog Circuits (ETAPS Satellite Event), Edinburgh, Scotland, April 2–10, 2005. ENTCS, vol. 153, pp. 9–22 (2006)
9. Frehse, G., Krogh, B.H., Rutenbar, R.A.: Verifying analog oscillator circuits using forward/backward refinement. In: Proc. Conf. on Design, Automation and Test in Europe (DATE 06). ACM SIGDA, Munich, Germany (2006)
10. van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H.: Formal verification of chi models using phaver. In: Troch, I., Breitenecker, F. (eds.) Proc. MathMod 2006, Vienna, ARGESIM Reports, February (2006)
11. Podelski, A., Wagner, S.: Model checking of hybrid systems: From reachability towards stability. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC. LNCS, vol. 3927, pp. 507–521. Springer, Heidelberg (2006)
12. Doyen, L., Henzinger, T.A., Raskin, J.-F.: Automatic rectangular refinement of affine hybrid systems. In: Proc. FORMATS'05. LNCS, vol. 3829, pp. 144–161. Springer, Heidelberg (2005)
13. Ben Makhlouf, I., Kowalewski, S.: An evaluation of two recent reachability analysis tools for hybrid systems. In Proc. IFAC Conf. Analysis and Design of Hybrid Systems (ADHS'06) (2006)
14. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theor. Comput. Sci. **138**(1), 3–34 (1995)
15. Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid I/O automata. Informat. Comput. **185**(1), 105–157 (2003)
16. Cofer, D.D., Engstrom, E., Goldman, R.P., Musliner, D.J., Vestal, S.: Applications of model checking at Honeywell Laboratories. In: Dwyer, M.B. (ed.) Model Checking Software, 8th Int. SPIN Workshop, Toronto, Canada, May 19–20, 2001. LNCS, vol. 2057, pp. 296–303. Springer, Heidelberg (2001)
17. Henzinger, T.A., Preussig, J., Wong-Toi, H.: Some lessons from the hytech experience. In: Proc. of the 40th Annual Conf. on Decision and Control (CDC'01), pp. 2887–2892. IEEE Press, New York (2001)
18. Kowalewski, S., Stursberg, O., Fritz, M., Graf, H., Hoffmann, I., Preussig, J., Remelhe, M., Simon, S., Treseler, H.: A case study in tool-aided analysis of discretely controlled continuous systems: The two tanks problem. In: Antsaklis, P.J., Kohn, W., Lemmon, M.D., Nerode, A., Sastry, S. (eds.) Hybrid Systems V. LNCS, vol. 1567, pp. 163–185. Springer, Heidelberg (1999)
19. Tomlin, C.: Verification of an air traffic management protocol using hytech. Course Project for EE290A, taught by Prof. T. A. Henzinger, Spring 1996, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley (1996)
20. Henzinger, T.A., Wong-Toi, H.: Using HyTech to synthesize control parameters for a steam boiler. In: Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control. LNCS, vol. 1165, pp. 265–282. Springer, Heidelberg (1996)
21. Müller, O., Stauner, T.: Modelling and verification using linear hybrid automata—a case study. Mathe. Comput. Modell. Dynam. Syst. **6**(1), 71–89 (2000)
22. Bagnara, R., Ricci, E., Zaffanella, E., Hill, P.M.: Possibly not closed convex polyhedra and the Parma Polyhedra Library. In: Hermenegildo, M.V., Puebla, G. (eds.) Static Analysis: Proc. of the 9th Int. Symposium. LNCS, vol. 2477, pp. 213–229. Springer, Madrid, Spain (2002)
23. Granlund, T., Ryde, K.: The GNU Multiple Precision arithmetic library version 4.0 (2001). http://www.swox.com/gmp/
24. Henzinger, T.A., Ho, P.-H., Wong-Toi, H.: HYTECH: the next generation. In: Proc. of the 16th IEEE Real-Time Systems Symposium (RTSS '95), p. 56. IEEE Computer Society (1995)
25. Stursberg, O., Kowalewski, S.: Approximating switched continuous systems by rectangular automata. In Proc. 5th European Control Conference, Karlsruhe (1999)
26. Hartong, W., Hedrich, L., Barke, E.: On discrete modeling and model checking for nonlinear analog systems. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification, 14th Int. Conference, CAV 2002, Copenhagen, Denmark, July 27–31, 2002. LNCS, vol. 2404, pp. 401–413. Springer, Heidelberg (2002)
27. Henzinger, T.A., Horowitz, B., Majumdar, R., Wong-Toi, H.: Beyond HYTECH: Hybrid systems analysis using interval numerical methods. In: Lynch, N.A., Krogh, B.H. (eds.) Hybrid Systems: Computation and Control, Third International Workshop, HSCC 2000, Pittsburgh, PA, USA, March 23–25, 2000. LNCS, vol. 1790, pp. 130–144. Springer, Heidelberg (2000)

28. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation based abstraction refinement. In: Morari, M., Thiele, L. (eds.) Proc. of the 8th Int. Workshop on Hybrid Systems: Computation and Control. LNCS, vol. 3414, pp. 573–589. Springer, Heidelberg (2005)
29. Silva, B.I., Stursberg, O., Krogh, B.H., Engell, S.: An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In Proc. 40th Conf. on Decision and Control (CDC'01), December (2001)
30. Wong-Toi, H.: Symbolic approximations for verifying real-time systems, December (1994)
31. Preußig, J., Kowalewski, S., Wong-Toi, H., Henzinger, T.A.: An algorithm for the approximative analysis of rectangular automata. In: Proc. of the Fifth Int. Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT). LNCS, vol. 1486, pp. 228–240. Springer, Heidelberg (1998)
32. Preußig, J.: Formale Überprüfung der Korrektheit von Steuerungen mittels rektangulärer Automaten. PhD thesis, Schriftenreihe des Lehrstuhls für Anlagensteuerungstechnik Band 4/2000, Universität Dortmund, Shaker Verlag, 2000. (in German)
33. Stursberg, O., Krogh, B.H.: Efficient representation and computation of reachable sets for hybrid systems. In: Maler, O., Pnueli, A. (eds.) Hybrid Systems: Computation and Control, 6th Int. Workshop, HSCC 2003 Prague, Czech Republic, April 3–5, 2003. LNCS, vol. 2623, pp. 482–497. Springer, Heidelberg (2003)
34. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) Hybrid Systems. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993)
35. Lafferriere, G., Pappas, G.J., Yovine, S.: Symbolic reachability computation for families of linear vector fields. J. Symbolic Comput. **32**, 231–253 (2001)
36. Frehse, G.: On timed simulation relations for hybrid systems and compositionality. In: Asarin, E., Bouyer, P. (eds.) FORMATS. LNCS, vol. 4202, pp. 200–214. Springer, Heidelberg (2006). ISBN 3-540-45026-2
37. Halbwachs, N., Proy, Y.-E., Roumanoff, P.: Verification of real-time systems using linear relation analysis. Formal Methods in System Design: An Int. Journal **11**(2), 157–185 (1997)
38. Ho, P.-H., Wong-Toi, H.: Automated analysis of an audio control protocol. In: Proc. Conf. on Computer-Aided Verification. LNCS, vol. 939, pages 381–394. Springer, Liege, Belgium (1995)
39. Fehnker, A., Ivancic, F.: Benchmarks for hybrid systems verification. In: Alur, R., Pappas, G.J. (eds.) Hybrid Systems: Computation and Control, 7th Int. Workshop, HSCC 2004, Philadelphia, PA, USA, March 25–27, 2004. LNCS, vol. 2993, pp. 326–341. Springer, Heidelberg (2004)
40. Ivancic, F.: Modeling and Analysis of Hybrid Systems. PhD thesis, University of Pennsylvania, Philadelphia, PA, December (2003)
41. Bemporad, A., Morari, M.: Verification of hybrid systems via mathematical programming. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) Hybrid Systems: Computation and Control, Second International Workshop, HSCC'99, Berg en Dal, The Netherlands, March 29–31, 1999. LNCS, vol. 1569, pp. 31–45. Springer, Heidelberg (1999)
42. Gupta, S., Krogh, B.H., Rutenbar, R.A.: Towards formal verification of analog designs. In: Proc. IEEE Intl. Conf. on Computer-Aided Design (ICCAD-2004), November 7–11, 2004, San Jose CA (USA)