PHAVer: Algorithmic Verification of Hybrid Systems past HyTech

Goran Frehse[1]

Dept. of Electrical and Computer Engineering, Carnegie Mellon University,
Pittsburgh, PA 15213, USA, gfrehse@ece.cmu.edu,
WWW home page: http://www.andrew.cmu.edu/~gfrehse

Abstract. In 1995, HyTech broke new ground as a potentially powerful tool for verifying hybrid systems – yet it has remained severely limited in its applicability to more complex 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. Affine dynamics are handled by on-the-fly overapproximation and by partitioning the state space based on user-definable constraints and the dynamics of the system. PHAVer's exact arithmetic is robust due to the use of the Parma Polyhedra Library, which supports arbitrarily large numbers. To 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 show the effectiveness of the approach.

1 Introduction

Systems with discrete as well as continuous dynamics, i.e., hybrid systems, are notoriously complex to analyze, and the algorithmic verification of hybrid systems remains a challenging problem, both from a theoretic point of view as well as from the implementation side. Ideally, one would like to obtain either an exact result or a conservative overapproximation of the behavior of the system, e.g., as the set of reachable states. An exact computation is possible with linear hybrid automata (LHA) [1], which are defined by linear predicates and piecewise constant bounds on the derivatives. They were proposed and studied in detail by Henzinger et al., see [2] for an extensive discussion, who presented in 1995 a tool called HyTech that could perform various computations with such systems [3]. It featured a powerful input language and functionality, but suffered from a major flaw: its exact arithmetic was using limited digits, which can quickly lead to overflow errors. While it was successfully used to analyze a number of examples, see [4,5] and references therein, the overflow problem prohibits any application to more complex systems.

The valuable experiences with HyTech have prompted a number of suggestions for improvement, a summary of which can be found in [4]. We address the most pressing ones with PHAVer (Polyhedral Hybrid Automaton Verifyer), a new tool for analyzing linear hybrid automata with the following characteristics:

– exact and robust arithmetic based on the Parma Polyhedra Library [6],
– on-the-fly overapproximation of piecewise affine dynamics,
– conservative limiting of bits and constraints in polyhedral computations,
– support for compositional and assume-guarantee reasoning. [1]

PHAVer's extended functionality and computational robustness open up new application domains as well as research issues that were abandoned because of the limits of previous implementations. Exact arithmetic entails, in addition to the security and beauty of formal correctness, the significant advantage of a separation of concerns. Problems of convergence, combinatorial explosion and nondeterminism can be identified as such, which is very difficult if they are intertwined with numerical difficulties. We present PHAVer's algorithm for overapproximating piecewise affine dynamics with LHA, which partitions locations with user-specified constraints to limit and localize the overapproximation. The constraints allow the user to include expert knowledge in refining certain variables to a specified detail, and can be adapted to the dynamics by prioritizing the size or the spread angle of the derivatives of a location. Due to the exact arithmetic, the size of coefficients as well as the number of constraints that define polyhedra can grow excessively. We propose methods to simplify polyhedra by limiting both the number of bits and contraints. The applicability of PHAVer and the effectiveness of the proposed methods are demonstrated with a navigation benchmark [7], and a tunnel diode circuit [8]. 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 abstraction between different models, and for assume-guarantee reasoning. For lack of space, the reader is referred to [9] for further details on the approach.

Earlier attempts to improve over HyTech include the use of interval arithmetic [10], which can quickly lead to prohibitively large overapproximations. An algorithm specialized on rectangular automata was proposed in [11] 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 over-approximation can become prohibitively large. An improvement was proposed in [12] 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. While most tools for timed automata use exact computations, we are not aware of tools for hybrid systems that do so apart from HyTech. 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++ [13]. Our on-the-fly overapproximation essentially performs a partitioning of the state space similar to the approach in [14]. For the simplification of polyhedra it has been suggested to use bounding boxes or oriented rectangular hulls [15]. Instead, we propose to simply drop the

[1] Not addressed are more advanced input capabilities like hierarchy, templates and directional communication labels, since we consider these easily and more appropriately handled by a GUI-frontend or editor.

least significant of the constraints, as this seems a good compromise in terms of accuracy and speed. For a survey of verification tools for hybrid automata, see [16].

In the next section, we will briefly introduce the hybrid automaton model used by PHAVer, which has a simple Input/Output structure to support compositional reasoning. 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 sum up the results and present conclusions in Sect. 5.

2 Hybrid I/O-Automata with Affine Dynamics

The theory of hybrid I/O-automata has been developed extensively by Lynch, Segala, Vaandrager and Weinberg [17]. It is a very general framework that is based on (almost) arbitrary trajectories of a set of variables, which can have different dynamic types. Since our focus is on obtaining a computable framework for compositional reasoning, we have proposed a simple concept of I/O-automata in [9], which is largely based on the hybrid automata in [1]. Given a set Var of variables, a valuation is a function $v : Var \to \mathbb{R}$. Let $V(Var)$ denote the set of valuations over Var. An *activity* is a function $f : \mathbb{R}^{\geq 0} \to V(Var)$. Let $act(Var)$ denote the set of activities over the variables in Var. A set S of activities is *time-invariant* if for all $f \in S, d \in \mathbb{R}^{\geq 0} : f_d(t) := f(t+d) \in S$. Let $\downarrow_{Var})$ be the projection onto the variables in Var.

Definition 1 (Hybrid I/O-Automaton). *[9]A* hybrid Input/Output-automaton *(HIOA)* $H = (Loc, Var_S, Var_I, Var_O, Lab, \to, Act, Inv, Init)$ *consists of the following:*

- *A finite set Loc of locations.*
- *Finite and disjoint sets of* state *and* input *variables,* Var_S *and* Var_I*, and of* output *variables* $Var_O \subseteq Var_S$*. Let* $Var = Var_S \cup Var_I$*.*
- *A finite set Lab of synchronization labels.*
- *A finite set of discrete transitions* $\to \subseteq Loc \times Lab \times 2^{V(Var) \times V(Var)} \times Loc$*. A transition* $(l, a, \mu, l') \in \to$ *is also written as* $l \xrightarrow{a,\mu}_H l'$*.*
- *A mapping* $Act : Loc \to 2^{act(Var)}$ *to time-invariant sets of activities.*
- *A mapping* $Inv : Loc \to 2^{V(Var)}$ *from locations to sets of valuations.*
- *Initial states* $Init \subseteq Loc \times V(Var)$ *s.t.* $(l, v) \in Init \Rightarrow v \in Inv(l)$*.*

In PHAVer, we deal with hybrid automata that can be analyzed using polyhedra, i.e., finite linear formulas. A *linear expression* is of the form $\sum_i a_i x_i + b$, and a *convex linear formula* is a finite conjunction of constraints $\sum_i a_i x_i + b \bowtie 0$, with $a_i, b \in \mathbb{Z}$, $x_i \in Var$ and a sign $\bowtie \in \{<, \leq, =\}$. A *non-convex linear formula*, or *linear formula*, is a finite disjunction of convex linear formulas. A *linear hybrid automaton* (LHA) [1] is a hybrid automaton in which the invariants and the

continuous transition relation are given by linear formulas over Var, and the activities are given by linear formulas over the time derivatives of the variables. If the dynamics are given by linear formulas over the derivatives and the variables, we call it an *affine hybrid automaton.* [2]

3 Reachability Analysis in PHAVer

A reachability analysis computes all states that are connected to the initial states by a run. We enhance the fixpoint computation algorithm for reachability with operators for the partitioning of locations and the simplification of sets of states described by polyhedra. The partitioning of locations is used when affine dynamics are overapproximated with LHA-dynamics, where locations are split into smaller parts to improve the accuracy. The simplification operator fulfills two purposes: Firstly, the overapproximation of sets of states with a simpler representation keeps the complexity from growing beyond computationally manageable limits. Secondly, since termination is not guaranteed for linear hybrid automata, overapproximation of the sets of states as well as the set of derivatives can be used to accelerate convergence and possibly force termination by reducing the model to a class where reachability is decidable. The challenge lies in trading speed, termination and resource consumption against the loss of accuracy. The algorithm used in PHAVer for computing the set of reachable states is shown in Fig. 1. We give a brief summary of the operators used. Let X, Y and $Y_1, \ldots, Y_z$

procedure *GetReach*
  Input: a set of initial states S_I
  Output: the set of states S_R reachable from S_I
  $(S_I, \{S_I\}) := partition\_loc(S_I, \{S_I\})$;
  $W, S_R := time\_elapse(S_I)$;
  while $W \neq \emptyset$ **do**
    $N := trans\_post(W)$;
    $(N, (S_I, S_R, W)) := partition\_loc(N, (S_I, S_R, W))$;
    $N := cheap\_difference(N, S_R)$;
    $N := union\_approx(N, S_R)$;
    $N := simplify(N)$;
    $N := time\_post(N, simplify(time\_deriv(N, Inv)))$;
    $S_R := S_R \cup N$;
    $W := N$
  od.

Fig. 1. Reachability Algorithm in PHAVer

[2] In literature, a LHA is also referred to as a *piecewise constant* HA, and an affine HA as a *linear* HA.

be arbitrary sets of states, each described by a set of convex polyhedra for each location.

Post-Operators: The operator *time_elapse*(X, Y) computes the successors of a set of states X by letting time elapse according to a set Y that attributes a set of derivatives to each location. The successors of discrete transitions are given by *trans_post*(X). A detailed description can be found in [2].

Overapproximating Operators: The operator *cheap_difference*(X, Y) computes a overapproximation of $X \setminus Y$ by returning the polyhedra in Y that are not individually contained in some polyhedra of X. The gain in speed usually far outweighs the fact that more states are iterated than necessary [2]. With *union_approx*(X, Y), the union of new states X and old states Y can optionally be overapproximated, e.g., by using the convex hull. If there are no new states for a location, the operator returns the empty set for that location. The *simplify* operator is used to reduce the complexity the representation of states by overapproximation. It can also be applied to the set of derivatives in the location. Current options in PHAVer for *simplify* include a bounding box overapproximation, limiting the number of bits used by the coefficients of constraints, and limiting the number of constraints.

Partitioning Operators: The operator *partition_loc*$(X, (Y_1, \ldots, Y_z))$ partitions the locations with states in X as described in Sect. 3.1 and maps the states in $Y_1, \ldots, Y_z$ to the new set of locations. The operator *time_deriv*(X, Y) computes the set of derivatives that any state in X might exhibit, provided that the states are confined to Y:

$$\begin{aligned} \mathit{time\_deriv}(X, Y) = \{(l, \dot{f}(t)) | \exists (l, v) \in X, f \in \mathit{Act}(l), t \in \mathbb{R}^{\geq 0} : \\ (f(0) = v \wedge \forall t', 0 \leq t' \leq t : f(t') \in Y)\} \end{aligned}$$

In the following section we give a more detailed description of the partitioning operator and its parameters.

3.1 On-the-fly Over-Approximation of Affine Dynamics

While PHAVer's computations are based on linear hybrid automata models, it also accepts affine dynamics, which are then overapproximated conservatively. The approximation error depends on the size of the location and the dynamics, so PHAVer offers to partition reachable locations during the analysis. The partitioning takes place by splitting locations recursively along user-defined hyperplanes until a minimum size is reached or the dynamics are sufficiently partitioned.

The *relaxed affine dynamics* are given by a convex linear formula for its derivatives, i.e., a conjunction of constraints

$$a_i^T \dot{x} + \hat{a}_i^T x \bowtie_i b_i, \qquad a_i, \hat{a}_i \in \mathbb{Z}^n, b_i \in \mathbb{Z}, \bowtie_i \in \{<, \leq, =\},\ i = 1, \ldots, m. \quad (1)$$

for each location. In the following, we assume the equalities to be modeled using conjuncts of pairs of inequalities. In a location *loc*, the constraints (1) are overapproximated conservatively with constraints of the form $\alpha_i \dot{x} \bowtie_i \beta_i$, $\alpha_i \in \mathbb{Z}^n, \beta_i \in \mathbb{Z}$, by finding the infimum of (1) inside the invariant $Inv(loc)$. Let

$$p/q = \inf_{x \in Inv(loc)} \hat{a}_i^T x, \qquad p, q \in \mathbb{Z}.$$

If p/q exists, the set of $\dot{x}$ that fulfill (1) is bounded by $a_i^T \dot{x} \bowtie_i b_i - p/q$, otherwise the constraint must be dropped. The linear constraint on $\dot{x}$ is then given by $\alpha_i = qa_i$, $\beta_i = qb_i - p$.

The resulting overapproximation error depends on the size of the locations and the dynamics but can be made arbitrarily small by defining suitably small locations. PHAVer does so by recursively splitting a location along a suitable hyperplane chosen from a user-provided set. The splitting is repeated in reachable locations until a certain threshold, e.g., a minimum size, is reached. 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

$$\sphericalangle(X) = \arccos \min_{x,y \in X} x^T y / |x||y|$$

and the spread of the derivatives of states X confined to states Y in location *loc*

$$\sphericalangle_{deriv}(loc, X, Y) = \sphericalangle\left(\{v | (loc, v) \in time\_deriv(X, Y)\}\right).$$

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, or the constraints are prioritized according to the spread of the derivatives in the location after the splitting.

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

$$\Delta(a_h) = \max_{x \in Inv(loc)} a_h^T x - \min_{x \in Inv(loc)} a_h^T x.$$

In PHAVer, the user provides a list of candidate normal vectors $a_{h,i}$ and the minimum and maximum slack that the hyperplanes will have in the partitioned locations, i.e.,

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

This allows the user to include expert knowledge by choosing planes and location sizes suitable for the system. The candidate hyperplanes are prioritized according to a user-controlled list of criteria. We consider the criteria to be a map

$$split\_crit : \{a^T x \bowtie b | a \in \mathbb{Z}^n, b \in \mathbb{Z}\} \times Loc \times 2^{S_H} \mapsto (\mathbb{R} \cup \infty \cup -\infty)^z$$

that attributes a z-tuple of prioritizing measures, evaluated lexicographically, to each constraint, and takes into account a set of valuations considered of interest. Two special symbols are included: ∞ voids the constraint, but it can be overruled by $-\infty$, which takes precedence over all other factors. The currently implemented measure $split\_crit(a^T x \bowtie b, loc, N)$ takes into account the set N of reachable states in the location, and offers the following choices:

1. Prioritize constraints according to their slack:
$$split\_crit_1 = \begin{cases} \Delta(a_h)/\Delta_{min,h} & \text{if} \Delta(a_h) > \Delta_{min,h}, \\ \infty & \text{otherwise.} \end{cases}$$
2. Prioritize constraints that have reachable states only on one side:
$$split\_crit_2 = \begin{cases} 1 \text{ if } \exists x, x' \in N : a^T x < b \wedge a^T x' > b \\ 0 \text{ otherwise.} \end{cases}$$
3. Prioritize constraints according to the spread of the derivatives. Discard constraint if a minimum spread $\lhd_{min}$ is reached and the slack is smaller than $\Delta_{max,h}$:
$$split\_crit_3 = \begin{cases} -\lhd_{deriv}(loc, N, Inv) \text{ if } \lhd_{deriv}(loc, N, Inv) \geq \lhd_{min} \\ \qquad\qquad\qquad\qquad \vee\ \Delta(a_h) > \Delta_{max,h}, \\ \infty \qquad\qquad\qquad\qquad \text{otherwise.} \end{cases}$$
4. Prioritize constraints according to the derivative spread after the constraint is applied:
$$split\_crit_4 = -\max\{\lhd_{deriv}(loc, N, \{(l, x) \in Inv \mid a^T x \leq b\}), \\ \lhd_{deriv}(loc, N, \{(l, x) \in Inv \mid a^T x \geq b\})\}.$$

For efficiency, the partitioning is applied on the fly as shown in the reachability algorithm of Fig. 1. The algorithms for splitting a location, and refining the location with the prioritized candidate constraints are shown in Fig. 2 and Fig. 3.

3.2 Example: Navigation Benchmark

We illustrate the reachability analysis of PHAVer with a benchmark proposed in [7]. It models an object moving in a plane, and following dynamically a set of desired velocities $v_d(i) = (sin(i\pi/4), cos(i\pi/4))^T$, $i = 0, \ldots, 7$, where i is attributed to each unit square in the plane by a given map M. 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. 4, whose maps are given by:

$$M_{\text{NAV01}} = M_{\text{NAV02}} = M_{\text{NAV03}} = \begin{pmatrix} \texttt{B}\ 2\ 4 \\ 2\ 3\ 4 \\ 2\ 2\ \texttt{A} \end{pmatrix}, M_{\text{NAV04}} = \begin{pmatrix} \texttt{B}\ 2\ 4 \\ 2\ 2\ 4 \\ 1\ 1\ \texttt{A} \end{pmatrix}.$$

procedure *SplitLocation*

Input: HIOA $H = (Loc,\ Var_S,\ Var_I,\ Var_O,\ Lab,\ \rightarrow,\ Act,\ Inv,\ Init)$,
location loc, constraint $a_i^T x \bowtie_i b_i$, splitting label τ_H,
list $\{Y_1, \ldots, Y_n\}$ of set of states of H for remapping

Output: Hybrid I/O-automaton H with split location loc

Loc**:=**$\{l \in Loc \mid l \neq loc\} \cup \{(loc, \leq), (loc, \geq)\}$**;**
$\rightarrow$**:=**$\{(l, a, \mu, l') \in \rightarrow \mid l \neq loc \wedge l' \neq loc\}$
$\cup\{(l, a, \mu, (loc, \leq)), (l, a, \mu, (loc, \geq)) \mid (l, a, \mu, loc) \in \rightarrow\}$
$\cup\{((loc, \leq), a, \mu, l'), ((loc, \geq), a, \mu, l') \mid (loc, a, \mu, l) \in \rightarrow\}$
$\cup\{(l, \tau_H, \{x' = x | x \in Var\}, l') \mid l, l' \in \{(loc, \leq), (loc, \geq)\}\}$**;**
Act**:=**$\{l \mapsto x(t) \in Act \mid l \neq loc\}$
$\cup\{(loc, \bowtie) \mapsto x(t) \mid loc \mapsto x(t) \in Act, \bowtie \in \{\leq, \geq\}\}$**;**
for $S \in \{Y_1, \ldots, Y_n\} \cup \{Inv, Init\}$ **do**
S**:=**$\{(l, x) \in S \mid l \neq loc\}$
$\cup\{((loc, \bowtie), x) \mid (loc, x) \in S \wedge a_i^T x \bowtie b_i, \bowtie \in \{\leq, \geq\}\}$
od.

Fig. 2. Splitting a location along a hyperplane

procedure *partition_loc*

Input: HIOA $H = (Loc,\ Var_S,\ Var_I,\ Var_O,\ Lab,\ \rightarrow,\ Act,\ Inv,\ Init)$,
set of investigated states N, set of candidate constraints
$Cand = \{(a_{h,1}, \Delta_{min,1}, \Delta_{max,1}), \ldots, (a_{h,m}, \Delta_{min,m}, \Delta_{max,m})\}$,
list $\{Y_1, \ldots, Y_n\}$ of set of states of H for remapping

Output: Hybrid I/O-automaton H with locations in N partitioned

for $loc \in \{l \in Loc | \exists x : (l, x) \in N\}$ **do**
do
for $i = 1, \ldots, m$ **do**
b_i**:=**$1/2 \left(\max_{x \in Inv(loc)} a_{h,i}^T x + \min_{x \in Inv(loc)} a_{h,i}^T x \right)$**;**
c_i**:=***split_crit*$(a_{h,i}^T x = b_i, loc, N)$
od;
k**:=**$\underset{i=1,\ldots,m}{\text{argmin}}\ c_i$**;**
if $\infty \notin c_k \vee -\infty \in c_k$ **then**
SplitLocation$(H, loc, a_{h,k}^T x = b_k, \tau_H, \{Y_1, \ldots, Y_n\})$
od
while k **exists and** $\infty \notin c_k \vee -\infty \in c_k$ **od**
od.

Fig. 3. Refining states with a set of candidate constraints

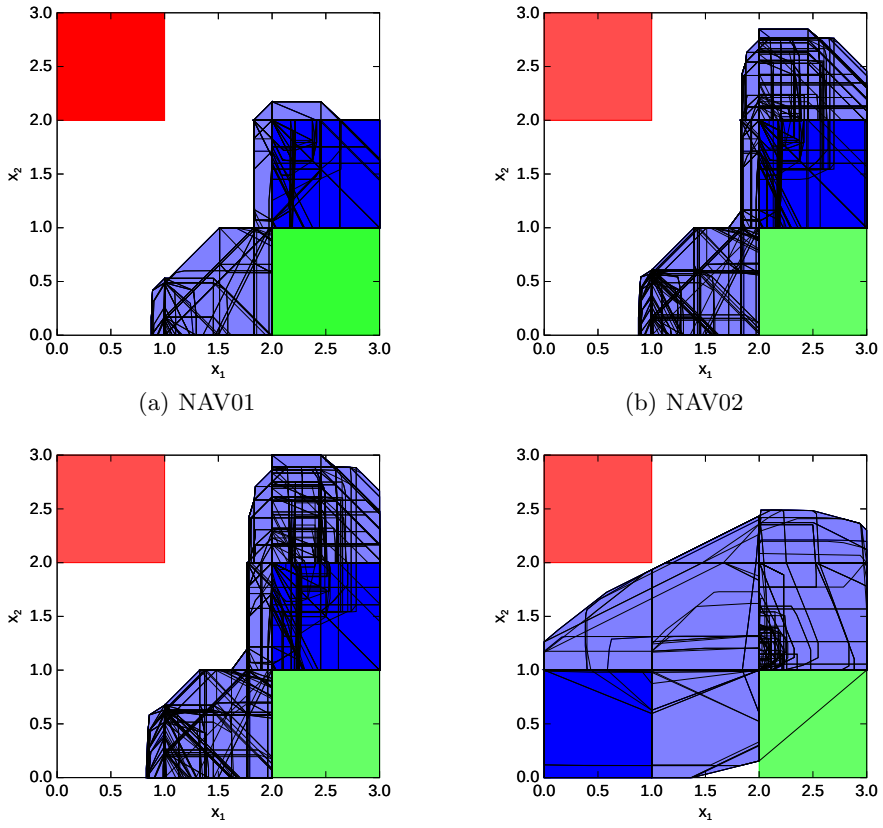
The dynamics of the 4-dimensional state vector $(x_1, x_2, v_1, v_2)^T$ are given by

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

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

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

As splitting constraints we use $\mathit{Cand} = \{(v_1, \delta_1, \infty), (v_2, \delta_2, \infty)\}$, where appropriate δ_i were established by some trial-and-error runs, and $(\mathit{split\_crit}_1)$ 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



(a) NAV01 (b) NAV02

(c) NAV03 (d) NAV04

Fig. 4. Reachable states in the x_1, x_2-plane (initial states darkest)

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 4 shows the set of reachable states computed by PHAVer as a result. Computation times and memory consumption are shown in Table 1, and were obtained on a Pentium IV, 1.9GHz with 768 MB RAM running Linux. For the instances NAV01–NAV03, the analysis was fairly straightforward, with $\delta_i = 0.5$. For the instance NAV04 we had to set $\delta_i = 0.25$, and the analysis did not terminate at first. We applied a heuristic: The convex hull was computed for the first 20 iterations for speed, then switched to normal reachability, and at iteration 40 a bounding box simplification was triggered manually. In comparison, for a predicate abstraction tool the following times were reported in [18]: For NAV01–NAV03 $34s$, $153s$ (68MB) and $152s$ (180MB), respectively, on a Sun Enterprise 3000 (4 x 250 MHz UltraSPARC) with 1 GB RAM.

Table 1. Computation times and memory requirements

Instance	Time	Memory	Iter.	Automaton Loc.	Automaton Trans.	Reachable Set Loc.	Reachable Set Polyh.
NAV01	34.73 s	62.6 MB	13	141	3452	79	646
NAV02	62.16 s	89.7 MB	13	153	3716	84	1406
NAV02[i]	41.05 s	53.7 MB	13	148	3661	84	84
NAV03	61.88 s	90.0 MB	13	153	3716	84	1406
NAV04[ii]	225.08 s	116.3 MB	45	267	7773	167	362

[i] convex hull, [ii] convex hull up to iter. 20, bounding box at iter. 40

4 Managing the Complexity of Polyhedra

A set of symbolic states is described by a linear formula, the convex sub-formulas of which define convex polyhedra, which in turn are described by a set of constraints. 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, e.g., with resets or invariants. To keep the complexity manageable, we propose the simplification of complex polyhedra in a strictly conservative fashion by limiting the number of bits, i.e., the size of coefficients, and the number of constraints. We reduce only inequalities to preserve the affine dimension of the polyhedron. 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 result-

ing hysteresis between exact computations and overapproximations gives cyclic dependencies time to stabilize.

4.1 Limiting the Number of Bits

We consider the ith constraint $a_i^T x \bowtie_i b_i$ of a polyhedron of the form $Ax+b \bowtie 0$, where a_i is a vector of the coefficients $a_{ij} \in \mathbb{Z}$ of A, $i = 1, \ldots, m$, $j = 1, \ldots, n$, $\bowtie$ is a vector of signs $\bowtie_i \in \{\leq, <, =\}$, and b is a vector of 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. The goal is to find a new constraint $\alpha_i^T x \bowtie_i \beta_i$ with coefficients α_{ij} having less than z bits, i.e., $|\alpha_{ij}|, |\beta_i| \leq 2^z - 1$, with the least overapproximation possible. Expressing the new coefficients in terms of a scaling factor $f > 0$, rounding errors r_{ij}, $|r_{ij}| \leq 0.5$ and an error r_i for the inhomogeneous term we get $\alpha_{ij} = fa_{ij} + r_{ij}$ and $\beta_i = fb_i + r_i$. There is no a-priori bound on r_i, since it depends on the new direction α_i and the other constraints that define the polyhedron. In our iterative approach we initially assume β_i to be close to fb_i. Since β_i must be rounded upwards to guarantee conservativeness, we get $|r_i| \leq 1$ as an optimistic estimate. The initial upper bounds for f are therefore given by:

$$f \leq (2^z - 3/2)/|a_{ij}|, \text{ and} \tag{2}$$

$$f \leq (2^z - 2)/|b_i|. \tag{3}$$

To predict the effects of rounding precisely is difficult and would lead to a mixed integer linear program, so we employ a heuristic algorithm, shown in Fig. 5. 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 (2),(3), 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 coefficients $\alpha_{ij} = 0$, in which case the problem is infeasible. Note that it is not guaranteed that the new polyhedron is bounded. Figure 6 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 the polyhedron, as in (c). Rounding of q yields β_i, and the polyhedron outlined in (d).

4.2 Limiting the Number of Constraints

To reduce the complexity of a polyhedron, we propose to drop constraints based on a criterion *crit* that measures the the difference between the polyhedron with and without the constraint. 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 must be replaced by two inequalities.

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 it's ith constraint. Then the difference between the points contained P and $P^{\setminus i}$ is the polyhedron $P^{\neg i} = P^{\setminus i} \cup$

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 $\alpha_i^T x \bowtie_i b_i$

$success := false$**;**
$f := min\{(2^z - 3/2)/|a_{ij}|, (2^z - 2)/|b_i| \mid j = 1, \ldots, n\}$**;**
while $\neg success$ **do**
  for $j = 1, \ldots, n$ **do** $\alpha_{ij} := round(f a_{ij})$ **od;**
  $q := \min\limits_{x \in P} \alpha_i^T x$ **;**
  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. 5. Limiting the number of bits of a constraint

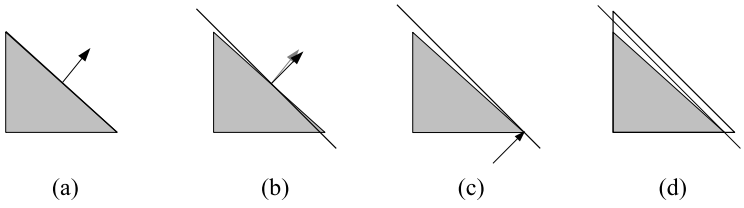


Fig. 6. Limiting the number of bits of a constraint

$\{-a_i^T x \bowtie_i -b_i\}$, where $(\bowtie_i, \bowtie_i) \in \{(<, \leq), (\leq, <)\}$, 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 a_i, 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.

We consider two general procedures of selecting the z most important out of m original constraints:

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

While deconstruction is more likely to preserve as much as possible of the original polyhedron, construction requires less iterations if $m > 2z$. 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 construction method with an angle criterion was the fastest in our experiments. The algorithm is shown in Fig. 7, where C is the set of candidate constraints and H is the set of chosen constraints. H is initialized with the set of equalities 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.

4.3 Example: Tunnel-Diode Oscillator Circuit

Consider a tunnel-diode oscillator circuit [8]. It models the current I and the voltage drop V of a tunnel diode in parallel to the capacitor of a serial RLC circuit, which are in stable oscillation for the given parameters. The state equations are given by

$$\begin{aligned} \dot{V} &= 1/C(-I_d(V) + I), \\ \dot{I} &= 1/L(-V - 1/G \cdot I + V_{in}), \end{aligned}$$

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;**
$H := \{a_k^T x \bowtie_k b_k \mid k = \text{argmin}_k(\max_j |a_{kj}|)\} \cup \{a_i^T x \bowtie_i b_i | \bowtie_i \in \{=\}\}$**;**
$C := P \setminus H$**;**
while $(|C| > 0 \wedge (|H| < z \vee (\mathit{bounded}(P) \wedge \neg \mathit{bounded}(H)))$ **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.

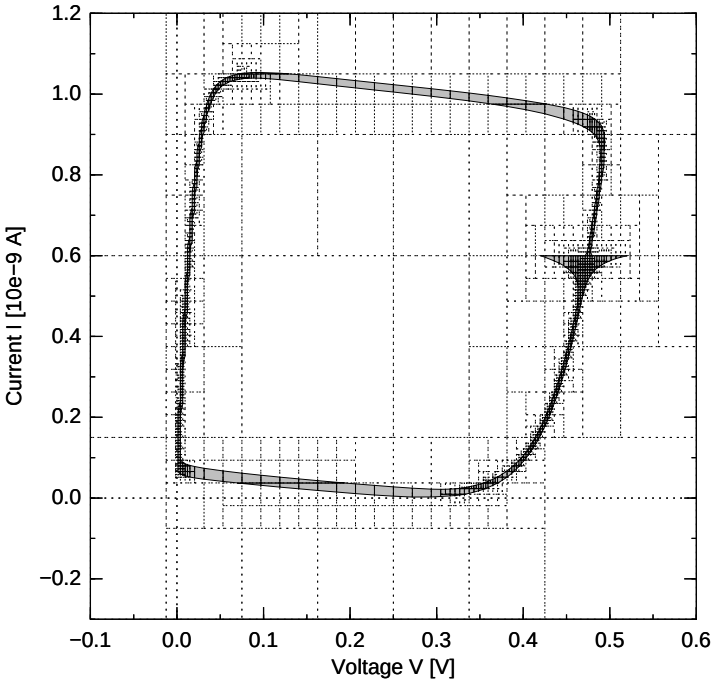
Fig. 7. Reconstructing a polyhedron with a limited number of constraints by angle prioritization

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

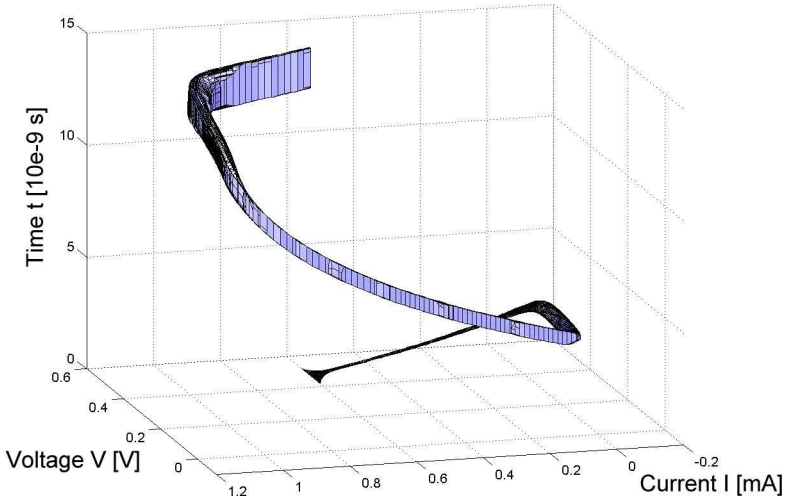
$$I_d(V) = \begin{cases} 6.0105V^3 - 0.9917V^2 + 0.0545V & \text{if } V \leq 0.055, \\ 0.0692V^3 - 0.0421V^2 + 0.004V + 8.9579e{-}4 & \text{if } 0.055 \leq V \leq 0.35, \\ 0.2634V^3 - 0.2765V^2 + 0.0968V - 0.0112 & \text{if } 0.35 \leq V. \end{cases}$$

The dynamics were approximated with LHA, similar to the approach in Sect. 3.1. Figure 8(a) shows the convex hull of the reachable states starting from $V \in [0.42V, 0.52V]$, $I = 0.6mA$. It also shows the invariants (dashed) generated by the partitioning algorithm using constraints $\mathit{Cand} = \{(V, 0.7/128, 0.7/16), (I, 1.5/128, 1.5/16)\}$, i.e., max. 128 partitions in both directions, and splitting criterion $(\mathit{split\_crit}_3, \mathit{split\_crit}_1)$ with $\lhd_{min} = \arccos(0.99)$. The analysis with PHAVer took $52.63s$ and 55MB RAM, with the largest coefficient taking up 7352 bits and at most 7 constraints per polyhedron.

A stopwatch was added to the system to measure the cycle time, i.e., the maximum time it takes any state to 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. 8(b) is only possible with limits on both. We compare the exact analysis for constraints $\mathit{Cand} = \{(V, 0.7/32, 0.7/16), (I, 1.5/32, 1.5/16)\}$ with an analysis limiting the bits to 16 when a threshold of 300 bits is reached, and a limit of 32 constraints at a threshold of 56. Figures 9(a) and 9(b) show 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 979s (210MB) when exact, and 79s (39.6MB) when limited. At a more than tenfold increase in speed, the overapproximation is negligible and results in a cycle time estimate that is only 0.25 percent larger.

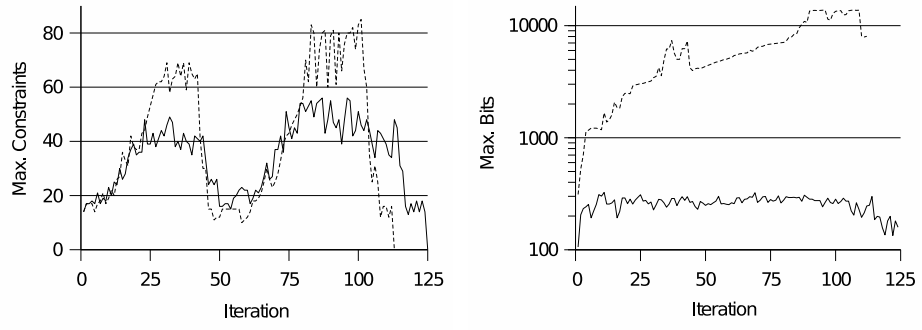


(a) V-I-Plane, invariants dashed



(b) Clocked

Fig. 8. Reachable states of Tunnel Diode Circuit



(a) Number of bits (b) Number of constraints

Fig. 9. Clocked Tunnel Diode Circuit, exact (dashed) and with limits on bits and constraints (solid)

5 Conclusions

PHAVer, a new tool for verifying safety properties of linear hybrid automata, provides exact, robust arithmetic, on-the-fly overapproximation of affine dynamics, and supports compositional and assume/guarantee-reasoning. To manage the complexity of the underlying polyhedral computations, we proposed methods for conservatively limiting the number of bits and constraints that describe a polyhedron. Experimental results for a navigation benchmark and a tunnel diode circuit demonstrated the effectiveness of the approach. Future research will focus on heuristics for guaranteeing termination, adapting the partitioning further to the dynamics and improved search algorithms. PHAVer is available at `http://www.cs.ru.nl/~goranf/`.

Acknowledgements. The author is most grateful for the numerous inspiring discussions with Prof. Bruce Krogh, whose insightful guidance was indispensable in this work, 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.

References

1. Henzinger, T.A.: The theory of hybrid automata. In: Proc. IEEE Symp. Logic in Computer Science, LICS'96, New Brunswick, New Jersey, 27-30 July 1996, IEEE Computer Society (1996) 278–292
2. Ho, P.H.: Automatic Analysis of Hybrid Systems. PhD thesis, Cornell University (1995) Technical Report CSD-TR95-1536.

3. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HYTECH: A model checker for hybrid systems. Int. Journal on Software Tools for Technology Transfer **1** (1997) 110–122
4. Henzinger, T.A., Preussig, J., Wong-Toi, H.: Some lessons from the hytech experience. In: Proceedings of the 40th Annual Conference on Decision and Control (CDC'01), IEEE Press (2001) pp. 2887–2892
5. 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.: SPIN. Volume 2057 of LNCS., Springer (2001) 296–303
6. 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. Int. Symp. Volume 2477 of LNCS., Springer (2002) 213–229
7. Fehnker, A., Ivancic, F.: Benchmarks for hybrid systems verification. In Alur, R., Pappas, G.J., eds.: HSCC'04. Volume 2993 of LNCS., Springer (2004) 326–341
8. 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), Nov. 7–11, 2004, San Jose CA (USA). (2004)
9. Frehse, G., Han, Z., Krogh, B.H.: Assume-guarantee reasoning for hybrid i/o-automata by over-approximation of continuous interaction. In: Proc. IEEE Conf. Decision and Control (CDC'04), Dec. 14–17, 2004, Atlantis, Bahamas. (2004)
10. 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.: HSCC. Volume 1790 of LNCS., Springer (2000) 130–144
11. Preussig, J., Kowalewski, S., Wong-Toi, H., Henzinger, T.A.: An algorithm for the approximative analysis of rectangular automata. In: Proceedings of the Fifth International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT). Number 1486 in LNCS, Springer-Verlag (1998) 228–240
12. 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).
13. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: Hytech: the next generation. In: Proc. IEEE Real-Time Systems Symp. (RTSS'95), IEEE Computer Society (1995) 56–65
14. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: Algorithmic analysis of nonlinear hybrid systems. IEEE Transactions on Automatic Control **43** (1998) 540–554
15. Stursberg, O., Krogh, B.H.: Efficient representation and computation of reachable sets for hybrid systems. In Maler, O., Pnueli, A., eds.: HSCC'03. Volume 2623 of LNCS., Springer (2003) 482–497
16. 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 Conference on Decision and Control (CDC'01). (2001)
17. Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid I/O automata. Information and Computation **185** (2003) 105–157
18. Ivancic, F.: Modeling and Analysis of Hybrid Systems. PhD thesis, University of Pennsylvania, Philadelphia, PA (2003)