Combining Control and Data Abstraction in the Verification of Hybrid Systems

Xavier Briand and Bertrand Jeannet

***Abstract*—This paper addresses the verification of hybrid systems built as the composition of a discrete software controller interacting with a physical environment exhibiting a continuous behavior. The goal is to attack the problem of the combinatorial explosion of discrete states that may happen if a complex software controller is considered. It proposes as a solution to extend an existing abstract interpretation technique, namely dynamic partitioning, to hybrid systems described in a symbolic formalism. Dynamic partitioning allows us finely tune the tradeoff between precision and efficiency in a reachability analysis. It shows the effectiveness of the approach by a case study that combines a nontrivial controller specified in the synchronous dataflow programming language Lustre with its physical environment.**

***Index Terms*—Abstract interpretation, hybrid systems, logico-numerical properties, synchronous languages verification.**

I. Introduction

HYBRID SYSTEMS have the particularity to combine a discrete behavior, specified with traditional test and assignment operations, with a continuous behavior, specified by the mean of differential equations or inclusions. They primarily allow us to model a physical environment ruled by physical laws, which may be either purely continuous, or mixing discrete and continuous aspects (like a bouncing ball). Hybrid systems are also particularly well-suited to model the behavior of a software controller that exhibits a discrete behavior, interacting with a physical environment ruled by physical laws. Such systems have recently been given the name of *cyberphysical systems*.

This paper targets specifically this latter case, and is motivated by the problem of analysing a LUSTRE synchronous program interacting with a physical environment. LUSTRE [23] is a domain-specific dataflow language for programming control-command systems that periodically sample inputs from their environment, compute outputs and move to a new internal state. When verifying properties on such programs, it is mandatory to take into account a reasonably accurate model of their physical environment, as these programs make assumptions on their environment. For instance, a speed regulation system implicitly assumes that when it orders an acceleration, the speed should increase. The resulting combination of a program and its environment is a hybrid system, that has usually the specificity of having a large discrete state-space, inherited from the state-space of the controller program.

Verification of hybrid systems focuses, however, mainly on the analysis of the continuous behavior; the values of *non-numerical* variables are encoded in the control structure. Invariants for *numerical* variables are then defined for *each* corresponding control point, as illustrated in Fig. 1(a). This may result in a combinatorial explosion of the number of control points, a well-known problem in the verification of finite-state systems. Moreover, compared to the case of purely discrete systems, hybrid systems make this combinatorial explosion even more difficult to tackle because invariants are more complex properties (e.g., convex polyhedra) rather than a boolean value (indicating that a state is or is not reachable).

The aim of this paper is to tackle this combinatorial explosion problem by extending the principle of *dynamic partitioning* to hybrid systems [29], [30]. The initial motivation for this technique, based on abstract interpretation, was to apply linear relation analysis [16], [25] to dataflow synchronous programs manipulating Boolean and numerical variables. The idea is to consider more general and less detailed control structures, such as the one depicted in Fig. 1(b). Fig. 1 illustrates the less precise invariants obtained by the analysis if we merge the locations according to the property $\mathsf{b0} = \mathsf{b1}$.

The term "dynamic" refers to the ability of *incrementally refining* such a control structure in order to reach a sufficient precision for the verification goal. The refinement is performed in conjunction with a combination of forward and backward analyses, so that only states that potentially belong to a counter-example are considered in the refinement process.

A. Existing Approaches for Verifying Hybrid Systems

Since the first papers on the verification of such systems [2], [24], [27], the approaches based on the use of convex polyhedra and on the resolution of fixpoint computations are still of interest. Convex polyhedra are indeed able to *infer* subtle relationship between the variables of a system [15]. An obvious limitation of convex polyhedra is that they cannot provide good approximations of non convex invariants. The HYTECH tool [26] solves this problem by using unions of convex polyhedra, but it results a possibly non-terminating (and more costly) analysis, unlike the abstract interpretation approach of [25]. As an alternative, [5] and [6] suggest approximating subsets of $\mathbb{R}^n$ with unions of hypercubes, using

Manuscript received December 16, 2009; revised April 10, 2010; accepted May 19, 2010. Date of current version September 22, 2010. This paper was recommended by Associate Editor R. Bloem.

The authors are with INRIA-Grenoble, Rhône-Alpes, Montbonnot Saint Martin 38334, France (e-mail: xav.briand@yahoo.fr; bertrand.jeannet@inria.fr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2010.2066010

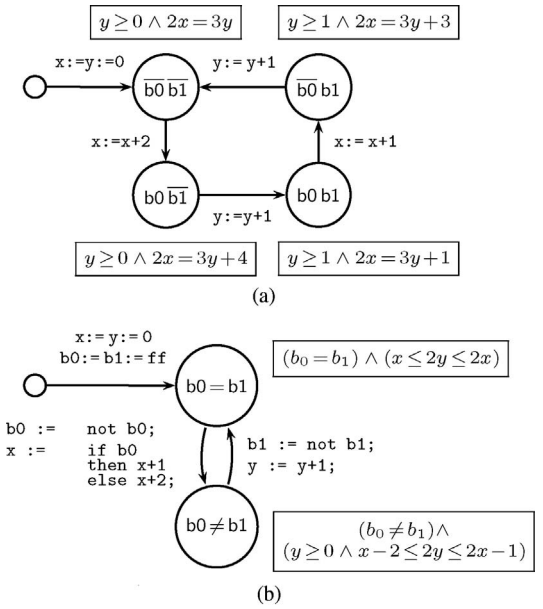

Fig. 1. Associating invariants to control location. The boxes formula give the invariant computed for each location with a linear relation analysis. (a) Fully explicit control structure. (b) Partially explicit control structure.

efficient algorithms. A strength of this method is the ability to have canonical representation of non-convex sets and to handle more complex differential equations. However, a combinatorial explosion of the numbers of hypercubes quickly arises. Ellipsoid methods have also been proposed [32]. Presented as a successor of HYTECH, the PHAVER tool offers sophisticated refinement techniques for representing non-convex invariants by unions of polyhedra and for approximating on-the-fly linear differential equations by simpler piecewise-constant polyhedral inclusions that can be treated directly with convex polyhedra. Convergence of computations can be achieved by various heuristics.

As mentioned in [17], HYTECH and PHAVER fail when dealing with large discrete space because only continuous behaviors are treated symbolically. [17] proposes a fully symbolic technique based on backward (greatest) fixpoint computation, in which sets of states are represented *exactly* with a variant of Boolean circuits mixing Boolean variables *and* linear constraints. The technique does not guarantee termination (on unbounded time intervals), and relies on a sophisticated semi-canonical representation rather than on approximations to address the efficiency issue. In the case of discrete systems, [11] adopts a similar approach by combining binary decision diagrams (BDDs) and Presburger formulas in disjunctive normal forms.

Methods based on predicate abstraction will be discussed later in Section IV.

All the above-mentioned methods are based on reachable state exploration. Another approach, mostly applied to discrete systems until recently, exploits the power of modern satisfiability (SAT) and satisfiability modulo theory (SMT) solvers to perform two different tasks:

1) finding counter-examples of some length $k \leq K$, by encoding this problem as a (generalized) SAT problem;
2) proving properties by induction of order k, which consists of proving that if a property is true for any k consecutive execution steps, it is true for the step $k+1$ (in addition to the corresponding base case that considers the initial states).

Reference [18] applies (1), in the particular case of hybrid systems with *rectangular* constant inclusion, which is less general than our model. Technique (2) is not complete, in particular because it does not take into account reachability information, but it can be very efficient in practice (see [22] for its application to discrete LUSTRE programs). We are not aware of an application of k-induction to hybrid systems.[1]

B. Contribution

We present here a reachable state analysis that combines efficiently discrete and continuous behavior for the verification of hybrid systems obtained as the composition of a physical environment and a software controller, as illustrated in Section II.

To implement our idea, we need a higher-level model for hybrid systems.

1) We should be able to embed dataflow LUSTRE program in our model.
2) We also need a more symbolic model, as a key point of our technique is to handle symbolically both the discrete and the continuous part of the state-space.

From a specification point of view, we thus propose in Section III a flexible model for hybrid systems which is more symbolic in a number of aspects. Some usual constraints are relaxed, like the requirement that invariants and guards should be convex. One can also combine freely numerical and *non-numerical* variables in formulas, as in LUSTRE programs, and we propose an original way to specify more concisely continuous behaviors, in particular by specifying with a single formula both continuous invariants and constraints on derivatives.

Section IV reminds the principles of dynamic partitioning developed in [29]. In this context, Section V extends these principles to hybrid systems, by defining a suitable method for computing *continuous* post and preconditions induced in a partitioned abstract domain. Integrated in our tool NBAC, this extension allows us to inherit the features of dynamic partitioning for hybrid systems.

We last illustrate in Section VI with experiments the potential of our approach. We first performed various analyses on the example presented in Section II, and we then tackled a very detailed model of the famous steam-boiler specification of Abrial [1]. Such a system could hardly be handled without treating symbolically the discrete state-space, due to the complexity of its discrete behavior.

Our original contributions are located in Sections III, V, and VI. This paper is a journal version of [10], with some material considered as not essential removed and other material explained with more details and examples.

[1]Unless they are abstracted as discrete systems.

II. Connecting a LUSTRE Program to a Hybrid Environment

Fig. 2 shows a model of a system composed of a LUSTRE disk controller interacting with its physical environment, namely a disk motor device (hybrid system). Fig. 2(a) depicts the behavior of the disk motor. It emits its speed on the input channel, and obeys to the command received with the output channel, but may diverge slightly from the ideal behavior (parameter ϵ).

Fig. 2(b) depicts the disk controller. It receives the speed of the disk and emits appropriate signals plus and minus to maintain the motor speed within a specified range (here the interval $[8, 12]$). This program is embedded in a hybrid automaton which receives the speed on the channel input, computes the reaction of the LUSTRE controller and stores the computed outputs in internal variables, before emitting them on channel output.

The process of Fig. 2(c) forces the synchronizations on channels input and output to take place in the right order, within specific time intervals. We model thus the reaction time of the controller, and variations in the sampling period and the reaction time. The parameters are $\epsilon, m, M, IM, Om, OM$. t is a local clock used for measuring delays, and pc is the implicit program counter that encodes the three locations (we dot not have explicit control location in our formal model presented in Section III, although we use them in graphical descriptions).

Fig. 2(d) depicts the property observer. d is the time during which the speed has been outside the interval [8, 12]. We want to check if $d \leq 8$ always holds. As initially the speed is 0, the controller should put the motor speed in the desired range quickly enough, and control it properly afterward.

III. Symbolic Hybrid Automata (SHA)

We introduce in this section a symbolic model for hybrid systems, which allows us to manipulate symbolically locations and location invariants, and which is actually used in Fig. 2. Thus, we can avoid the state-explosion problem in the specification and delay this issue to the analysis.

From a computational point of view, the expressiveness of SHA is identical to linear hybrid automata. However, from a specification language point of view, SHA are more expressive in so far as they allow much more compact descriptions.

A. SHA Model

A SHA $H = (V, Init, \Delta_H, D_H)$ is defined by a set V of variables evolving from initial state $Init$ according either to *discrete and instantaneous changes* (with assignments) specified by Δ_H, or to *continuous evolutions* specified with a global, *conditional differential inclusion* D_H. More precisely:

1) V is a finite set of variables partitioned into variables of any type in V_Q (subject to discrete behaviors only) and real-valued variables in V_X. Q (resp. X) denotes the set of valuations of variables in V_Q (resp. V_X). $S = Q \times X$ is the set of states of the system and $Init \subseteq S$ the initial states.

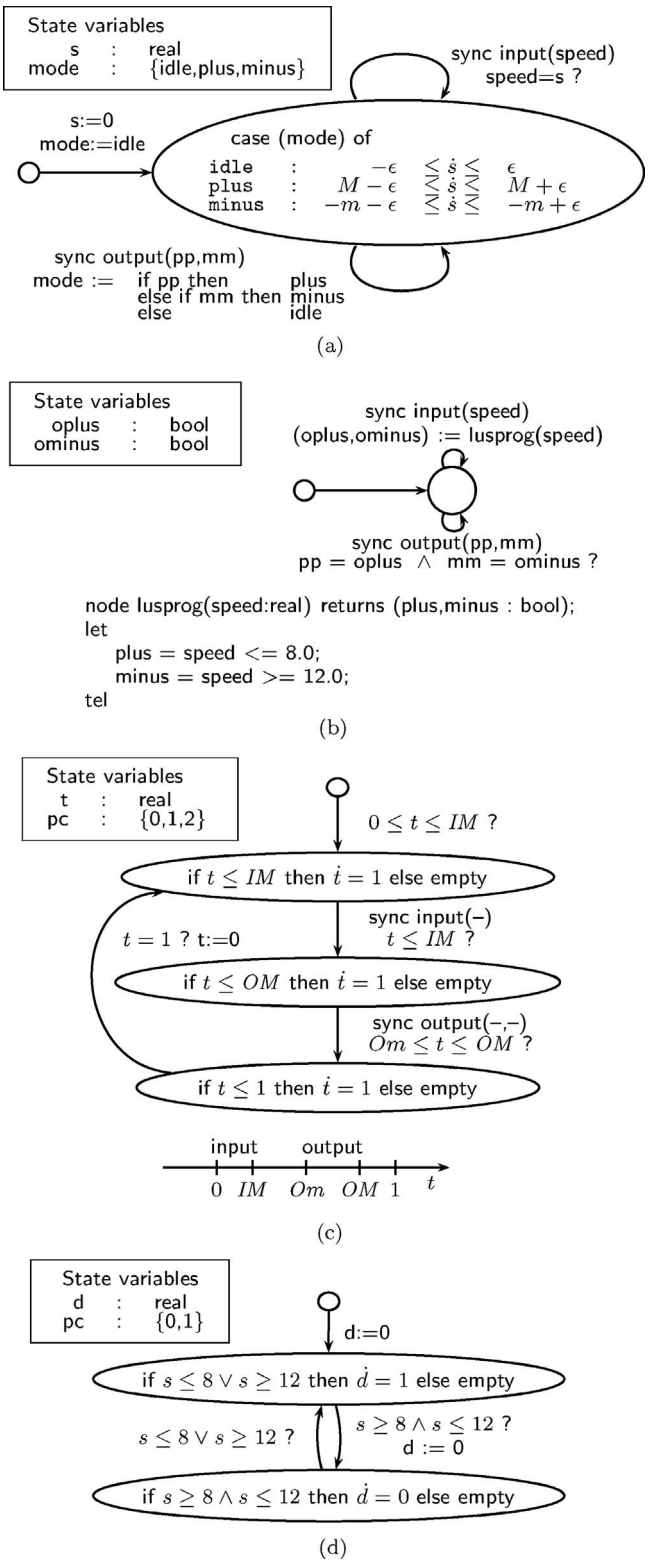

Fig. 2. Disk controller connected to its environment, and a property observer. (a) Environment: disk motor device emits the speed of the disk on channel input and reacting to the controller (channel output). (b) LUSTRE controller embedded in a process. It emits plus and minus commands to the physical device. (c) Scheduler defining the scheduling above. (d) Property observer on the physical environment: d counts the delay for which the speed s has been outside the desired range.

$V_Q = \{\mathsf{mode}\}$, $V_X = \{\mathsf{s}\}$, $Init = (\mathsf{s}=0 \wedge \mathsf{mode}=idle)$
$\Delta = \{T_1, T_2\}$ with
$T_1 = (\sigma = \mathsf{input}, \quad p = \mathsf{speed}, \quad G = (speed = s), \ A = id)$
$T_2 = (\sigma = \mathsf{output}, \ p = (\mathsf{pp}, \mathsf{mm}), \ G = true, \quad A = \ldots)$
$D_H =$ if $\mathsf{mode}=idle$ then $-\epsilon \le \dot{s} \wedge \dot{s} \le \epsilon$
else if $\mathsf{mode}=plus$ then $M-\epsilon \le \dot{s} \wedge \dot{s} \le M+\epsilon$
else $-m-\epsilon \le \dot{s} \wedge \dot{s} \le -m+\epsilon$

Fig. 3. Formal definition of the SHA of Fig. 2(a).

2) Discrete changes of the system are defined by a finite set Δ_H of transitions. In a *discrete transition* $\delta = (\sigma, p, G, A) \in \Delta_H$:
   a) σ is a *channel* carrying a tuple of *communication parameters* $p = \langle p_1, \ldots, p_k \rangle$;
   b) the *guard* $G \subseteq S \times P_\sigma$ is a predicate on the variables and the communication parameters (P_σ denotes the set of valuations of parameters);
   c) the *assignment* $A : S \times P_\sigma \to S$ defines the evolution of the values of variables during the discrete transition.

   For instance, in Fig. 2(a) the transition labeled by the output channel is defined by σ = `output`, $p = \langle \mathtt{pp}, \mathtt{mm} \rangle$, G = `true`, and A assigns the variable `mode` depending on the communication parameters `pp`, `mm`, and leaves the other state variables unchanged.
3) The continuous evolution is defined by a function $D_H : Q \times X \to 2^X$ that associates with each discrete state $q \in Q$ a *differential inclusion* $\dot{x}(t) \in D_H(q, x(t))$. Examples of *differential inclusions* D_H are [if b then dx = dy + 1 else dx = dy] or [if x> = 2 then dx = dy + 1 else dx = dy].

For instance, Fig. 3 gives the formal definition of the SHA depicted in Fig. 2(a).

We detail some key differences of our model compared to the "classical" model of hybrid automata most often considered in the verification community [2], [19].

1) It does not restrict *a priori* the types of the discrete variables in V_Q, as the software controller part may manipulate variables of various types (integers, floating-point numbers, and so on).
2) It does not use explicit locations or control points; these need to be encoded with *program counter* variables. This enables us to play with the control structure, as shown in Fig. 1.
3) The constraints D_H on the derivatives of continuous variables are a function on all state variables rather than a function on discrete locations only.
4) The classical model uses the notion of *location invariant* to partially solve the non-determinism between the time elapse and the firing of a discrete transition. A location invariant is a predicate on continuous variables that must be true for the time to elapse, and that forces a discrete transition to be taken when it becomes false. For instance, in Fig. 5(a) the constraint $1 \le x \le 3$ is a location invariant that must be true for x to evolve according to the differential equation $\dot{x} = x$. In our model location invariants are implicitly defined by (the guards of) the differential inclusion function D_H, as explained below.

Let us illustrate by a few examples points 3 and 4. The differential inclusion [if b then dx = dy + 1 else dx = dy] uses the Boolean variable `b` to encode two traditional locations with the universe as invariant [see also Fig. 2(a)]. The differential inclusion [`if x> = 2 then dx = dy + 1 else dx = dy`] specifies two traditional locations with invariants $x \ge 2$ (resp. $x < 2$) and differential inclusions $\dot{x} = \dot{y}+1$ (resp. $\dot{x} = \dot{y}$), with an implicit discrete transition between them when $x = 2$; trajectories may cross the frontier of the two connex regions. [if x <= 3 then dx = 1 else empty] is our way to specify a traditional location with differential equation $\dot{x} = 1$ and invariant $x \le 3$; when $x > 3$, there are no possible valuations for the derivatives ($D_H = \emptyset$), hence the time cannot elapse any more.

SHAs can be composed in parallel and communicate by *rendez-vous* on valued channels. When a synchronization takes place on a channel $\sigma(p)$, all the involved SHAs should agree on the value of the communication parameters p. The corresponding product, used in Fig. 2, is classical and is not detailed here. Notice that once SHAs are composed in parallel, in the resulting product SHA the synchronizations on valued channels are still meaningful, because they introduce "input" communication parameters that may be constrained by the guard and used in the assignment of a transition. Stated differently, they allow us to model opened systems receiving (possibly constrained) inputs from their environments.

1) *Syntax of Expressions:* In our verification tool, functions involved in the definition of hybrid automata are functions that can be put under the form defined by the grammar of Fig. 4. For instance

> dx1 = (if b1 then 1 else 0) and dx2 = (if b2 then 1 else 0)

can be rewritten as

> if b1
> then (if b2 then dx1 = dx2 = 1 else dx1 = 1 and dx2 = 0)
> else (if b2 then dx1 = 0 and dx2 = 1 else dx1 = dx2 = 0)

which belongs to this grammar.

Such functions are internally represented with binary decision diagrams (multiterminal BDDs, MTBDDs, [9]) built on atomic decisions (Boolean variables or linear constraints) and elementary functions, as in [28]. For elementary differential inclusions, denoted in Fig. 4 by the non-terminal "⟨diffcons⟩," in this paper we consider only linear hybrid automata, that is, hybrid systems with constant and closed differential inclusion (conjunction of non-strict linear constraints on derivatives, like $1 \le \dot{x} + 2\dot{y} \le 3$). This excludes affine differential equations of the form $\dot{x} = x$. They can however be approximated by piecewise constant differential inclusions. Notice that a few tools like PHAVer internally perform such approximations.

B. Behavior of SHA

The *run* of a SHA H is composed of a succession of discrete and continuous transitions. The global transition relation is $\to = \to_c \cup \to_d$ is defined as follows.

$\langle$lincons$\rangle$::=$\langle$linexpr$\rangle$ ("><0"|">=0") (linear constraint)
$\langle$atom$\rangle$::=$\langle$boolvar$\rangle$ | $\langle$lincons$\rangle$ (atomic condition)
$\langle$diffcons$\rangle$::=$\langle$lincons$\rangle$ (**and** $\langle$lincons$\rangle$)* (diff. constraint)
$\langle$boolexpr$\rangle$::=$\langle$true$\rangle$ | $\langle$false$\rangle$
| **if** $\langle$atom$\rangle$ **then** $\langle$boolexpr$\rangle$ **else** $\langle$boolexpr$\rangle$
$\langle$numexpr$\rangle$::=$\langle$linexpr$\rangle$
| **if** $\langle$atom$\rangle$ **then** $\langle$numexpr$\rangle$ **else** $\langle$numexpr$\rangle$
$\langle$diffexpr$\rangle$::=$\langle$diffcons$\rangle$
| **if** $\langle$atom$\rangle$ **then** $\langle$diffexpr$\rangle$ **else** $\langle$diffexpr$\rangle$

Fig. 4. Syntax of normalized expressions in SHA.

1) *Discrete Transition Relation:* The discrete transition relation $\rightarrow_d \subseteq S \times S$ induced by Δ_H is defined by

$$\frac{(\sigma, p, G, A) \in \Delta_H,\ \ v_p \in P_\sigma,\ \ (s, v_p) \in G \wedge s' = A(s, v_p)}{s \rightarrow_d s'}.$$

2) *Continuous Transition Relation:* We denote by $\mathcal{F}_T$ the set of functions $f : [0, T] \rightarrow X$ piecewise $\mathcal{C}^1$, i.e., there is a finite sequence $T_0 = 0 < T_1 < \ldots < T_n = T$ such that f is continuously differentiable on $]T_i, T_{i+1}[$ and has a left limit in T_i and a right limit in T_{i+1}. The continuous transition relation $\rightarrow_c \subseteq S \times S$ induced by the conditional differential inclusion D_H is defined by

$$\frac{q \in Q,\quad T \geq 0, f \in \mathcal{F}_T,\quad f(0) = x \wedge f(T) = x' \quad \forall i < n,\ \forall t \in]T_i, T_{i+1}[\colon \dot{f}(t) \in D_H(q, f(t))}{s = (q, x) \rightarrow_c s' = (q, x')}. \quad (1)$$

Remark that if $D_H(s) = \emptyset$, there is no possible continuous transition from s, so the time cannot elapse any more, and only discrete transitions can possibly be taken. This allows to implement the traditional notion of invariants as explained in Section III-A. The *linear hybrid automata* defined in [2] are SHA where the valuations of discrete variables Q are the locations, and where $D_H(q, x) = \text{if } x \in I_q \text{ then } C_q \text{ else } \emptyset$, with the invariant I_q and condition on derivatives C_q are convex polyhedra.

We introduce also few additional notations, used below for reachability analysis. Given a discrete state q we call T-trajectories the functions $\nu : [0, T] \rightarrow S$ such that $\nu(t) = [q, f(t)]$ and we denote by $\mathcal{T}_T$ the set of T-trajectories. The postcondition and precondition operators $post, pre : 2^S \rightarrow 2^S$ are defined as

$$post(X) = \{s' \mid s \in X \wedge s \rightarrow s'\} \quad (2)$$
$$pre(X) = \{s \mid s' \in X \wedge s \rightarrow s'\}. \quad (3)$$

For a monotone function $F : L \rightarrow L$ where L is a lattice, $\text{lfp}(F)$ denotes the least solution of the fixpoint equation $X = F(X)$. States reachable for an initial set $I \subseteq S$ or coreachable from a final set $F \subseteq S$ can be characterized by the following fixpoint equations on the lattice $(2^S, \subseteq)$:

$$reach(I) = \text{lfp}(\lambda X.I \cup post(X)) \quad (4)$$
$$coreach(F) = \text{lfp}(\lambda X.F \cup pre(X)). \quad (5)$$

As we focus in this paper only on (co)reachability properties, we are not concerned with some pathological behavior w.r.t. liveness properties such as deadlocks or Zeno runs (i.e., runs with an unbounded number of discrete transitions within a bounded time interval). Our analysis method still returns sound results in such cases.

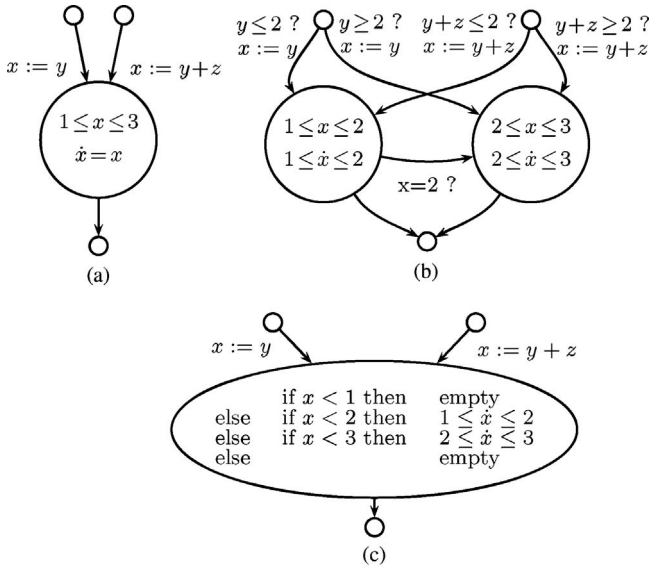

Fig. 5. Hybrid automaton and its abstraction in two different models. (a) Automaton. (b) Classical abstraction [26]. (c) Abstraction with SHA.

C. Discussion

Although the primary motivation for our symbolic model is to implement the ideas developed in the introduction regarding symbolic verification techniques, we think it is a very interesting model for its expressiveness and compactness.

For instance, the automaton of Fig. 5(a) can be abstracted by the automaton of Fig. 5(b). This requires however the duplication of discrete transitions, sometimes with guards modifications (incoming transitions), whereas our model enables a more straightforward specification of the abstraction [Fig. 5(c)]. In the same way, in Fig. 2(a), we do not need to specify explicitly the discrete transitions between the different continuous modes defined according to the value of variable mode. Fig. 2(d) illustrates the ability to specify with the same formula both the location invariant and the constraints on derivatives, and the possibility to have non-convex invariants.

In this respect, our SHA model is higher-level than the classical models considered in [2] and in the HYTECH or the PHAVER verification tools. However, it is still a mathematical model in the sense that it remains automaton-based and it does not offer modeling constructs such as the χ hybrid process algebra [8], [38].

If we look now to models mainly dedicated to *simulation*, like SIMULINK [35] or MODELICA, [34] the picture is different. On the one hand, in those models both the discrete and the continuous behavior may be specified in a very general way, using for instance C functions. On the other hand, simulation models forbid (implicit) non-deterministic choices between leaving the time elapse or firing discrete transitions. They thus replace the notion of (location) invariant by the notion of *zero-crossings*, which are numerical expressions given to

the numerical integration solver. The solver will stop the integration process as soon as one of these expressions crosses zero from below and changes its sign. When this happens, the solver gives the hand back to the discrete simulator, which then fires the discrete transition guarded by the activated zero-crossing.

For verification purposes however, location invariants or guarded differential inclusions as in our case are more adequate for symbolically computing the effect of time elapse.

IV. Abstract Interpretation and Dynamic Partitioning

We want to check an invariance property on a hybrid system, or equivalently to show that some states cannot be reached. If $BAD \subseteq S$ denotes such states, we expect

$$reach(Init) \cap BAD = \emptyset \quad \text{or} \quad coreach(BAD) \cap Init = \emptyset.$$

As the sets $reach$ and $coreach$ are not computable for the considered systems, we will use approximation techniques of [29] based on abstract interpretation [14].

A. Base Abstract Domain

The idea of abstract interpretation is to replace the powerset of states $(2^S, \subseteq)$ partially ordered by inclusion [on which (4) and (5) are defined] by a simpler abstract lattice $(A, \sqsubseteq)$ in order to achieve reasonable performance of the resulting abstract analysis, without being too imprecise either. An abstract domain may be seen as a class of geometrical shapes that enjoy nice properties w.r.t. their expressiveness and the complexity of the operations on them. The meaning of an *abstract value* $a \in A$ is given by a *concretization function* $\gamma : A \to 2^S$ such that $a_1 \sqsubseteq a_2 \Leftrightarrow \gamma(a_1) \subseteq \gamma(a_2)$.

For instance, if $S = \mathbb{R}^m$, one can replace 2^S by the *convex polyhedra* abstract domain $\mathsf{Pol}(\mathbb{R}^m)$. As convex polyhedra are just specific subsets of $\mathbb{R}^m$, γ is just the identity. The abstraction comes from the fact that the set union $\cup$ which is the least upper bound of the lattice $(2^S, \subseteq)$ is approximated by the convex hull $\sqcup_{\mathcal{C}}$ which is the least upper bound of the lattice $(\mathsf{Pol}(\mathbb{R}^m), \subseteq)$.

In our particular case, a *concrete value* is a subset of the state-space $S \simeq \mathbb{B}^n \times \mathbb{R}^m$ (enumerated variables $\times$ numerical variables). Here we make the choice that an *abstract value*

$$a = (B, P) \quad \in \quad A = 2^{\mathbb{B}^n} \times \mathsf{Pol}(\mathbb{R}^m) \tag{6}$$

is the conjunction of a Boolean formula B (represented with a BDD) and an m-dimensional convex polyhedron P. The concretization function $\gamma : A \to 2^S$ and the least upper bound $\sqcup$ are defined by

$$\gamma(B, P) = \{(b, x) \in S \mid b \in B \wedge x \in P\} \tag{7}$$

$$(B_1, P_1) \sqcup (B_2, P_2) = (B_1 \vee B_2, P_1 \sqcup_{\mathcal{C}} P_2). \tag{8}$$

Such an abstract domain forgets the relations between variables of different types, which is a quite rough approximation. For instance, one cannot represent the relation $(b_0 \Leftrightarrow x_0 \geq 0)$, because $(b_0, x_0 \geq 0) \sqcup (\neg b_0, x_0 < 0) = (\mathsf{true}, \mathbb{R})$ according to (8).

An alternative could be to consider the much more precise abstract domain

$$A' = \mathbb{B}^n \to \mathsf{Pol}(\mathbb{R}^m) \tag{9}$$

with

$$\gamma'(f \in A') = \{(b, x) \mid x \in f(b)\} \tag{10}$$

$$f_1 \sqcup' f_2 = \lambda b \ . \ f_1(b) \sqcup_{\mathcal{C}} f_2(b) \tag{11}$$

in which a convex polyhedra is associated with each discrete state, which allows us to represent accurately a relation like $(b_0 \Leftrightarrow x_0 \geq 0)$. This solution does not address however the combinatorial explosion problem. More generally, none of these two solutions can represent non-convex invariants for numerical variables.

1) *Solving Fixpoint Equations on an Abstract Domain:* We remind that we want to solve the fixpoint (4) and (5). The transfer fixpoint theorem says that if $F^\alpha : A \to A$ is a *correct approximation* of $F : 2^S \to 2^S$, which means that $\gamma \circ F^\alpha \supseteq F \circ \gamma$, then $\gamma \circ \mathrm{lfp}(F^\alpha) \supseteq \mathrm{lfp}(F)$. In other words, a fixpoint equation $X = F(X), X \in 2^S$ can be abstracted by $Y = F^\alpha(Y), Y \in A$, and the least solution $Y_0 = \mathrm{lfp}(F^\alpha)$ overapproximates the exact set $X_0 = \mathrm{lfp}(F)$.

If F^α is continuous and A is a complete lattice that contains no infinitely increasing sequences, the most classical way to compute $\mathrm{lfp}(F^\alpha)$ is to compute the sequence

$$Y^{(0)} = \bot \ , \ \ Y^{(n+1)} = F^\alpha(Y^{(n)}) \tag{12}$$

which converges in a finite number of steps to $\mathrm{lfp}(F^\alpha)$ according to Kleene's theorem. Otherwise, if A is not complete and/or contains infinitely increasing sequences (as for instance the convex polyhedra lattice), in addition to the *static approximation* induced by the abstract lattice, one introduces a *dynamic approximation* by using an extrapolation operator $\nabla : A \times A \to A$ called *widening* [14]. Equation (12) is replaced by

$$Y^{(0)} = \bot \ , \ \ Y^{(n+1)} = Y^{(n)} \nabla F^\alpha(Y^{(n)}) \tag{13}$$

which converges in a finite number of steps to a *post-fixpoint* $Y \sqsupseteq F^\alpha(Y) \sqsupseteq \mathrm{lfp}(F^\alpha)$ thanks to technical properties of ∇.[2]

The aim of abstract interpretation framework is thus to generalize the intuitive approximation of geometrical shapes (for instance by bounding boxes or convex polyhedra) to the approximation of more complex objects encountered in program analysis. An important point to be noticed is that in numerical analysis, the notion of approximation is based on the notion of topological neighborhoods, whereas in program analysis and abstract interpretation, it is based on set inclusion.

[2]The use of widening solves in particular problems related to Zeno behavior in timed and hybrid systems.

B. Partitioned Abstract Domain

We already mentioned the shortcomings of the two abstract domains A and A' we defined for 2^S [(6) and (9)] w.r.t. precision and efficiency.

A flexible solution to these problems is to partition the efficient but not very precise abstract domain A in order to improve its expressiveness. Intuitively, partitioning allows us to introduce case reasoning by distinguish different situations and by manipulating disjunctions. For instance, if we partition A according to an inequality $x \geq 10$, we can represent exactly an invariant like

$$\overbrace{(b, x \geq 20)}^{\subseteq x \geq 10} \cup \overbrace{(true, -10 \leq x \leq 0)}^{\subseteq x < 10} \tag{14}$$

whereas a single abstract value in A can only approximate the above invariant with $(true, -10 \leq x)$.

More formally, if $\pi = \{A_1, \ldots, A_n\} \subseteq A$ is a finite set such that $\{S_k = \gamma(A_k) \mid 1 \leq k \leq n\}$ defines a finite partition of S, then we can define the abstract domain $A_\pi = \{(a_1, \ldots, a_n) \in A^n \mid a_k \sqsubseteq A_k\}$. Its concretization function and least upper bound are defined as

$$\gamma_\pi(a_1, \ldots, a_n) = \gamma(a_1) \cup \ldots \cup \gamma(a_n)$$
$$(a_1, \ldots, a_n) \sqcup_\pi (a'_1, \ldots, a'_n) = (a_1 \sqcup a'_1, \ldots a_n \sqcup a'_n).$$

This allows us to manipulate bounded and canonical unions of abstract values, and in particular to represent non-convex numerical invariants, as in (14).

To reflect the partition π in (4), we first exploit the isomorphism $2^S \approx 2^{S_1} \times \ldots \times 2^{S_n}$ to rewrite the reachability fixpoint equation $X = I \cup post(X)$ as

$$X_k = I_k \cup \bigcup_{k'} \overbrace{(post(X_{k'}) \cap S_k)}^{post^{k',k}}(X_{k'}) \tag{15}$$

with $X_k, I_k \subseteq S_k$ and $1 \leq k \leq n$. We then abstract it with

$$Y_k = Y_k^{(0)} \sqcup \bigsqcup_{k'} post_\alpha^{k',k}(Y_{k'}) \tag{16}$$

where $\gamma(Y_k^{(0)}) \supseteq I_k$ and $post_\alpha^{k',k} : A \to A$ is a correct approximation of $post^{k',k} : S_{k'} \to S_k$. The abstract interpretation framework ensures that we can compute iteratively the reachability set in the partitioned abstract domain A_π using widening. We obtain an overapproximation $reach^\sharp \supseteq reach$ of the concrete reachability set. Fig. 1 illustrates two possible partitioning of $S = \mathbb{B}^2 \times \mathbb{R}^2$, according either to all the Boolean variables, or to the condition $b_0 = b_1$, and the corresponding reachability analyses.

C. Partition Refinement

The more the partition π is detailed, the more the abstraction A_π is precise, but also costly. The idea of [29], illustrated in Fig. 6, is then to start with a simple partition, to perform reachability and coreachability analysis, and to intersect their results in order to focus on states and transitions that possibly belong to a counter-example [Fig. 6(b)].

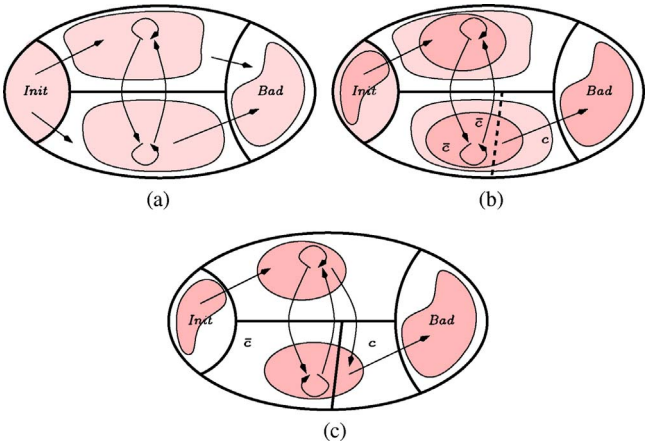

Fig. 6. Analysis on partitioned domain and partition refinement. (a) Reachability analysis. (b) Coreachability analysis. (c) Refinement.

If this set is empty, the property is proved. Otherwise, we refine the partition and start a new analysis cycle. Contrary to most predicate abstraction refinement techniques, that are based on the search of concrete counter-examples [3], our refinement technique can be viewed as based on abstract counter-examples, that is, it tries to remove paths from initial to bad states. In a partition member, a condition [like the condition c in Fig. 6(b)] that separates different behaviors (in terms of abstract transitions) is a good candidate for partition refinement, as it allows us to remove some transition paths in the partition. For instance, the refinement depicted in Fig. 6(c) makes clear that one cannot go in one step from the partition member labeled by $\bar{c}$ to the bad states, something which appeared as possible in Fig. 6(a).

Reference [29] proposes several refinement heuristics on this basis. Necessary conditions to jump from one partition member to another one (like the condition c in Fig. 6) are naturally obtained as a side effect of the technique described in Section V for computing postconditions.

1) *Dynamic Partitioning and Predicate Abstraction: Predicate abstraction* consists of abstracting a (hybrid) system with an abstract finite automaton, by partitioning the state-space of the original system according to a set of formulas, and then abstracting accordingly its transition relation [21]. The abstract system is then checked by classical *finite-state* exploration techniques. As the choice of a suitable partition is of crucial importance (as in dynamic partitioning), refinement techniques have been developed, both for non-hybrid or hybrid cases [3], [12], [27], based on concrete counter-examples or the notion of interpolants [33]. Predicate abstraction can be seen as an instance of dynamic partitioning, where the base abstract domain is the simple lattice $\{\bot, \top\}$ with $\bot \sqsubset \top$; the fixpoint computation on the abstract finite automaton can only show that an abstract state is either non-reachable ($\bot$) or possibly reachable ($\top$). In contrast, dynamic partitioning makes use of more sophisticated abstract domains like convex polyhedra [30], which allows a full range of properties lying between $\bot$ and $\top$ to be discovered by the fixpoint computation.

In some intuitive way, the "cleverness" of predicate abstraction is mainly located in the generation of a detailed partition by the refinement process, whereas dynamic partitioning relies less on a detailed partitioning and more on *propagation* of symbolic properties during fixpoint computations. In particular, in our method we forbid the refinement process to split the state-space according to a condition (Boolean variable or numerical constraint) that does not appear in the program. The main motivation is that we do not want to start enumerating the value of a counter $i = 0, i = 1, \ldots$ in the partition, something which can happen with some predicate abstraction refinement techniques, and which is much less useful in our case, as our fixpoint computations can discover new facts in a non-finite, more expressive abstract domain.

As a side-effect, our refinement process is guaranteed to terminate, leading to a finite partition such that the transition functions partially evaluated on each partition member do not contain conditions any more (according to the syntax of Fig. 4). However, there is no guarantee that the property is provable on the corresponding partitioned abstract domain.

V. Computing Continuous Postconditions

Section III presented our SHA model and Section IV presented our general approach to compute reachable states of such systems. To instantiate it, we need to compute the operator $post_\alpha^{k',k}$ appearing in (16), which is a correct approximation of the operator $\lambda X \subseteq X_{k'}.\big(post(X) \cap X_k\big)$. As the transition relation of a SHA has been defined in Section III-B as the union of a discrete and continuous transition relations, we can decompose the postcondition operator $post$ defined by (2) as the union of a *discrete* and a *continuous* postcondition operators. A method for efficiently computing the discrete postcondition operator is described in [28], so we will focus only on the computation of continuous postconditions.

In classical linear hybrid automata defined in [2], computing a continuous postcondition is quite straightforward; a change of mode requires a discrete transition, and in each mode the constraints on derivatives are constant; only trajectory segments that are line segments need to be considered. Our framework introduces two difficulties.

1) In the SHA model a change of mode can occur during continuous evolutions; the trajectory segments to consider are more complex and the induced postcondition should sometimes be approximated. Moreover, the topological aspects on the frontier separating two partition members have to be taken into account carefully.
2) We must take into account the interactions between the discrete and the continuous part of the state-space, especially in the context of general partitions of the state-space, which are more complex that the partition induced by the control structure of a classical hybrid automaton.

To address these difficulties, we first focus in Section V-A to the case where the state-space is purely continuous, and we then consider the general case in Section V-B. In each case, we first consider the easy subcase for which we can compute the exact postcondition operator, and then for the general case we propose approximation techniques similar to the techniques developed for computing discrete conditions in [28].

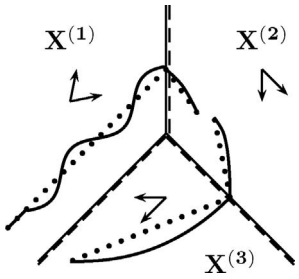

Fig. 7. Decomposing trajectory segments into simple ones and straightening them.

A. Case of Purely Continuous State-Space

We consider a SHA H without discrete state-space: $S = X = \mathbb{R}^m$. We fix a partition $X = \bigcup_{k\in K} X^k$ of the state-space into convex polyhedra, that defines a partitioned abstract domain. We denote by $F^{i,j}$ the frontier $(X^i \cap \bar{X}^j) \cup (\bar{X}^i \cap X^j)$, where $\bar{X}$ denotes the topological closure of a set X.

We first decompose general trajectories into sequences of *simple trajectories*, that are included in pairs of connected regions in the partition, and "crosses" only once the frontier (see Fig. 7).

Definition 1 (Simple Trajectories): We denote by $\mathcal{S}_T^{i,j}$ the subset of T-trajectories $f \in \mathcal{T}_T$ (see Section III-B2), named *simple* (T, i, j)*-trajectories*, such that

$$\exists T_f \ : \ f([0, T_f[) \subseteq X^i \ \wedge \ f(T_f) \in F^{i,j} \ \wedge \ f(]T_f, T]) \subseteq X^j.$$

We define a timed postcondition operator induced by such simple trajectories

$$post^{i,j}(Z) = \left\{ f(T) \mid T \geq 0, f \in \mathcal{S}_T^{i,j}, f(0) \in Z \right\}. \quad (17)$$

Full trajectories will be taken into account by iterating the application of $post^{i,j}$ for $i, j \in K$, during the iterative solving of (16).

1) *Exact Postcondition in a Particular Case:* We give now an exact value of $post^{i,j}$ in a particular case, where $D : X \to 2^X$ is defined as

$$D(x) = \begin{cases} D^i & \text{if } x \in I^i \\ \emptyset & \text{if } x \in (X^i \setminus I^i) \end{cases} \quad (18)$$

with $I^i \subseteq X^i$: D can only have one non-empty value D^i on each partition member. Moreover, we assume that D^i is closed and convex and I^i is convex. These assumptions are satisfied by the hybrid automata model considered in [2].

In order to compute $post^{i,j}$, we show below that only trajectories composed of two line segments need to be considered (or 1 if $i = j$) (see Fig. 7).

Definition 2 (2-Line Trajectories): We denote by $\mathcal{L}_T^{i,j}$ the subset of $f \in \mathcal{S}_T^{i,j}$, named *2-line* (T, i, j)*-trajectories*, such that

$$\exists d_i \in D^i, \exists d_j \in D^j \quad \dot{f}(]0, T_f[) = \{d_i\} \ \wedge \ \dot{f}(]T_f, T[) = \{d_j\}.$$

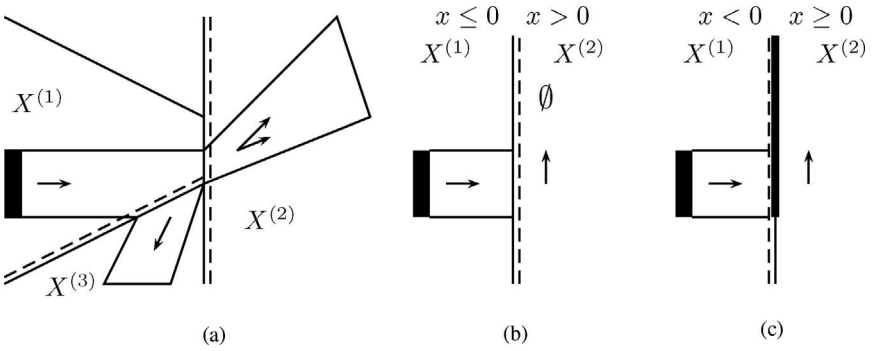

Fig. 8. Three examples of postcondition $post^{i,j}$.

Proposition 1 (Straightening Trajectories): Suppose $f \in \mathcal{S}_T^{i,j}$. Then there exists $g \in \mathcal{L}_T^{i,j}$ such that $T_f = T_g$, $f(0) = g(0)$, $f(T_f) = g(T_f)$ and $f(T) = g(T)$.

Hence, (17) can be rewritten as

$$post^{i,j}(Z) = \left\{ f(T) \mid T \geq 0,\ f \in \mathcal{L}_T^{i,j},\ f(0) \in Z \right\}. \quad (19)$$

Proof: As $f \in \mathcal{S}_T^{i,j}$ is a trajectory, for all $t \in]0, T[$, $\dot{f}(t) \in D(f(t))$ and then for all $t \in]0, T[$, $D(f(t)) \neq \emptyset$. Therefore $f(t) \in I^i \wedge \dot{f}(t) \in D^i$ if $t \in]0, T_f[$, and $f(t) \in I^j \wedge \dot{f}(t) \in D^j$ if $t \in]T_f, T[$. As $\dot{f}$ is piecewise continuous, it is summable and its sum can be approached by a Riemann sum $f(T_f) - f(0) = \int_0^{T_f} \dot{f}(t)dt = T_f \lim_{P\to\infty} \sum_{k=0}^{P-1} \frac{1}{P} \dot{f}\left(\frac{k+1/2}{P} T_f\right)$. As for any $1 \leq k < P$, $\dot{f}(\frac{k+1/2}{P} T_f) \in D_i$, the sum belongs to D_i, because D_i is convex. As it is also closed, the limit is also a vector belonging to D_i. So there exists $d_i \in D^i$ such that $\int_0^{T_f} \dot{f}(t)dt = T_f d_i$. For the same reason, there exists $d_j \in D_j$ such that $\int_{T_f}^{T} \dot{f}(t)dt = (T - T_f)d_j$. It is now easy to build a function g satisfying the proposition. ∎

We can now give a formulation of $post^{i,j}$ with the *time elapse operator* [25] in the case where I^i and I^j are convex polyhedra.

Theorem 1: Let $Z \nearrow^{D^i}$ be the set $\{z + dt \mid z \in Z, d \in D^i, t \geq 0\}$. Then $post^{i,j}(Z) =$

$$\Big[\underbrace{\Big(\overbrace{\big[(Z \cap \bar{I}^i) \nearrow^{D^i} \big]}^{\text{first segment extended to } +\infty} \cap \overbrace{\big[\bar{I}^i \cap F^{i,j} \cap \bar{I}^j\big]}^{\text{intersection with frontier and invariants}} \Big)}_{\text{end of first segment = start of second segment}} \nearrow^{D^j} \Big] \cap \bar{I}^j \cap X^j. \quad (20)$$

If the sets Z, D_i and D_j are convex *polyhedra* (rather than general convex sets), all the operations are implemented without approximation by standard convex polyhedra operations, as described in [25]. Fig. 8 gives examples. Notice the influence of the configuration at the frontier on the result [Fig. 8(b) and (c)].

Proof: Let $G^{i,j} = \bar{I}^i \cap F^{i,j} \cap \bar{I}^j$.

($\subseteq$) Suppose $y \in post^{i,j}(Z)$ as defined by (19). Then there exists $x \in Z$, $T \geq 0$, $f \in \mathcal{L}_T^{i,j}$ with $f(0) = x$, $f([0, T_f[) \in I^i \subseteq X^i$, $\dot{f}([0, T_f[) = d_i \in D^i$, $f(]T_f, T]) \subseteq I^j \subseteq X^j$, $\dot{f}(]T_f, T]) = d_j \in D^j$. We have

$$f(t) = \begin{array}{ll} f(0) + td_i & \text{for } t \in [0, T_f] \\ f(T_f) + (t - T_f)d_j & \text{for } t \in [T_f, T]. \end{array}$$

1) $x = f(0) \in Z \cap I^i$ and $f(T_f) \in (Z \cap I^i) \nearrow^{D^i}$.
2) As $f([0, T_f[) \in I^i$, $f(]T_f, T]) \subseteq I^j$, and f is continuous, $f(T_f) \in \bar{I}^i \cap \bar{I}^j$.
3) By definition 1, $f(T_f) \in X^i \cup X^j$; if $f(T_f) \in X^i$, as $f(]T_f, T]) \subseteq X^j$, by continuity $f(T_f) \in \bar{X}^j$; conversely if $f(T_f) \in X^j$ then $f(T_f) \in \bar{X}^i$; thus $f(T_f) \in F^{i,j}$ and together with 2 $f(T_f) \in G^{i,j}$.
4) By 1 and 3, $f(T_f) \in \Omega = [(Z \cap I^i) \nearrow^{D^i}] \cap G^{i,j}$.
5) $f(T) = f(T_f) + (T - T_f)d_j \in \Omega \nearrow^{D^j}$, and by hypothesis $f(T) \in I^j$. Thus $y \in (\Omega \nearrow^{D^j}) \cap I^j$.

($\supseteq$) Let $y \in post^{i,j}(Z)$ as defined by (20). Then there exists $x \in Z \cap I^i$, $t_i, t_j \geq 0$, $d_i \in D^i$, $d_j \in D^j$ such that

$$\begin{cases} x + t_i d_i \in G^{i,j} \\ y = x + t_i d_i + t_j d_j \in I^j \end{cases}.$$

Let us define the function

$$f(t) = \begin{array}{ll} x + td_i & \text{for } t \in [0, t_i] \\ f(t_i) + (t - t_i)d_j & \text{for } t \in]t_i, t_i + t_j]. \end{array}$$

We have to show that $f \in \mathcal{L}_{t_i+t_j}^{i,j}$ and that $\dot{f}(t) \in D(f)(t)$ for $t \in [0, t_i + t_j]$:

1) $\dot{f}([0, t_i[) = d_i \in D^i$ and $\dot{f}(]t_i, t_i + t_j]) = d_i \in D^i$;
2) $f(0) \in I^i$, $f(t_i) \in G^{i,j} \subseteq \bar{I}^i$; as $\bar{I}^i$ is convex, $f([0, t_i]) \in \bar{I}^i$; similarly, $f([t_i, t_i + t_j]) \in \bar{I}^j$. To conclude we have to check that $f([0, t_i[) \subseteq I^i \subseteq X^i$ and $f(]t_i, t_i + t_j]) \subseteq I^j \subseteq X^j$.

We first show that $f([0, t_i[) \subseteq I^i$. If $f(t_i) \in I^i$, we have by convexity $f([0, t_i[) \subseteq f([0, t_i]) \subseteq I^i$. Otherwise, assume that $f(t_i) \in \bar{I}^i \setminus I^i$. We prove that $\exists t \in]0, t_i[$: $f(t_0) \notin I^i$ leads to a contradiction. Let t_0 be such a t. As I^i is a convex polyhedra and $f(t_0) \in \bar{I}^i \setminus I^i$, this means that there exists a constraint $a \cdot x + b < 0$ satisfied by I^i but not by $\bar{I}^i$, such that $a \cdot f(t_0) + b = 0$. As $f(0) \in I^i$, $a \cdot f(0) + b < 0$. But this implies that for any $t > t_0$, $a \cdot f(t) + b > 0$, so that in particular $f(t_i) \notin \bar{I}^i$, which violates the hypothesis. Thus $f([0, t_i[) \subseteq I^i$.

The proof of $f(]t_i, t_i + t_j]) \subseteq I^j$ is similar. ∎

2) *Approximating the Postcondition in the General Case:* If we remove the assumption that D is constant on each partition member, it is hopeless to give an exact *and* computable formulation of $post^{i,j}$. This indeed reduces to the computation of *post* in a non-partitioned system, which has been shown to be computable only if $X = \mathbb{R}^2$ [7]. Nevertheless, we can

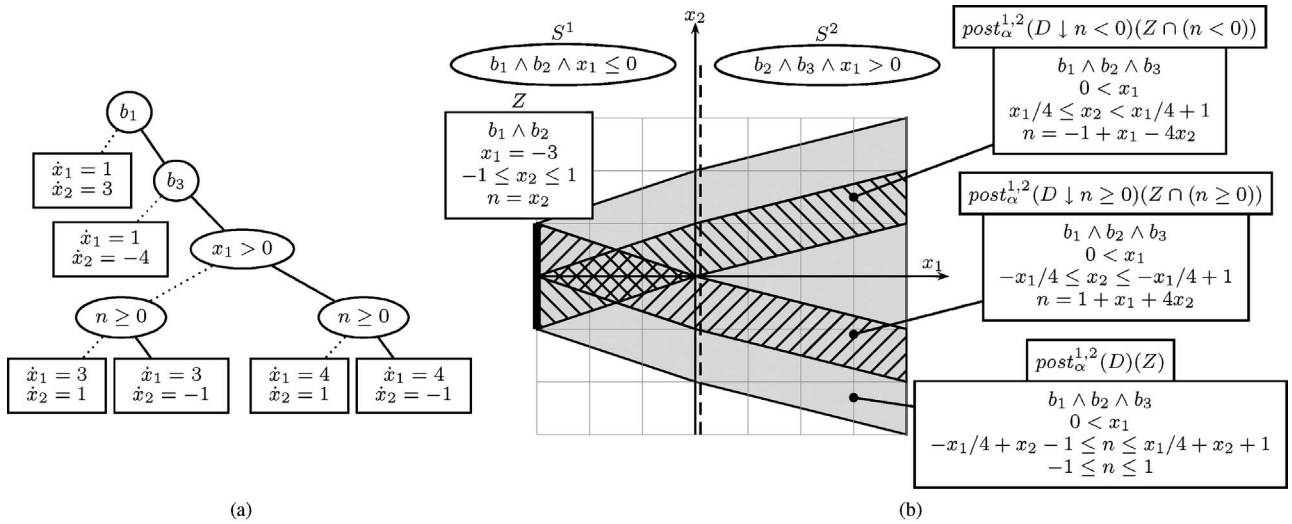

Fig. 9. Different approximations of $post_\alpha^{1,2}(D)(Z)$. $D \downarrow c$ denotes the function D partially evaluated on the condition c. The hatched zones correspond to the result obtained by splitting according to the condition $n \geq 0$, the light grey zone corresponds to the result obtained without any splitting. Both techniques take into account that any trajectory going from S^1 to S^2 satisfies $b_1 \wedge b_2 \wedge b_3$, which allows us to remove the corresponding conditions in the function $D(\vec{b}, n, \vec{x})$. Notice also that the discrete numerical variable n is related in the invariants to the continuous variables $\vec{x}$. (a) $D(\vec{b}, n, \vec{x})$. (b) Splitting or not w.r.t. $n \geq 0$ in postcondition.

consider two solutions.

1) Refine the partition and go back to the situation of the previous section.
2) Replace the function $D : S \rightarrow 2^X$ by a suitable overapproximation.

For the first case, remark that D is expressed by a conditional function (see Fig. 4) and then, such a finite partition exists. Nevertheless, this solution may require a very detailed partition, which induces an expensive analysis. The second solution we will describe is more flexible and more general; if the partition is not detailed enough, it induces approximations, however those can be controlled and improved by refining the partition.

Let $Convex(X)$ denote the smallest convex set containing X, and

$$Supp(D) = \{x \mid D(x) \neq \emptyset\}$$
$$D(X) = \{D(x) \mid x \in X\}.$$

We suggest the following approximation; we define an operator $post_\alpha^{i,j}$ obtained by applying (20) with, for $k \in \{i, j\}$:

$$I^k = Convex(Supp(D) \cap X^k)$$
$$D^k = Convex(D(X^k)).$$

If D is constant and has a convex support on each partition member, we fall back to the previous case $post_\alpha^{i,j} = post^{i,j}$. Thus, the approximations in $post_\alpha^{i,j}$ will be controlled by the fineness of the partition and will be improved during the partition refinement process. Algorithmically, if D is defined by a conditional functions (Fig. 4), we just have to compute the convex hull of a finite set of convex polyhedra.

B. Integrating the Discrete State-Space

Now we consider the general case where $S = Q \times X$. We fix a partition $S = \bigcup_k S_k$ of S into abstract values, i.e., such that for any k, $S_k = \gamma(A_k)$ is the conjunction of a Boolean formula and a convex polyhedra. For the sake of simplicity, we assume that there is no real variable in Q, so that an abstract value Z can be decomposed as $Z = \langle Z_Q, Z_X \rangle \in A$.

1) *Exact Postcondition in a Particular Case:* We assume here a hypothesis similar to (18)

$$D(q, x) = \begin{cases} D^i & \text{if } (q, x) \in I^i \\ \emptyset & \text{if } (q, x) \in (S^i \setminus I^i) \end{cases} \tag{21}$$

with $I^i \subseteq S^i$ an abstract value and D^i a convex polyhedra. We adapt (20) by taking into account the fact that the discrete state-space does not evolve when the time elapses, that is, a trajectory which belong to $S^i \cup S^j$ is necessarily included in $(S_Q^i \cap S_Q^j) \times (S_X^i \cup S_X^j)$. We obtain

$$post^{i,j}(Z) = \left\langle \begin{array}{c} Z_Q \cap I_Q^i \cap I_Q^j \\ \left[\left(\left[\, (Z_X \cap I_X^i) \nearrow^{D^i} \right] \cap \left[\overline{I_X^i} \cap F^{i,j} \cap \overline{I_X^j} \right] \right) \nearrow^{D^j} \right] \cap I_X^j \end{array} \right\rangle. \tag{22}$$

2) *Approximating the Postcondition in the General Case:* If D is not constant on each partition member, we approximate it as for the purely numeric case. We again take into account the fact that the discrete state-space does not evolve when the time elapses, but in a more subtle way. We define an operator $post_\alpha^{i,j}$ obtained by applying (22) with

$$I_Q^i = \left[Supp(D) \cap S^i \right]_Q$$
$$I_X^i = Convex\left(\left[Supp(D) \cap S^i \cap (I_Q^j \times X) \right]_X \right)$$
$$D^i = Convex\left(D\left(S^i \cap (I_Q^j \times X) \right) \right)$$

and conversely for D^j, I_Q^j, and I_X^j. The resulting operator is denoted by $post_\alpha^{i,j}(D)(Z)$ and is illustrated in Fig. 9 by the

TABLE I
DISK CONTROLLER'S ANALYSIS (SEE FIG. 2) WITH $m = M = 2$ AND $\epsilon = 0.8$

Statistics About the Example

State Variables	Input Variables	Num. Constraints	Cond. Derivative Function
9 bool + 3 num	5 bool + 1 num	16	7 diff. inclusions, guarded by 66 minterms

	$[0, IM]$	$[Om, OM]$	Options	Success ?	Partition Size loc/trans	Comments
1	[0, 0]	[0, 0]	No refinement	No	7/13	Ideal instantaneous reaction
2	[0, 0]	[0, 0]	Guided ref. w.r.t. oplus, ominus, mode	No	20/70	
3	[0, 0]	[0, 0]	As 2 + automatic ref.	Yes	57/149	Ref. w.r.t. $s \geq 8$ and $s \leq 12$ (mainly)
4	[0, 0]	[0.6, 0.7]	As 3	Yes	58/233	
5	[0, 0.59]	[0.6, 0.7]	As 3	Yes	86/304	
6	[0, 0]	[0.9, 0.9]	As 3	No	21/42	Partition simplified at the end
7	[0, 0]	$[k, k]$	As 3	(Yes)	24/52	$k < 0.8$ inferred necessary condition
8	[0, 0]	[0.0, 0.0]	Only automatic ref.	Yes	40/119	To be compared to 3
9	[0, 0.59]	[0.6, 0.7]	Only automatic ref.	Yes	235/850	To be compared to 4

light grey region, with $i=1$, $j=2$, and

$$I^1 = \langle b_1 \wedge b_2 \wedge b_3,\ x_1 \leq 0 \rangle \quad D^1 = \{\dot{x_1} = 3 \wedge -1 \leq \dot{x_2} \leq 1\}$$
$$I^2 = \langle b_1 \wedge b_2 \wedge b_3,\ x_1 > 0 \rangle \quad D^2 = \{\dot{x_1} = 4 \wedge -1 \leq \dot{x_2} \leq 1\}.$$

3) *Opening Tests on Discrete Conditions:* On the discrete state-space we can also reason by cases, by splitting the argument Z according to discrete conditions which do not depend on variables involved in continuous evolution. If such a condition c appears in the conditional function D, we decompose the postcondition as follows:

$$post_\alpha^{i,j}\big(ite(c, D^+, D^-)\big)(Z) = post_\alpha^{i,j}(D^+)\left(Z \cap [\![c]\!]\right) \ \sqcup\ post_\alpha^{i,j}(D^-)\left(Z \cap [\![\neg c]\!]\right)$$

where $[\![c]\!]$ is the set of states satisfying c. This operator is more precise, as illustrated in Fig. 9 by the hatched region; here we split the argument Z and the function D according to the constraint $n \geq 0$, which results in

$$(I^1)^+ = (I^1)^- = I^1 \qquad (I^2)^+ = (I^2)^- = I^2$$
$$(D^1)^+ = \{\dot{x_1} = 3 \wedge \dot{x_2} = -1\} \qquad (D^2)^+ = \{\dot{x_1} = 4 \wedge \dot{x_2} = -1\}$$
$$(D^1)^- = \{\dot{x_1} = 3 \wedge \dot{x_2} = 1\} \qquad (D^2)^- = \{\dot{x_1} = 4 \wedge \dot{x_2} = 1\}.$$

We can control the tradeoff between precision and efficiency by the depth of such decomposition. Moreover, if now the partition member S^2 is refined according to the constraint $x_2 \geq 0$ into S^2_+ and S^2_-, splitting the argument Z according to the constraint $n \geq 0$ allows us to discover that $n \geq 0$ is a necessary condition to reach S^2_- from Z, and to use this information for a further partition refinement.

VI. EXPERIMENTS

A. Implementation

The technique presented in Section V for computing continuous postconditions (and preconditions) has been implemented in the NBAC tool [29]. NBAC implements the principles of dynamic partitioning and is connected to an input automaton language (implementing the SHA model) and to the LUSTRE compiler. It exploits the CUDD BDD library [36] and the APRON numerical abstract domain library [31]. NBAC works as follows.

1) It takes a system together with the property to prove, and builds the BDDs and multi-terminal BDDS (MTBDDs) corresponding to the transition functions and the conditional derivative functions.
2) It then perform reachability analysis (from initial states) and coreachability analysis (from final or bad states) on the *Boolean abstraction* of the system. This corresponds to standard model-checking, with the difference that the numerical constraints appearing in Boolean transition functions are treated as additional Boolean inputs, taking into account some (easy) implication between them. This process can typically infer that transition functions of the form b1' = (x + 2y > 1); b2' = (x + 2y > 0) implies that $b'_1 \Rightarrow b'_2$.
   BDDs and MTBDDs are then partially evaluated on the resulting restricted state-space using a generalized cofactor operator [13].
3) For some properties, step 2 is powerful enough to prove the property. Otherwise, the tool starts a full analysis on both Boolean and numerical variables. It begins with a rough initial partition and alternates analysis and automatic partition refinement steps until proving the property, as described in Section IV.

Compared to the purely discrete version of NBAC, we made two adaptations to the refinement process; first, we took into account the results of continuous postcondition operations when gathering the necessary conditions to jump from one partition member to another one (see Section IV). We also prevented the splitting of a partition member satisfying $x \geq 0$ according to the constraint $x \leq 0$. This situation happened very easily with an automaton like the scheduler depicted in Fig. 2(c), and leads to useless refinements; the case $x = 0$ should usually remain merged with either the case $x < 0$ or the case $x > 0$ in the partition (it remains separated when computing postconditions).

TABLE II

ANALYSIS OF THE STEAMBOILER CASE STUDY OF [1]. DEPENDING ON THE ASSUMPTION ON THE ENVIRONMENT, MANY BOOLEAN VARIABLES REMAIN CONSTANT. THE LUSTRE CODE HAS ABOUT 500 LOC, AND MAKES A LARGE USE OF LUSTRE ARRAYS

Statistics About the Example

Assumpt	State Variables	Input Variables	Numerical Constraints	Cond. Derivative Function
1, 2, 3	85 bool + 8 num	10 bool + 2 num	27	9 inclusions guarded by 268 minterms
3	87 bool + 9 num	11 bool + 2 num	30	17 inclusions guarded by 162 minterms

	Assumptions	Success	Max Partition Size	Alternative (Time, Max. Partition Size)
1min	No failure	Yes, 44 s	218/751	33 s + 105 s, 1212/3809
1max	–	Yes, 81 s	338/1188	
2min	Possible failure of pump 0	Yes, 235 s	879/3297	3 m + 14 m, 4964/20 122
2max	–	Yes, 138 s	557/2083	
3min	Possible failure of steam flow device	Yes, 43 s	218/751	33 s + 105 s, 1212/3809
3max	–	Yes, 82 s	338/1188	
4 min	Possible failure of water level device, during at most 20 s separated by at least 40 s	Yes, 24 m	314/1157	25 m + >60 m, >6201/28 209
4max	–	Yes, 91 m	296/1137	

B. Disk Controller Example

We first illustrate the usefulness of partitioning according to numerical constraints (similar observations have been made in the context of predicate abstraction [3]). We analyzed the system described in Section II and Fig. 2.

We verify that the disk motor speed never stays more than 8 consecutive time units outside the desired range. Thus, we already need to partition the state-space according to a numerical constraint ($d \geq 8$) in order to separate "good" and bad sets of states.

We start all analyses with the control structures of the scheduler and the property observer process made explicit in the partitioned abstract domain. Table I gives some statistics about the example, and details our experimental results for various parameters and options. Concerning the statistics about the derivative function, notice that the number of distinct differential inclusions does not correspond to the number of modes in classical model. For instance, if one considers the conditional derivative function of the scheduler of Fig. 2(c), it contains only two inclusions $\emptyset$ and $\dot{t} = 1$, but this corresponds to four modes. A timed automaton [4] generates in our model a derivative function with only two inclusions: $\emptyset$ (when outside the global invariant), and the set $\bigwedge_i \dot{c_i} = 1$ which specifies that all clocks have a rate of 1.

In Table I, lines 1–3 show that we need to refine the partition w.r.t. numerical constraints in order to prove the property. In particular, line 3 shows that we need to partition according to $s \geq 8$ and $s \leq 12$ to make the invariants of Fig. 2(d) convex. Lines 3–5 illustrate that more nondeterminism in the scheduler requires more partition refinement steps. We fail to show the property if $Om = OM = 0.9$ (line 6). We then find a necessary condition, $k < 0.8$, on the parameter $k = Om = OM$ (line 7) for the property to hold. Thus, we are able to analyze the influence of the reaction delay between input and output of the discrete controller. Lines 8–9 show the analysis without initial guided refinement; the results are comparable (line 8) or worse (line 9).

C. Steamboiler Case-Study

We now show that if we treat the discrete part symbolically, we can scale up w.r.t. the complexity of the discrete controller. The case-study is the steam-boiler controller of Abrial [1]. We implemented faithfully the original specification of [1] in LUSTRE (with three instead of four pumps and without initialization phase). Hybrid automata model the behavior of physical quantities and the scheduler depicted in Fig. 2(c) (with $IM = Om = OM = 0$ as in [1]). The controller can enable or disable 0, 1, 2, or 3 pumps at each step, and takes into account detected failures. It needs some kind of anticipation, as there is a delay when switching up a pump. Moreover, the controller has to maintain its own view of the environment, and to exploit this simulated view in order to take adequate decision, for instance in case of water-level device failure. The first versions of the controller were wrong and required some refinements to be correct (limit cases were detected using a LUSTRE simulator and a discrete version of the environment).

The property to be verified is that the water level q stays in $[M1, M2]$. This property is decomposed into two properties (minimal and maximal bound). We check it for various assumptions on the environment, modeling different fault models. Indeed, if all the devices silently fail, the property on the water level cannot be ensured by the controller. We considered thus four fault models on the environment. The top table of Table II gives statistics about the global model obtained, in terms of number of variables, numerical constraints, and derivative functions. It should be noted that depending on the considered fault model, several Boolean variables may be constant, so we give below other means if evaluating the complexity of the global model.

Table II presents our experimental results. "Max partition size" refers to the maximal size of the partition in the course of the refinement process. We used the control structure of the hybrid part of the system as the initial partition, and we then relied on the automatic refinement heuristics described

in [29] [except for assumption 4, where we guided the first refinement step according to the state of the pumps (on or off)]. Assumption 4 corresponds to the most complex fault model: the controller has to use its own simulation of the water level to take decisions, and the property is true only if the failure does not last too much, and if there is enough time between two failures to allow the system to go back to a "normal" state. For this assumption, the number of refinement steps is comparable to the other experiments, but the analysis time is much higher. This is essentially due to more complex BDDs and convex polyhedra computations.

As the other verification tools for hybrid systems accept only lower-level non-symbolic automata, which manipulates only numerical variables, making a direct comparison would have required non-trivial translators and would have been very involving. We make here a comparison with the following alternative analysis method, consisting of:

1) performing a reachability analysis of the Boolean abstraction of the system, and restricting the initial partition to the computed state-space;
2) refining incrementally the partition according to *all* Booleans;
3) and last analyzing and further refining the partition until success.

The column "Alternative" gives the time and max. partition size obtained by this method. The time is decomposed in the time of steps $1 + 2$ and the time of step 3. The max. partition size first gives an idea of the size of the reachable part of a non-symbolic automaton model, and also shows that our automatic refinement is quite good, as our method succeeds to prove the property on a much smaller partition. Regarding time, the alternative technique is always more expensive (by factors of 3–4), or even fails for assumption 4.

Regarding the level of automation, the initial partition we provide is reasonable; it is the explicit control structure of the hybrid environment. The very first refinement steps are guided and further partition according to the state of the three pumps. Then we rely only on automatic refinement heuristics described in [29]. For assumption 4, we added a few more guided refinement steps, by considering some state variables of the LUSTRE controller that corresponds to state variables of the environment.

More details can be found at http://pop-art.inrialpes.fr/people/bjeannet/nbachybrid/nbachybrid.html.

VII. CONCLUSION

We first proposed a symbolic model for hybrid systems, SHA, that allowed us to embed directly higher-level LUSTRE program in a hybrid automaton and also to implement an analysis technique which can combine symbolically both the discrete and the continuous behavior. We then defined the abstract postcondition induced by this symbolic setting, in the context of a partitioned domain.

We implemented the proposed method and we succeeded to prove the global safety property of a very faithful implementation of the steamboiler case study, for various assumptions on the environment and possible failures (occurrences and/or duration). The alternative technique that enumerates all reachable Boolean valuations before analyzing numerical variables proved to be much slower, or impractical in complex cases.

We insist on the fact that previous attempts to verify hybrid models of this case study focused on the physical model of the devices and considered a very simplified version of the software controller. Besides the limitations of the used verification methods, they could hardly specify a detailed model without relying on a real programming language like LUSTRE.

Our experiments also raised some issues. First, we made the classical observation that guiding the refinement process by providing some (obvious) control structure [like the scheduler of Fig. 2(c)] helps a lot. Related to this is that the refinement process performs only partition refinement, and does not have criteria to group back partition members when it seems that the refinement did not really improve the precision. It also appears that the widening in some cases loses important information, which causes further refinement steps. The guided widening technique of [20] improves the precision, but makes the analysis much more expensive. Another point is that it is difficult to manually inspect on the examples how the refinement proceeds w.r.t. the original LUSTRE program, because our tool exploits a lower-level representation of the LUSTRE program that loses its structure. As a consequence, we plan to connect our tool directly to the LUSTRE language in order to implement more sophisticated refinement techniques, to identify more easily performance bottlenecks, and to experiment a larger set of examples.

It would be also very interesting to adapt and experiment the k-induction approach (see [22]) in the case of hybrid systems, and to compare it to this approach. The two approaches are indeed quite complementary; the k-induction approach ignores reachability information and performs only bounded propagation of properties, but in a *exact* way; our abstract interpretation based approach performs unbounded propagation at the cost of approximations.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their comments and suggestions which were very helpful to improve this paper.

REFERENCES

[1] J.-R. Abrial, E. Börger, and H. Langmaack, Eds., *Formal Methods for Industrial Applications: Specifying and Progr. the Steam Boiler* (Lecture Notes in Computer Sciences, vol. 1165). Berlin, Germany: Springer, 1996.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theor. Comput. Sci. B*, vol. 138, no. 1, pp. 3–34, Feb. 1995.

[3] R. Alur, T. Dang, and F. Ivancic, "Counter-example guided predicate abstraction of hybrid systems," in *Proc. TACAS*, LNCS 2619. 2003, pp. 208–223.

[4] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, Apr. 1994.

[5] E. Asarin, O. Bournez, T. Dang, and O. Maler, "Approximate reachability analysis of piecewise-linear dynamical systems," in *Proc. HSCC*, LNCS 1790. 2000, pp. 20–31.
[6] E. Asarin, T. Dang, and O. Maler, "The d/dt tool for verification of hybrid systems," in *Proc. CAV*, LNCS 2404. 2002, pp. 365–370.
[7] E. Asarin, O. Maler, and A. Pnueli, "Reachability analysis of dynamical systems having piecewise-constant derivatives," *Theor. Comput. Sci.*, vol. 138, no. 1, pp. 35–65, Feb. 1995.
[8] J. C. M. Baeten, D. A. van Beek, P. J. L. Cuijpers, M. A. Reniers, J. E. Rooda, R. R. H. Schiffelers, and R. J. M. Theunissen, "Model-based engineering of embedded systems using the hybrid process algebra Chi," *Electron. Notes Theor. Comput. Sci.*, vol. 209, pp. 21–53, Apr. 2008.
[9] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," *Formal Methods Syst. Des.*, vol. 10, nos. 2–3, pp. 171–206, Apr.–May 1997.
[10] X. Briand and B. Jeannet, "Combining control and data abstraction in the verification of hybrid systems," in *Proc. Int. Conf. MEMOCODE*, Jul. 2009, pp. 141–150.
[11] T. Bultan, R. Gerber, and C. League, "Composite model-checking: verification with type-specific symbolic representations," *ACM Trans Softw. Eng. Methodol.*, vol. 9, no. 1, pp. 3–50, Jan. 2000.
[12] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Proc. CAV*, LNCS 1855. 2000, pp. 154–169.
[13] O. Coudert, C. Berthet, and J. C. Madre, "Verification of sequential machines using Boolean functional vectors," in *Formal VLSI Correctness Verification*. Amsterdam, The Netherlands: North-Holland, 1989.
[14] P. Cousot and R. Cousot, "Abstract interpretation and application to logic programs," *J. Logic Programming*, vol. 13, nos. 2–3, pp. 103–179, 1992.
[15] P. Cousot and R. Cousot, "Comparing the Galois connection and widening/narrowing approaches to abstract interpretation," in *Proc. PLILP*, LNCS 631. 1992, pp. 269–295.
[16] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program," in *Proc. POPL*, 1978, pp. 84–96.
[17] W. Damm, S. Disch, H. Hungar, S. Jacobs, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz, "Exact state set representations in the verification of linear hybrid systems with large discrete state space," in *Proc. ATVA*, LNCS 4762. 2007, pp. 425–440.
[18] M. Fränzle and C. Herde, "Hysat: An efficient proof engine for bounded model checking of hybrid systems," *Formal Methods Syst. Design*, vol. 30, no. 3, pp. 179–198, Jun. 2007.
[19] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," in *Proc. HSCC*, LNCS 3414. 2005, pp. 258–273.
[20] D. Gopan and T. W. Reps, "Guided static analysis," in *Proc. SAS*, LNCS 4634. 2007, pp. 349–365.
[21] S. Graf and H. Saidi, "Construction of abstract state graphs with PVS," in *Proc. CAV*, LNCS 1254. 1997, pp. 72–83.
[22] G. Hagen and C. Tinelli, "Scaling up the formal verification of Lustre programs with SMT-based techniques," in *Proc. FMCAD*, 2008, article 15.
[23] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous dataflow programming language LUSTRE," *Proc. IEEE*, vol. 79, no. 9, pp. 1305–1320, Sep. 1991.
[24] N. Halbwachs, Y.-E. Proy, and P. Raymond, "Verification of linear hybrid systems by means of convex approximations," in *Proc. SAS*, LNCS 864. 1994, pp. 223–237.
[25] N. Halbwachs, Y. E. Proy, and P. Roumanoff, "Verification of real-time systems using linear relation analysis," *Formal Methods Syst. Des.*, vol. 11, no. 2, pp. 157–185, 1997.
[26] T. Henzinger, P.-H. Ho, and H. Wong-Toi, "Algorithmic analysis of nonlinear hybrid systems," *IEEE Trans. Autom. Control*, vol. 43, no. 4, pp. 540–554, Apr. 1998.
[27] T. Henzinger, P. Ho, and H. Wong-To, "HyTech: The next generation," in *Proc. RTSS*, 1995, pp. 56–65.
[28] B. Jeannet, "Representing and approximating transfer functions in abstract interpretation of heterogeneous datatypes," in *Proc. SAS*, LNCS 2477. Sep. 2002, pp. 52–68.
[29] B. Jeannet, "Dynamic partitioning in linear relation analysis: Application to the verification of reactive systems," *Formal Methods Syst. Des.*, vol. 23, no. 1, pp. 5–37, Jul. 2003.
[30] B. Jeannet, N. Halbwachs, and P. Raymond, "Dynamic partitioning in analyses of numerical properties," in *Proc. SAS*, LNCS 1694. Sep. 1999, p. 849.
[31] B. Jeannet and A. Miné, "APRON: A library of numerical abstract domains for static analysis," in *Proc. CAV*, LNCS 5643. 2009, pp. 661–667 [Online]. Available: http://apron.cri.ensmp.fr/library
[32] A. Kurzhanski and P. Varaiya, "Ellipsoidal techniques for reachability analysis," in *Proc. HSCC*, LNCS 1790. 2000, pp. 202–214.
[33] K. L. McMillan and N. Amla, "Automatic abstraction without counterexamples," in *Proc. TACAS*, LNCS 2619. 2003, pp. 2–17.
[34] *Modelica: Modeling of Complex Physical Systems* [Online]. Available: http://www.modelica.org
[35] *Simulink* [Online]. Available: http://www.mathworks.com
[36] F. Somenzi. *CUDD: Colorado University Decision Diagram Package* [Online]. Available: ftp://vlsi.colorado.edu/pub
[37] A. Tiwari, "Abstractions for hybrid systems," *Formal Methods Syst. Des.*, vol. 32, no. 1, pp. 57–83, 2008.
[38] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers, "Syntax and consistent equation semantics of hybrid Chi," *J. Logic Algebraical Prog.*, vol. 68, nos. 1–2, pp. 129–210, 2006.

Xavier Briand received the Ph.D. degree in computer science from the University of Bordeaux I, Laboratory for Research in Computer Science, Bordeaux, France, in 2006.

He was a Post-Doctoral Researcher with INRIA-Grenoble, Rhône-Alpes, France, from 2007 to 2009. His current research interests include program verification, abstract interpretation, and control of distributed system (synthesis, partial observation).

Bertrand Jeannet received the Ph.D. degree in computer science from the Institut National Polytechnique de Grenoble, Grenoble, France, in 2000.

He is currently a Research Scientist with INRIA-Grenoble, Rhône-Alpes, France. He was a Post-Doctoral Researcher with the University of Aalborg, Aalborg, Denmark, before joining INRIA-Rennes, Rennes, France, in 2001 and moving to INRIA-Grenoble in 2007. He is the co-author of the APRON Library for Numerical Abstract Domain (http://apron.cri.ensmp.fr/library/), as well as other tools and libraries dedicated to static analysis. His current research interests include program verification, abstract interpretation, particularly numerical variables, shape analysis, interprocedural and concurrent program analysis, and applications to program testing.