Combining control and data abstraction in the verification of hybrid systems

Xavier Briand
INRIA Grenoble – Rhône-Alpes, France
Email: xavier.briand@inria.fr

Bertrand Jeannet
INRIA Grenoble – Rhône-Alpes, France
Email: bertrand.jeannet@inria.fr

Abstract

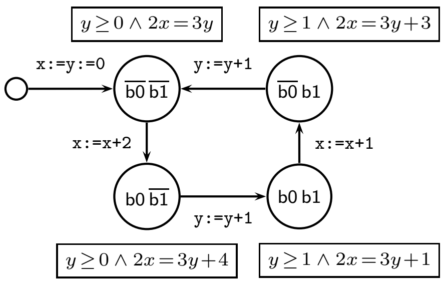
We address the verification of hybrid systems built as the composition of a discrete software controller interacting with a physical environment exhibiting a continuous behavior. Our goal is to attack the problem of the combinatorial explosion of discrete states that may happen if a complex software controller is considered. We propose as a solution to extend an existing abstract interpretation technique, namely dynamic partitioning, to hybrid systems described in a symbolic formalism. Dynamic partitioning allows to finely tune the tradeoff between precision and efficiency in the analysis. We show the effectiveness of the approach by a case study that combines a non trivial controller specified in the synchronous dataflow programming language Lustre *with its physical environment.*

1. Introduction

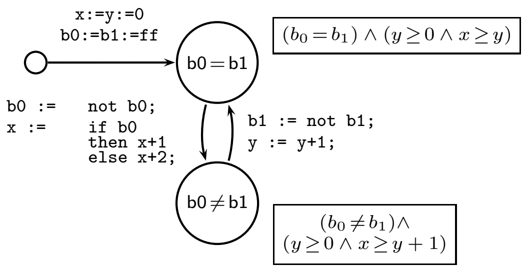
Hybrid systems combine discrete behaviors and continuous behaviors. They are suitable to model physical phenomena interacting with a software controller part with discrete behavior. The primary motivation of this work is the analysis of Lustre synchronous dataflow programs interacting with a physical environment. Lustre [16] is a domain-specific language for programming control-command systems that periodically sample inputs from their environment, compute outputs and move to a new internal state.

Even if the differential equations or inclusions defining the continuous behaviors are relatively simple, the combination with a large discrete space makes the analysis unmanageable in most of the cases.

Verification of hybrid systems focuses 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 (see Fig. 1(a) for a small illustration). 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,


$y \geq 0 \wedge 2x = 3y$
$y \geq 1 \wedge 2x = 3y+3$
x:=y:=0
y:=y+1
$\overline{b0}\,\overline{b1}$
$\overline{b0}\,b1$
x:=x+2
x:=x+1
$b0\,\overline{b1}$
b0 b1
y:=y+1
$y \geq 0 \wedge 2x = 3y+4$
$y \geq 1 \wedge 2x = 3y+1$

(a) fully explicit control structure


x:=y:=0
b0:=b1:=ff
b0 = b1
$(b_0 = b_1) \wedge (y \geq 0 \wedge x \geq y)$
b0 := not b0;
x := if b0 then x+1 else x+2;
b1 := not b1;
y := y+1;
b0 ≠ b1
$(b_0 \neq b_1) \wedge$
$(y \geq 0 \wedge x \geq y + 1)$

(b) partially explicit control structure

Figure 1. Associating invariants to control location. The boxes formula give the invariant computed for each location with a linear relation analysis

hybrid systems makes this combinatorial explosion even more difficult to tackle because invariants are more complex properties (e.g., convex polyhedra) rather than a boolean value (a state is reachable or not).

This paper proposes to extend the principle of *dynamic partitioning* to hybrid systems. The initial motivation for this technique [24], [23], based on abstract interpretation [10], was to apply linear relation analysis [12], [18] 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 on Fig. 1(b). Fig. 1 illustrates the less precise invariants obtained by the analysis if we merge the locations according to the property `b0=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 analysis, so that only states that potentially belong to a counter-example are considered in the refinement process.

Since the first papers on the verification of such systems ([17], [2], [20]), 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 [11]. An obvious limitation of convex polyhedra is that they cannot provide good approximations of non convex invariants. The HyTech tool [19] 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 [18]. As an alternative, [4], [5] suggest to approximate subsets of $\mathbb{R}^n$ with unions of hypercubes, using 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 may arise. Ellipsoid methods have also been proposed [25]. 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 inclusion that can be treated directly with convex polyhedra. Convergence of computations can be achieved by various heuristics.

As mentioned in [13], HyTech and PHAver fail when dealing with large discrete space because only continuous behaviors are treated symbolically. [13] 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, [8] adopts a similar approach by combining BDDs and Presburger formulas in disjunctive normal forms.

This paper. We present here a way to combine 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 2.

From a specification point of view, we present in section 3 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. In particular, our model allows us to embed dataflow Lustre programs.

Section 4 reminds the principles of dynamic partitioning developed in [23]. In this context, section 5 extends the technique of [23], [22] for computing post and preconditions induced by the continuous behavior in a partitioned abstract domain. Integrated in our tool NBac, this extension allows to inherit the features of dynamic partitioning for hybrid systems.

We illustrate in section 6 with experiments the potential of our approach. We first performed various analysis on the example presented in section 2, and we then tackled a very detailed model of the famous steam-boiler specification of R. Abrial [1]. Such a system could hardly be handled without treating symbolically the discrete state-space, due to the complexity of its discrete behavior.

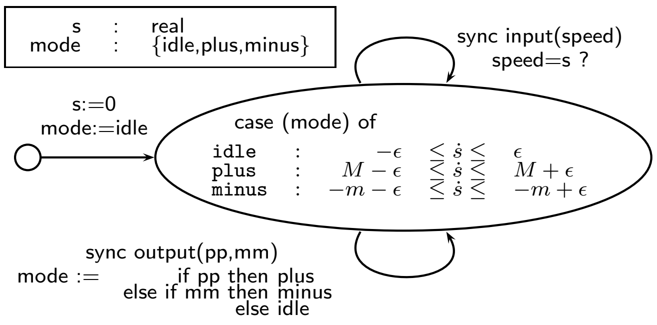
2. Connecting a Lustre program to a hybrid environment

Fig. 2 shows a model of a system composed of a Lustre disk controller interacting with a physical environment, the 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).

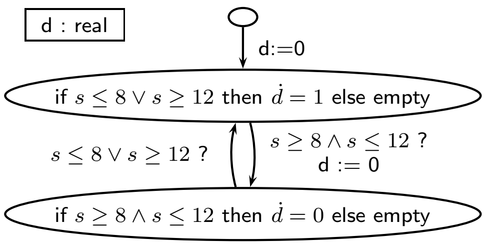
Fig. 2(c) depicts the disk controller. It receives the speed of the disk and emits appropriate signals plus and minus to maintain the motor speed within a specified range (here $[8, 12]$). This program is embedded in a hybrid automaton which receives the speed on the channel input, computes the reaction of the Lustre controller and stores the computed outputs in internal variables, before emitting them on channel output. Fig. 2(b) depicts the property observer. d is the time during which the speed has been outside $[8, 12]$. 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. The process of Fig. 2(d) 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 $\epsilon, m, M, Im, Om, OM$.

3. Symbolic Hybrid Automata (SHA)

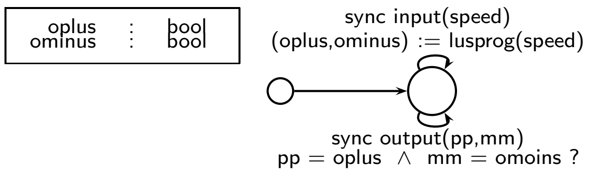
We introduce a symbolic model for hybrid systems, which allows to manipulate symbolically the notions of locations and location invariants. Thus we can avoid the



(a) Environment: disk motor device emits the speed of the disk on channel input and reacting to the controller (channel output)

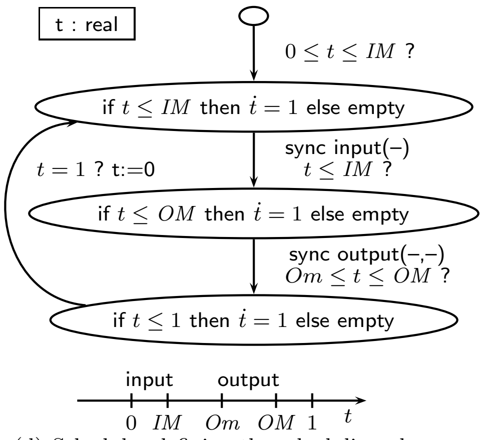

d : real
d:=0
if $s \le 8 \vee s \ge 12$ then $\dot{d} = 1$ else empty
$s \le 8 \vee s \ge 12$?
$s \ge 8 \wedge s \le 12$?
d := 0
if $s \ge 8 \wedge s \le 12$ then $\dot{d} = 0$ else empty

(b) Property observer on the physical environment: d counts the delay for which the speed s has been outside the desired range.



```
node lusprog(speed:real) returns (plus,moins : bool);
let
   plus = speed <= 8.0;
   moins = speed >= 12.0;
tel
```

(c) LUSTRE controller embedded in a process. It emits plus and minus commands to the physical device.


t : real
$0 \le t \le IM$?
if $t \le IM$ then $\dot{t} = 1$ else empty
sync input(–)
$t \le IM$?
$t = 1$? t:=0
if $t \le OM$ then $\dot{t} = 1$ else empty
sync output(–,–)
$Om \le t \le OM$?
if $t \le 1$ then $\dot{t} = 1$ else empty
input
output
0
IM
Om
OM
1
t

(d) Scheduler defining the scheduling above

Figure 2. A disk controller connected to its environment, and a property observer.

state-explosion problem in the specification and we delay this question to the analysis.

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

3.1. The SHA model

A *symbolic hybrid automaton* (SHA) $H = (V, Init, \Delta_H, D_H)$ is defined by a set V of variables of different types that evolves 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. *Conditional differential inclusions* are expressions as [`if b then dx=dy+1 else dx=dy`] (Boolean variable `b` encodes two traditional locations with the universe as invariant), or as [`if x>=2 then dx=dy+1 else dx=dy`], which specifies two traditional locations with both invariants and differential inclusions (we have an implicit transition when $x = 2$: trajectories cross the frontier of the two regions). More precisely:

1) V is a finite set of variables partitioned into variables of any type V_Q (subject to discrete behaviors) and real-value variables V_X (continuous behavior). 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. Notice that we do not restrict *a priori* the types of the variables in V_Q, as the software controller part may manipulate variables of any type.

2) Discrete change of the system are defined by the finite set Δ_H of transitions. In a *discrete transition* $\delta = (\sigma, p, G, A) \in \Delta_H$, σ is an *action* carrying a tuple of *communication parameters* $p = \langle p_1, \ldots, p_k \rangle$. 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). The *assignment* $A : S \times P_\sigma \to S$ defines the evolution of the values of variables during the discrete transition.

3) Continuous evolution is defined by $D_H : Q \times X \to 2^X$. D_H associate to each discrete state $q \in Q$ a *differential inclusion* $\dot{x}(t) \in D(q, x(t))$. The constraints on the derivatives of continuous variables D is a function of any state rather than a function on discrete locations and its guards can specify implicitly traditional location invariants. This symbolic specification of the continuous evolution make easier the abstraction of complex differential equations by simpler piecewise-constant polyhedral inclusions [19] (Fig. 4 illustrates this point).

SHAs can be composed in parallel and communicate by *rendez-vous* on valued channels. The corresponding product is classical and is not detailed here.

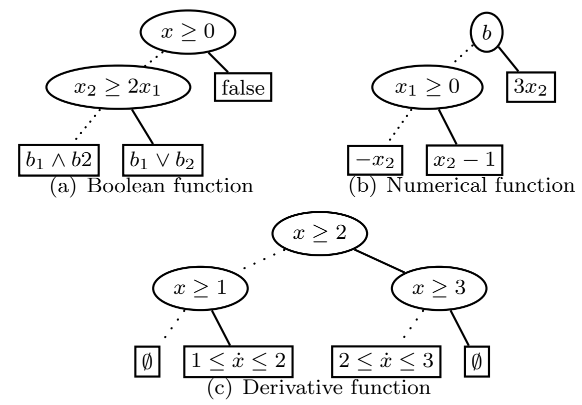


Figure 3. Conditional functions

Syntax of expressions. Functions and formulas involved in the definition of hybrid automata will be *conditional functions*, represented with binary decision diagrams (MTBDDs [7]) built on elementary predicates (Boolean variables or linear constraints) and elementary functions/sets (Boolean formulas, linear expressions, differential inclusions), as in [22], *c.f.* Fig. 3.

For differential inclusions, in this paper we consider only linear hybrid systems, that is, hybrid systems with constant differential inclusion (conjunction of linear constraints on derivatives, like $1 \leq \dot{x} + 2\dot{y} \leq 3$). This excludes affine differential equations of the form $\dot{x} = x$. They can however be approximated by piecewise constant differential inclusions. Notice that tools like PHAVer internally perform such approximations.

3.2. Behavior of SHA

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

Discrete transition relation. The discrete transition relation $\rightarrow_d \subseteq S \times S$ induced by Δ_H is defined by:

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

Continuous transition relation. We note $\mathcal{F}_T$ 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 derivable on $]T_i, T_{i+1}[$ and have 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:

$$\frac{\begin{array}{c} q \in Q, \quad f \in \mathcal{F}_T, T \geq 0, \quad f(0) = x \wedge f(T) = x' \\ \forall t \in [0, T] : \dot{f}(t) \in D_H(q, f(t)) \end{array}}{s = (q, x) \rightarrow_c s' = (q, x')}$$

Remark that if $D_H(s) = \emptyset$, there is no possible continuous transition from s, so the time cannot elapse any more. This allows to implement the traditional notion of invariants. For instance, [`if x>=2 then dx=dy+1 else empty`] specifies that as long as $x \geq 2$, the time can elapse and x and y will evolve according to the differential inclusion, otherwise the time is blocked, because there is no possible valuation of the derivatives, only discrete transitions may be taken. The *linear hybrid automata* defined in [2] are SHA where the valuations of discrete variables Q are the locations. For each location $q \in Q$, $\lambda x.D(q, x) \in \{\emptyset, C_q\}$. The set $\{x \mid D(q, x) \neq \emptyset\}$ corresponds to the usual notion of invariant of the location q, and C_q corresponds to the derivative constraints associated to location q.

We illustrate with the example below the expressiveness and compactness of SHA (although the primary motivation for such a symbolic model is to implement the ideas developed in the introduction regarding symbolic verification techniques). The automaton of Fig. 4(a) can be abstracted by the automaton of Fig. 4(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. 4(c).

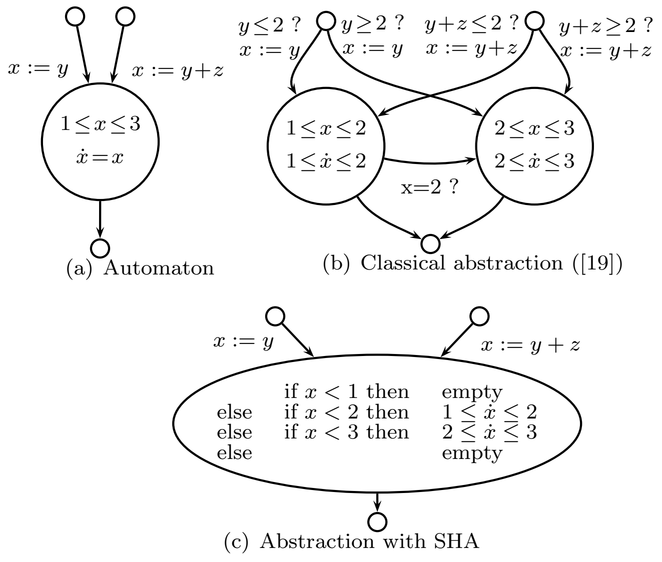


Figure 4. Hybrid automaton and its abstraction in 2 different models

We introduce also few additional notations, used below for reachability analysis. We call T-trajectories the functions $\nu : [0, T] \rightarrow S$ such that $\nu(t) = (q, f(t))$ for some q and we note $\mathcal{T}_T$ the set of T-trajectories. Moreover, the postcondition (resp. precondition) operator and the reachability (resp. coreachability) operators are defined as:

$$\begin{aligned} post(X) &= \{s' \mid s \in X \wedge s \to s'\} && (1)\\ pre(X) &= \{s \mid s' \in X \wedge s \to s'\} && (2)\\ reach(X_0) &= \mathrm{lfp}(\lambda X. X_0 \cup post(X)) && (3)\\ coreach(X_0) &= \mathrm{lfp}(\lambda X. X_0 \cup pre(X)) && (4) \end{aligned}$$

4. Abstract interpretation and dynamic partitioning

We want to check an invariance property on an 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 using the abstract interpretation framework of [23], that we remind in this section.

Base abstract domain. The idea of abstract interpretation is to replace the powerset of states 2^S (on which the fixpoint Eqns. (3) and (4) are defined) by a simpler abstract domain A in order to achieve reasonable performance of the resulting abstract analysis, without being too imprecise either.

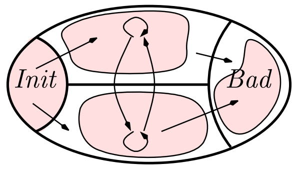
An abstract value $a \in A$ is essentially a specific subset of the state-space $S \simeq \mathbb{B}^n \times \mathbb{R}^m$ (enumerated variables $\times$ numerical variables). Here we make the choice that an abstract value

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

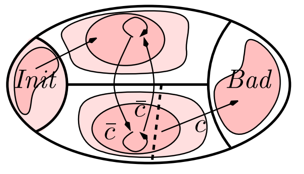
is the conjunction of a Boolean formula B (represented with BDDs) and a m-dimensional convex polyhedron P. The concretization function γ is defined by $\gamma(B, P) = \{(b, r) \mid b \in B \wedge r \in P\}$. Such an abstract value forgets the relations between variables of different types, which is a quite rough approximation.

An alternative could be to consider the much more precise domain $A' = \mathbb{B}^n \to \mathsf{Pol}(\mathbb{R}^m)$, in which a convex polyhedra is associated to each discrete state, but this does not address the combinatorial explosion problem. In addition, none of these two solutions can represent non-convex invariants for numerical variables.

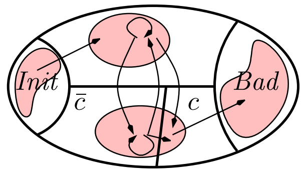
Partitioned abstract domain. The idea of dynamic partitioning is to partition the efficient but not very precise abstract domain A in order to improve its expressiveness [23]. Intuitively, partitioning allows to introduce case reasoning by distinguish different situations. More formally, let $\pi : K \to A$ be a finite partition of S into abstract values, *i.e.*, K is a finite set, $k \neq k' \Rightarrow \pi(k) \sqcap \pi(k') = \bot$, and $S = \bigcup_{k \in K} \gamma(\pi(k))$. A partitioned abstract value is then a function $f : K \to A$ satisfying



(a) Reachability analysis



(b) Coreachability analysis



(c) Refinement

Figure 5. Analysis on partitioned domain and partition refinement

$f \sqsubseteq \pi$. Its meaning is defined by the concretization function

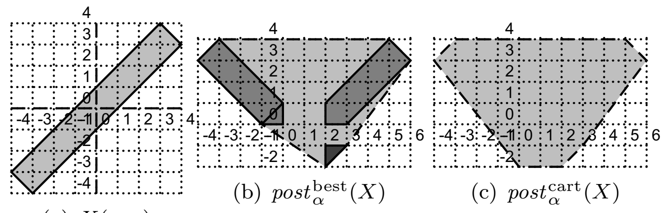
$$\begin{array}{llll} \gamma_\pi : & A_\pi = (K \to A) & \longrightarrow & 2^S \\ & f & \longmapsto & \bigcup_k \gamma(f(k)) \end{array}$$

This allows to manipulate bounded and canonical unions of abstract values, and in particular to represent non-convex numerical invariants. The reachability Eqn. (3) is typically abstracted into an equation system

$$Y^{(k)} = Y_0^{(k)} \sqcup \bigsqcup_{k' \to k} post_\alpha^{k',k}(Y^{(k')}) \quad Y^{(k)} \sqsubseteq \pi(k) \qquad (5)$$

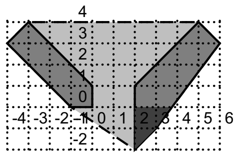
The abstract interpretation framework ensures that we can compute iteratively the reachability set in the (partitioned) abstract domain, using widening for ensuring convergence. We obtain an overapproximation $reach^\sharp \supseteq$ *reach* of the concrete reachability set.

Partition refinement. The more the partition π is detailed, the more the abstraction A_π is precise, but also costly. The idea, illustrated on Fig. 5, is then to start with a simple partition, to perform reachability and coreachability analysis, and to intersect their result in order to focus on states and transitions that possibly belong to a counter-example, Fig. 5.(b). If this set is empty, the property is proved. Otherwise, we refine the partition and we 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. 5.(b)) that separates different behaviors (in terms of abstract transitions) is a good candidate for partition refinement, as it allows to remove some transition paths in the partition. For instance, the refinement depicted on Fig. 5.(c) makes clear that one cannot go in one step from the partition



(a) $X(x,y)$ (b) $post_\alpha^{\text{best}}(X)$ (c) $post_\alpha^{\text{cart}}(X)$

The polygons in medium/dark grey are the images of the polygons obtained by intersecting X with the conjunctions of conditions, and the polygon in light grey is their convex hull (which is the final result). The dark grey part depicts $post(X,Y)$ with $Y=\{(x,y) \mid y<0\}$.



(d) $post_{\alpha,x\geq 0}^{\text{cart}}(X)$

Figure 6. Computation of discrete postcondition

member labelled by $\bar{c}$ to the bad states, something which appeared as possible in Fig. 5.(a). [23] proposes several refinement heuristics on this basis.

Dynamic partitioning and predicate abstraction. *Predicate abstraction* consists in abstracting an (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 [15]. 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 [9], [3], based on concrete counter-examples. 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 [24], 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.

Computing discrete post- and pre- conditions. We very briefly sketch with an example the technique described in [22] for computing discrete postconditions in a partitioned system, which is needed to solve Eqn. (5). Assume we have a transition

$$\begin{aligned} x' = f^x &= \textbf{if } x \geq 0 \textbf{ then } x+2 \textbf{ else } x \\ y' = f^y &= \textbf{if } y \geq 0 \textbf{ then } y \textbf{ else } (\textbf{if } x \geq 0 \textbf{ then } y-1 \\ &\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else } -y) \end{aligned}$$

and the polyhedra X depicted on Fig. 6.(a). The best correct approximation of *post*(X) is obtained by considering all the conjunctions of numerical conditions, in order to reduce the computation to a combination of convex polyhedra operations, Fig. 6.(b):

$$\begin{aligned} post_\alpha^{\text{best}}(X) = \exists x \exists y (\ & X \sqcap \{y<0, x<0\} \sqcap \{x'=x, y'=-y\} & \sqcup \\ & X \sqcap \{y<0, x\geq 0\} \sqcap \{x'=x+2, y'=y-1\} & \sqcup \\ & X \sqcap \{y\geq 0, x<0\} \sqcap \{x'=x, y'=y\} & \sqcup \\ & X \sqcap \{y\geq 0, x\geq 0\} \sqcap \{x'=x+2, y'=y\} &) \end{aligned}$$

However, this solution is exponential in the number of variables v and of conditions in functions f_v. An alternative is to update separately each variable, Fig. 6.(c):

$$\begin{aligned} & post_\alpha^{\text{cart}}(X) = \exists x \exists y \Big(\\ & \quad \big(X \sqcap \{x<0\} \sqcap \{x'=x\} \ \sqcup \ X \sqcap \{x\geq 0\} \sqcap \{x'=x+2\} \big) \sqcap \\ & \quad \big(X \sqcap \{y<0, x<0\} \sqcap \{y'=-y\} \ \sqcup \ X \sqcap \{y<0, x\geq 0\} \sqcap \{y'=y-1\} \ \sqcup \\ & \qquad\qquad\qquad \sqcup \ X \sqcap \{y\geq 0\} \sqcap \{y'=y\} \big) \ \Big) \end{aligned}$$

This solution is linear in the number of variables (but still exponential in the number of conditions). To improve its precision, we can factorize the test on $x \geq 0$, which appears in both functions f_x and f_y, Fig. 6.(d):

$$\begin{aligned} & post_{\alpha,x\geq 0}^{\text{cart}}(X) = \exists x \exists y \Big(\\ & \quad \big((X \sqcap \{x<0\} \sqcap \{x'=x\}) \ \sqcap \ (X \sqcap \{x<0, y<0\} \sqcap \{y'=-y\} \ \sqcup \\ & \qquad\qquad\qquad X \sqcap \{x<0, y\geq 0\} \sqcap \{y'=y\}) \big) \ \sqcup \\ & \quad \big((X \sqcap \{x\geq 0\} \sqcap \{x'=x+2\}) \ \sqcap \ (X \sqcap \{x\geq 0, y<0\} \sqcap \{y'=y-1\} \ \sqcup \\ & \qquad\qquad\qquad X \sqcap \{x\geq 0, y\geq 0\} \sqcap \{y'=y\}) \big) \ \Big) \end{aligned} \tag{6}$$

On a partitioned system, one intersects the results of elementary postconditions with the destination partition member *before* taking their convex hull. For instance, if we take $Y = \{(x,y) \mid y<0\}$, then $post_\alpha^{\text{best}}(X,Y) \sqsubset post_{\alpha,x\geq 0}^{\text{cart}}(X,Y)$ (depicted in dark grey on Figs. 6(b)(d)), although $post_\alpha^{\text{best}}(X) = post_{\alpha,x\geq 0}^{\text{cart}}(X)$.

[22] develops such techniques in the more general case in which Boolean and numerical variables/conditions are freely combined, in order to tune the tradeoff between efficiency and accuracy of computations.

Extension to SHA. The methodology described above can be applied to the verification of hybrid systems: the precondition and postcondition operators should just take into account continuous behaviors in the partitioned domain. In classical (linear) hybrid automata, it is quite straightforward [2]: 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.

In the SHA model, a change of mode can occur during continuous evolutions: the trajectory segments to consider will be slightly more complex. This will be the object of section 5 where we focus on the continuous postcondition. Even if the formulas for this operator are intuitive, the topological aspects on the frontier separating 2 partition members have to be taken into account carefully. We will first compute the exact postcondition for the ideal case and then apply the approximation techniques sketched above for the discrete postcondition.

5. Computing continuous postconditions

5.1. The 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 note $F^{i,j}$ the frontier $(X^i \cap \bar{X^j}) \cup (\bar{X^i} \cap X^j)$ ($\bar{X^i}$ is the adherence of X^i, see [21]). We first decompose general trajectories into sequences of simpler ones, that are included in pairs of connected regions in the partition, and "crosses" only once the frontier.

Definition 1 (Simple trajectories): We note $\mathcal{S}_T^{i,j}$ the subset of T-trajectories $f \in \mathcal{T}_T$ (see section 3.2), named *simple* (T, i, j)*-trajectories*, where there exists a duration, noted T_f, such that $f([0, T_f[) \subseteq X^i$, $f(T_f) \in F^{i,j}$ and $f(]T_f, T]) \subseteq X^j$. □

We then define a timed postcondition operator for these simple trajectories:

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

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 Eqn. (5).

Exact postcondition. We give now an exact value of $post^{i,j}$ in a particular case. We assume here that the evolution function $D : X \to 2^X$ can have only two values $\emptyset$ or D^i on each partition member X^i i.e. there exists $I^i \subseteq X^i$ such that:

$$D(x) = \begin{cases} D^i & \text{if } x \in I^i \\ \emptyset & \text{if } x \in (X^i \setminus I^i) \end{cases} \qquad (7)$$

Moreover, we assume that D^i is closed and convex and I^i is convex. In order to compute $post^{i,j}$, we show below that only trajectories composed of 2 line segments need to be considered (or 1 if $i = j$).

Definition 2 (2-Line trajectories): We note $\mathcal{L}_T^{i,j}$ the subset of $f \in \mathcal{S}_T^{i,j}$, named *2-line* (T, i, j)*-trajectories*, where there exists $d_i \in D^i$ and $d_j \in D^j$ such that $\dot{f}([0, T_f[) = \{d_i\}$ and $\dot{f}(]T_f, T]) = \{d_j\}$. □

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, the timed postcondition operator can be written:

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

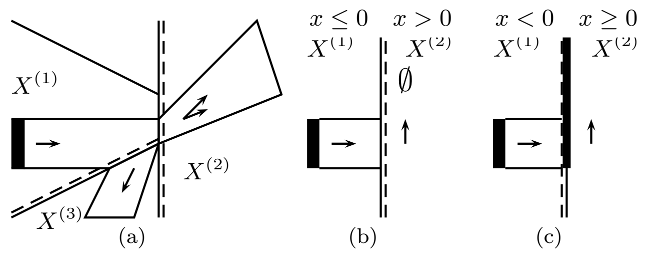


Figure 7. Examples of postcondition $post^{i,j}$

We can now give a formulation of $post^{i,j}$ with the *time elapse operator* [18] 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 $post^{i,j}(Z) =$

$$\left[\left(\left[\, (Z \cap I^i) \nearrow^{D^i} \, \right] \cap \left[\bar{I^i} \cap F^{i,j} \cap \bar{I^j} \right] \right) \nearrow^{D^j} \right] \cap I^j \qquad (8)$$

□

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 [18]. Fig. 7 gives examples. Notice the influence of the configuration at the frontier on the result (Fig. 7(b), 7(c)).

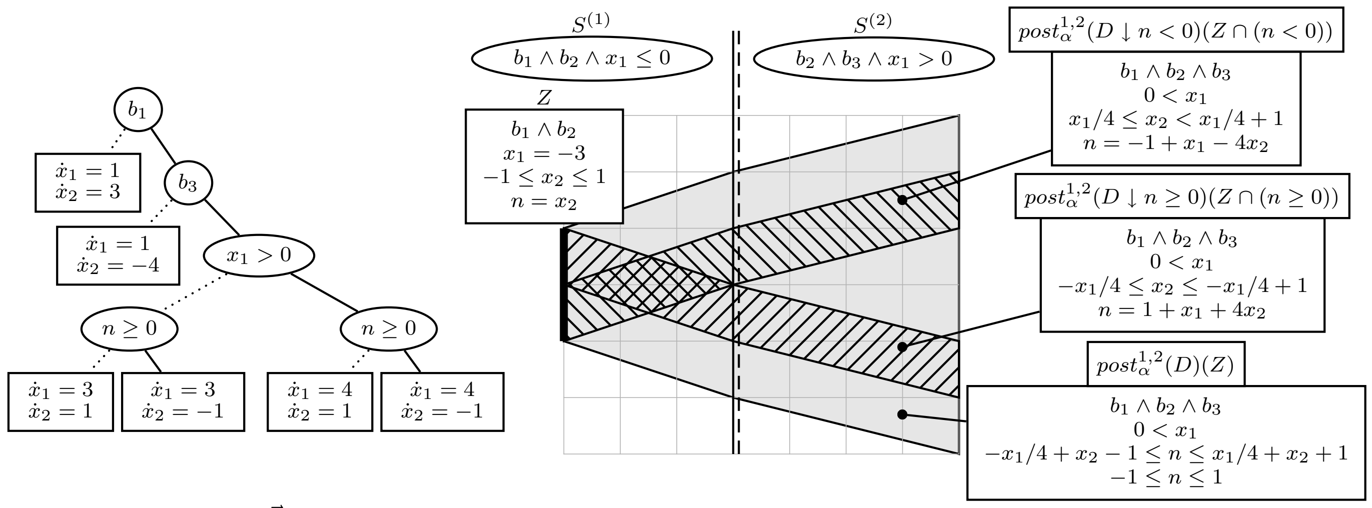
Approximating the postcondition. If we withdraw the assumption that D is constant on each partition member, it is hopeless to give an exact *and* computable formulation of $post^{i,j}$. This indeed reduces to the computation of *post* in a non-partitioned system, which has been shown to be computable only if $X = \mathbb{R}^2$ [6]. Nevertheless, we can consider two solutions:

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

For the first case, remark that D is expressed by a conditional function 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. We suggest the following approximation: we define an operator $post_{\alpha}^{i,j}$ obtained by applying Eqn. (8) with:

$$D^k = \text{Convex}(D(X^k)) \text{ and } I^k = \text{Convex}(\text{Supp}(D) \cap X^k)$$

for $k \in \{i, j\}$, where $D(X) \overset{\text{def}}{=} \{D(x) \mid x \in X\}$, $\text{Supp}(D) \overset{\text{def}}{=} \{x \mid D(x) \neq \emptyset\}$, and $\text{Convex}(X)$ denotes the smallest convex set containing X.


b_1
$\dot{x}_1 = 1$ $\dot{x}_2 = 3$
b_3
$\dot{x}_1 = 1$ $\dot{x}_2 = -4$
$x_1 > 0$
$n \geq 0$
$n \geq 0$
$\dot{x}_1 = 3$ $\dot{x}_2 = 1$
$\dot{x}_1 = 3$ $\dot{x}_2 = -1$
$\dot{x}_1 = 4$ $\dot{x}_2 = 1$
$\dot{x}_1 = 4$ $\dot{x}_2 = -1$
$S^{(1)}$
$b_1 \wedge b_2 \wedge x_1 \leq 0$
$S^{(2)}$
$b_2 \wedge b_3 \wedge x_1 > 0$
Z
$b_1 \wedge b_2$ $x_1 = -3$ $-1 \leq x_2 \leq 1$ $n = x_2$
$post_\alpha^{1,2}(D \downarrow n < 0)(Z \cap (n < 0))$
$b_1 \wedge b_2 \wedge b_3$ $0 < x_1$ $x_1/4 \leq x_2 < x_1/4 + 1$ $n = -1 + x_1 - 4x_2$
$post_\alpha^{1,2}(D \downarrow n \geq 0)(Z \cap (n \geq 0))$
$b_1 \wedge b_2 \wedge b_3$ $0 < x_1$ $-x_1/4 \leq x_2 \leq -x_1/4 + 1$ $n = 1 + x_1 + 4x_2$
$post_\alpha^{1,2}(D)(Z)$
$b_1 \wedge b_2 \wedge b_3$ $0 < x_1$ $-x_1/4 + x_2 - 1 \leq n \leq x_1/4 + x_2 + 1$ $-1 \leq n \leq 1$

(a) $D(\vec{b}, n, \vec{x})$ (b) Splitting or not w.r.t. $n \geq 0$ in postcondition

Figure 8. Different approximations of $post_\alpha^{1,2}(D)(Z)$. $D \downarrow c$ denotes the function D partially evaluated on the condition c.

If D is constant and has a convex support on each partition member, $post_\alpha^{i,j} = post^{i,j}$. 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 function, Fig. 3(c), we just have to compute the convex hull of a finite set of convex polyhedra.

5.2. Integrating the discrete state-space.

Now we consider the general case where $S = Q \times X$. We fix a partition $S = \bigcup_i S^i$ of S into abstract values that are 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 $Z = \langle Z_Q, Z_X \rangle \in A$.

Exact postcondition. We assume first that $D : S \to 2^X$ is constant on each partition member. In Eqn. (7), we now have, for each i, $I^i = I_Q^i \times I_X^i$. Moreover, we take 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 $\left(Q^i \cap Q^j\right) \times X$. Then, similarly to Eqn. (8), we have:

$$post^{i,j}(Z) = \left\langle \begin{array}{c} Z_Q \cap I_Q^i \cap I_Q^j \\ \Big[\Big(\big[\,(Z_X \cap I_X^i) \nearrow^{D^i}\big] \cap \big[\overline{I_X^i} \cap F^{i,j} \cap \overline{I_X^j}\big]\Big) \nearrow^{D^j}\Big] \cap I_X^j \end{array} \right\rangle \tag{9}$$

Approximating postcondition. 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 Eqn. (9) with:

$$D^i = \text{Convex}\big(D\big(S^i \cap (I_Q^j \times X)\big)\big)\ ,\ I_Q^i = \big[\text{Supp}(D) \cap S^i\big]_Q$$
$$\text{and}\quad I_X^i = \text{Convex}\big(\big[\text{Supp}(D) \cap S^i \cap (I_Q^j \times X)\big]_X\big)$$

and conversely for D^j, I_Q^j and I_X^j. The resulting operator is denoted $post_\alpha^{i,j}(D)(Z)$ and is illustrated on Fig. 8.

Now, on the discrete state-space we can also reason by cases, by splitting the argument Z according to discrete conditions (if then else or ite) which do not depend on variables involved in continuous evolution, similarly to Eqn. (6). If such a condition c appears in the conditional function D, we decompose the postcondition as follows:

$$\begin{array}{l} post_\alpha^{i,j}\big(ite(c, D^+, D^-)\big)(Z) = \\ \qquad post_\alpha^{i,j}(D^+)\,(Z \cap [\![c]\!]) \ \sqcup\ post_\alpha^{i,j}(D^-)\,(Z \cap [\![\neg c]\!]) \end{array}$$

where $[\![c]\!]$ is the set of states satisfying c. This operator is more precise, as illustrated on Fig. 8. we control the tradeoff between precision and efficiency by the depth of such decomposition.

6. Experiments

The technique presented in section 5 for computing continuous postconditions (and preconditions) has been implemented in the NBac tool [23]. NBac implements the principles of dynamic partitioning. It begins with a rough initial partition and alternates analysis and automatic partition refinement steps until proving the property, as described in section 4. NBac is connected to an input automaton language (implementing the SHA model) and to the Lustre compiler.

We first illustrate the usefulness of partitioning according to numerical constraints (similar observations

	$[0, IM]$	$[Om, OM]$	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. wrt. oplus, ominus, mode	no	20/70	
3	$[0, 0]$	$[0, 0]$	as 2 + automatic ref.	yes	57/149	ref. wrt. $s \geq 8$ 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

Table 1. The disk controller's analysis (see Fig. 2) with $m = M = 2$ and $\epsilon = 0.8$

	Assumptions	Success	Max partition size	Alternative
1min	no failure	yes, 44s	218/751	33s+105s, 1212/3809
1max	–	yes, 81s	338/1188	
2min	possible failure of pump 0	yes, 235s	879/3297	3m+14m, 4964/20122
2max	–	yes, 138s	557/2083	
3min	possible failure of steam flow device	yes, 43s	218/751	33s+105s, 1212/3809
3max	–	yes, 82s	338/1188	
4min	possible failure of water level device, during at most 20s separated by at least 40s	yes, 24m	314/1157	25m + >60m, >6201/28209
4max	–	yes, 91m	296/1137	

Table 2. Analysis of the steamboiler case study of [1]. The full system has around 90 Boolean state variables, 3 continuous and 2 discrete real state variables. Depending on the assumption on the environment, many Boolean variables remain constant). The Lustre code has about 500 LOC.

have been made in the context of predicate abstraction [3]). We analyzed the system described in Fig. 2. We verify that the disk motor speed is never 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. Tab. 1 gives the experiments' results for various parameters and options. Lines 1–3 show that we need to refine the partition wrt. numerical constraints in order to prove the property. 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).

We now show that if we treat the discrete part symbolically, we can scale up wrt. the complexity of the discrete controller. The case-study is the steam-boiler controller of J.-R. Abrial [1]. We implemented faithfully the original specification of [1] in LUSTRE (with 3 pumps and without initialization phase). Hybrid automata model the behavior of physical quantities and the scheduler depicted on Fig. 2(d) (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. 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).

Tab. 2 presents experimental results when we verify that the water level q stays in $[M1, M2]$. This property is decomposed into two properties (minimal and maximal bound) for each assumptions. "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 then relied on the automatic refinement heuristics described in [23] (except for assumption 4, where we guided the first steps). In the experiment 4, the most complex one, the number of refinement step 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 comparison, we depict in column "Alternative" the time and max. partition size obtained by first performing a Boolean analysis, then refining the partition according to *all* Booleans, and last analyzing (and refining) the resulting partition. This technique is always more expensive (by factor of 3-4), or even fails for the assumption 4. You can find these experiments at http://pop-art.inrialpes.fr/people/bjeannet/nbachybrid/nbachybrid.html. These experiments raise also some issues. It appears first that the widening in some cases looses important information, which causes further refinement steps. The guided widening technique of [14] improves the precision, but makes the analysis much more expensive. It is also difficult to analyze on examples how the refinement proceeds wrt. the original LUSTRE program, because our tool exploits a lower-level representation of the LUSTRE program that looses its structure. As a consequence, we plan to connect our tool directly to the LUSTRE language in order to implement more sophisticated refinement techniques and to identify more easily performance bottlenecks.

7. Conclusion

The symbolic model for hybrid systems (SHA) allowed us to embed directly higher-level LUSTRE program in hybrid automata and also to implement an analysis technique which can combine symbolically both the discrete and the continuous behavior. We define and implement in NBAC the abstract postcondition induced by this symbolic setting. We succeed 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 did consider 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.

References

[1] J.-R. Abrial, E. Brger, and H. Langmaack, editors. *Formal Methods for Industrial Applications: Specifying and Progr. the Steam Boiler*, volume 1165 of *LNCS*, 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. *Theoretical Computer Science B*, 138:3–34, 1995.

[3] R. Alur, T. Dang, and F. Ivancic. Counter-example guided predicate abstraction of hybrid systems. In *TACAS'03*, volume 2619 of *LNCS*, 2003.

[4] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control, HSCC'00*, volume 1790 of *LNCS*, 2000.

[5] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Computer Aided Verification, CAV'02*, volume 2404 of *LNCS*, 2002.

[6] E. Asarin, O. Maler, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138, 1995.

[7] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, 1997.

[8] T. Bultan, R. Gerber, and C. League. Composite model-checking: verification with type-specific symbolic representations. *ACM TOSEM*, 9(1):3–50, 2000.

[9] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV'00*, volume 1855 of *LNCS*, 2000.

[10] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3), 1992.

[11] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In *PLILP'92*, volume 631 of *LNCS*, 1992.

[12] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL'78*, 1978.

[13] 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 *ATVA'07*, volume 4762 of *LNCS*, 2007.

[14] Denis Gopan and Thomas W. Reps. Guided static analysis. In *SAS'07*, volume 4634 of *LNCS*, 2007.

[15] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *CAV'97*, volume 1254 of *LNCS*, 1997.

[16] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[17] N. Halbwachs, Y.-E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *SAS'94*, volume 864 of *LNCS*, 1994.

[18] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2), 1997.

[19] T. Henzinger and P-H. Ho ahd H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43, 1998.

[20] T. Henzinger, P. Ho, and H. Wong-To. HyTech: The next generation. In *RTSS'95*, 1995.

[21] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms.* Springer Verlag, 1996.

[22] B. Jeannet. Representing and approximating transfer functions in abstract interpretation of hetereogeneous datatypes. In *SAS'02*, volume 2477 of *LNCS*, 2002.

[23] B. Jeannet. Dynamic partitioning in linear relation analysis. application to the verification of reactive systems. *Formal Methods in System Design*, 23(1):5–37, 2003.

[24] B. Jeannet, N. Halbwachs, and P. Raymond. Dynamic partitioning in analyses of numerical properties. In *SAS'99*, volume 1694 of *LNCS*, 1999.

[25] A. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *Hybrid Systems: Computation and Control, HSCC'00*, volume 1790 of *LNCS*, 2000.