Automated Verification of Continuous Time Systems by Discrete Temporal Induction

Angelo Gargantini
Università di Bergamo - Italy
angelo.gargantini@unibg.it

Angelo Morzenti
Politecnico di Milano - Italy
morzenti@elet.polimi.it

Abstract

We present a temporal framework suitable for the specification and verification of safety properties of real time hybrid systems. We show that, given suitable assumptions (like non Zenoness and left continuity) continuous time can be discretized by introducing a next *operator that is similar to the one usually found in discrete time temporal logics and can be safely and effectively used in specifications as well as in verification. The proofs of properties can be conducted in a deductive style, and can be easily automated, especially when they are based on induction. We validate this approach by applying it to a simple hybrid system, the well-known thermostat example.*

1. Introduction

In this paper, we propose a temporal framework suitable to describe and prove properties of hybrid systems [2]. Hybrid systems contain variables ranging on continuous and on discrete domains. Moreover in hybrid systems the events (like the acquisition of an input signal in a reactive system or a variable value reaching a given threshold) can occur asynchronously, so that there is no minimum distance between two events or between two occurrences of the same event. On the other hand, we exclude Zeno behaviors: in a given finite interval of time only a finite number of events can occur, and the value of any continuous variable has a finite variability (this notion will be explained in detail in Section 2).

Formal notations dealing with timed systems typically make some assumptions on the underlying time structure. The temporal domain can be *discrete, dense* or *continuous*. For any time instant in a discrete time domain, there is a unique successor (often called *next*) and predecessor. On the other hand, in a dense or continuous time domain there always exists a third time instant between any two given ones. Clearly, selecting the appropriate temporal domain is a crucial part of defining a temporal model also for hybrid systems. Discrete time has a fixed minimal distance between any two events or values of variables, hence it is not suitable for modeling values that change in a continuous fashion nor asynchronous events.

In the present work we are not interested in distinguishing the features of dense and continuous time (for an interesting survey on this issue see [18]): for the sake of simplicity and generality we adopt as a time model simply $\Re$: time is linear and for each instant the system is exactly in one state: we avoid interleaving (where a system can traverse several states at the same time) and adopt a natural formalization for time dependent variables, as functions from $\Re$ to their domain. The variable values can be visualized using customary timing diagrams. We do not introduce a new state variable *time* nor a new action that increases time.

The adoption of $\Re$ as a time model allows for a higher generality, but inevitably poses some problems. First, in a continuous temporal domain Zeno behaviors are possible; we will therefore adopt some explicit assumptions on time-changing entities, taken from [16], to ensure that all entities of our models have the non-Zeno property. Furthermore, the use of continuous time makes specifying and verifying hybrid systems more complex and difficult than in the discrete case.

In $\Re$ one cannot use a next operator, and using temporal induction can be cumbersome as we have to deal with properties of real numbers. In fact, the use of discrete time is often intuitively appealing: for instance, in [11] the authors title a Section “The Joy of Discrete Time”. On the contrary, in our experience [16] reasoning with real numbers can be time consuming and require skill and ingenuity, besides a very good tool support.

In the present paper we try to combine the generality of a continuous time domain with the simplicity and intuitive appeal of a discrete one. Thanks to the hypothesis of non-Zeno behavior, we show that a *next* operator can be introduced, that refers to the state of the system "immediately after the present time" and makes it possible to adopt a simple axiom of induction to derive properties in a manner that

Proceedings of the Thirteenth International Symposium on Temporal Representation and Reasoning (TIME'06)
1530-1311/06 $20.00 © 2006 IEEE

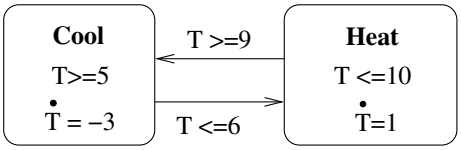

Figure 1. Hybrid automata for a thermostat

can be automated by means of a general-purpose theorem prover like SAL [13].

In Section 2 we briefly recall basic definitions of the TRIO language [21] and state our assumptions about temporal entities in terms of TRIO formulas. In Section 3 we introduce a *next* operator for continuous time and in Section 4 we present a framework for verification based on induction, which is embedded in an automatic theorem prover as explained in Section 5. Section 6 discusses some related literature and Section 7 concludes and presents some future work. Due to space limitations, we were forced to omit proofs of theorems and a few extensions of the presented results; the full version of the paper can be found in [17].

Throughout the paper, we use a simple specification of a thermostat (Fig. 1) which is a linear hybrid automaton and constitutes a simplified version of a case study presented in [1]. The thermostat continuously senses the temperature of the room and turns the heater on (*state* = *Heat*) and off (*state* = *Cool*). It monitors a continuous variable T *temperature* which linearly increases when in state *Heat* by one temperature degree every time unit (for example every one minute) ($\dot{T} = 1$) and linearly decreases when in state *Cool* ($\dot{T} = -3$). The system can switch from *Heat* to *Cool* only when *T* is greater or equal to 9. It can turn on the heat again when *T* is less or equal to 6. Moreover both states have invariants: in *Heat* the temperature must not exceed 10 and in *Cool* the temperature must be greater or equal to 5. The invariants are assumptions about the states: they ensure that "some discrete transition must be taken before the invariant becomes false"[1].

2. TRIO

TRIO, originally introduced in [21] as a formal method for the specification and verification of real time systems, is a first order logic augmented with temporal operators to express properties whose truth value may change over time. The meaning of a TRIO formula is given with respect to a current time instant which is left implicit and in the following it will often be denoted symbolically as *now*. The basic temporal operator is called *Dist*: for a given formula *W*, *Dist*(*W*, *t*) means that *W* is true at a time instant whose distance is *t* time units from the current instant *now*, i.e., from the instant when the sentence is claimed. Many other temporal operators can be derived from *Dist*, as shown in Table 1.

Futr(F, d)	$d \geq 0 \wedge \mathit{Dist}(F, d)$
Past(F, d)	$d \geq 0 \wedge \mathit{Dist}(F, -d)$
Alw(F)	$\forall d \mathit{Dist}(F, d)$
AlwF_i(F)	$\forall d(d \geq 0 \rightarrow Futr(F, d))$
Lasts(F, d)	$\forall d'(0 < d' < d \rightarrow Futr(F, d'))$
NowOn(F)	$\exists d(d > 0 \wedge Lasts(F, d))$
Lasted(F, d)	$\forall d'(0 < d' < d \rightarrow Past(F, d'))$
UpToNow(F)	$\exists d(d > 0 \wedge Lasted(F, d))$
Until(F,G)	$\exists d(d > 0 \wedge Lasts(F, d) \wedge \mathit{Dist}(G, d))$

Table 1. Trio Operators

Besides time dependent (TD) predicates, TRIO introduces *TD variables* with domain X, as variables whose value changes in *X* over time. TD variables are suitable to model physical quantities. To refer to values of a variable or term in the past or in the future, the operator *dist* (as a generalization of *Dist)* is introduced: for a given term *x*, *dist*(*x*,*t*) has the value that *x* had or will have at a time instant whose distance is *t* from now.

In the following, unless otherwise specified, any TRIO formula π representing a system property is intended to be enclosed in an outermost Alw operator and hence to stand for the formula $Alw(\pi)$ asserting that property π holds at every time instant. Next, to make the paper self contained, we report some definitions taken from [16].

Non-Zenoness. The classical informal definition of the non-Zeno requirement for a predicate *A* is that A changes a finite number of times in any bounded temporal interval. This is equivalent to requiring that there exists a time interval, before and after every time instant, where *A* is constantly true or it is constantly false.

Non-Zeno predicate: *a TRIO predicate A is non-Zeno iff* $(\mathit{UpToNow}(\neg A) \vee \mathit{UpToNow}(A)) \wedge (\mathit{NowOn}(\neg A) \vee \mathit{NowOn}(A))$

Recall that the property stated in this definition is intended as enclosed in an outermost Alw operator, hence it implies the classical definition of non Zenoness: no time instant can be an accumulation point of instants where property A changes, neither from the left (because of the operator *UpToNow*) nor from the right (because of the operator *NowOn*). This definition can be generalized for variables in a countable domain as follows.

Non-Zeno variable *x* in a countable domain: *x is non-Zeno iff* $\exists a\ \mathit{UpToNow}(x = a) \wedge \exists b\ \mathit{NowOn}(x = b)$

For such entities it is therefore meaningful to use locution such as "The value of A (or x) immediately before (or after) the current time". Variables on uncountable domains, like for instance the reals or any interval of reals, are required to be piecewise analytic when considered as a function of time.

Non-Zeno variable: *a variable x in an uncountable domain D is non-Zeno iff*

$$\exists f \exists g \left(\begin{array}{c} \{f, g\} \subset AF_o \wedge \exists d \forall t (0 < t < d \rightarrow \\ dist(x, t) = f(t) \wedge dist(x, -t) = g(t)) \end{array} \right)$$

where we denote as AF_o the set of functions that are analytic at 0. Indeed, if f and g are analytic (hence they have derivatives of all orders at the origin), then *x,* required to be equal to them, has a very regular behavior too. It could have only jump discontinuities in isolated points.

As proved in [16], non Zeno predicates, formulas, and variables combined with the usual operators ($<, >, \ldots$) give only non Zeno formulas. Furthermore, we assume that the inputs to the system under specification are non Zeno and that every formula of the TRIO system specification is written in such a way that it does not introduce any Zeno behavior (this is an easy condition to check, as there are very simple and general sufficient conditions ensuring it). In summary, we can therefore assume that, in our specifications, *each predicate, formula, time dependent variable x is non Zeno*, so that, there always exists a time interval preceding and one following the current time, where the modeled system is in a "stable state".

Interval based Formulas and Variables. In our experience, we verified that continuous time systems can be best modeled by partitioning the non-Zeno entities ranging over a discrete domain into two broad categories: those that hold for intervals, and therefore correspond to the intuitive notion of a state, and those that hold in isolated points, hence corresponding to the intuitive notion of an event.

For the sake of brevity we now present the formalization of interval-based entities in terms of TRIO axioms that consider only the case of predicates and formulas: it is a relatively easy exercise to extend the definitions to any variable ranging over a discrete domain.

Interval formula: *a predicate or formula I is interval-based* iff $(I \rightarrow \mathit{UpToNow}(I) \vee \mathit{NowOn}(I)) \wedge (\neg I \rightarrow \mathit{UpToNow}(\neg I) \vee \mathit{NowOn}(\neg I))$

Notice that this definition does not tell anything about the values of I at the precise instant when it changes its value, i.e., whether the intervals where I holds or does not hold are closed at their left or right hand. Various choices are possible, as discussed at depth in [16], but to make the formal modeling of systems and the derivation of their properties more systematic, uniform, and amenable to automation, we chose to adopt the convention that interval-based entities are left-continuous, as illustrated in Fig. 2 and formalized by the following definition.

Left-continuous interval formula: *a predicate or formula I is left-continuous interval-based* iff $(\mathit{UpToNow}(I) \rightarrow I) \wedge (\mathit{UpToNow}(\neg I) \rightarrow \neg I)$

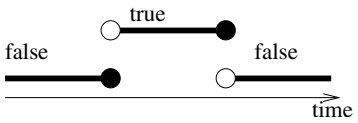

Figure 2. A left continuous interval predicate

In summary, in the rest of the paper we will assume that entities in the TRIO model of a continuous time system are either variables ranging over continuous domains that are piecewise analytic functions or variables ranging over discrete domains or formulas that are left-continuous interval-based.

3. The *next* Operator in the Continuous Time

Besides the usual TRIO operators, we introduce the *next* operator in continuous time, which resembles the usual next operator used in discrete time temporal logics like LTL, but is defined over a dense time. For the sake of clarity we distinguish between time dependent predicates and formulas, and time dependent variables.

Thanks to the non-Zeno property, for every TD predicate or formula *A* and for every time instant, there exists a positive ε such that *A* is steady in the future for at least ε time units. We introduce an operator *next* to represent the value of *A* in the future.

Next for predicates: $\mathit{next}(A) \equiv \exists \varepsilon (\varepsilon > 0 \wedge Lasts(A, \varepsilon))$

The designer may use the *next* operator to model something that becomes true and remains true for a while. For example, the immediate cause-effect relationship between an event *A* and a state *B*, assuming that there is a delay between *A* and *B* but such delay is negligible, can be modeled by $A \rightarrow \mathit{next}(B)$.

For TD variables, there still exists a *next* value, that may be not a constant value, but a function over time. The formal definition follows.

Next for variables: given a time dependent variable x and an analytic function $f \in AF_0$, $\mathit{next}(x) = f \equiv \exists \varepsilon \forall \alpha \, (0 < \alpha < \varepsilon \rightarrow \mathit{dist}(x, \alpha) = f(\alpha))$

Example 1 *If the time dependent variable x is now equal to zero and linearly increases with derivative equal to 1, then we write that $\mathit{next}(x) = ramp_0$ where $ramp_0$ denotes the analytic function that has derivative equal to 1 and starts from the origin. Since $ramp_0(y) = y$, the exact meaning of $\mathit{next}(x) = ramp_0$ is $\exists \varepsilon \forall \alpha (0 < \alpha < \varepsilon \rightarrow \mathit{dist}(x, \alpha) = \alpha)$.* □

The choice of the function f may (analytically) depend on the current value of x or more in general on the state of the system. For example if x linearly increases from its current value, then we can write $\mathit{next}(x) = x + ramp_0$,

whose exact meaning is $\exists\varepsilon\forall\alpha(0 < \alpha < \varepsilon \rightarrow \mathit{dist}(x,\alpha) = x + \alpha)$.

The behavior of any hybrid system is specified by a formula, called *Spec*, consisting of the conjunction of several sub-formulas $A_1 ... A_n$, which describe, by means of the *next* operator, how the state variables evolve.

Example 2 ***Thermostat Specification***. *The next value of the* state *depends on the current value of the* state *and the temperature* T*:*

$A_{state1} : state = \mathit{Heat} \wedge T \geq 9 \rightarrow$
$\mathit{next}(state) = \mathit{Heat} \vee \mathit{next}(state) = \mathit{Cool}$
$A_{state2} : state = \mathit{Heat} \wedge T < 9 \rightarrow \mathit{next}(state) = \mathit{Heat}$
$A_{state3} : state = \mathit{Cool} \wedge T \leq 6 \rightarrow$
$\mathit{next}(state) = \mathit{Heat} \vee \mathit{next}(state) = \mathit{Cool}$
$A_{state4} : state = \mathit{Cool} \wedge T > 6 \rightarrow \mathit{next}(state) = \mathit{Cool}$

All the possible behaviors of the thermostat between now and now $+ \varepsilon$ *are depicted in Figure 3, which shows a possible change in the thermostat* state *in the current instant and the continuous variation of the temperature* T*. At every time instant, there exists an interval in the future in which we can exclude a change of* state *and in which the temperature* T *has a constant derivative, either 1 or -3, depending on the value of* state *in the immediate future.*

$A_{T1} : \mathit{next}(state) = \mathit{Heat} \rightarrow \mathit{next}(T) = T + ramp_0$
$A_{T2} : \mathit{next}(state) = \mathit{Cool} \rightarrow \mathit{next}(T) = T - 3 \cdot ramp_0$

For the thermostat it is therefore Spec $= A_{state1} \wedge ... \wedge A_{state4} \wedge A_{T1} \wedge A_{T2}$*. Strictly speaking, the specification should also include the state invariants, which however are not expressed in terms of* next*; therefore they are not mentioned here and will be added to* Spec *in Section 5.* □

Properties of the next operator. The *next* operator is commutative w.r.t. the propositional connectives and arithmetical operators, i.e. $(\mathit{next}(x) \circ \mathit{next}(y)) \leftrightarrow \mathit{next}(x \circ y)$, with $\circ$ equal to $\vee, \wedge, \neg, \rightarrow, +, -, <, =, \ldots$

4. Property Verification

As shown in the previous section, the designers can use the *next* operator in continuous time like they do in discrete time. However analyzing specifications containing *next* in continuous time is difficult and classical algorithms for discrete time do not apply. In general, the deductive proof of a desired property ϕ starting from the specification of the system *Spec*, i.e. the proof of *Spec* $\rightarrow \phi$, may require time, skill and ingenuity, as shown in [16]. The *next* value of a variable may not be a numerical or symbolic constant, but it may be an analytic function of time like $ramp_0$, $cos(t)$ or e^{-kt}. By expanding the definition of *next* one would obtain many formulas containing quantifiers like $\exists\varepsilon\forall\alpha \ldots$ which are difficult to analyze.

The specification *Spec* could be simplified if ε and α could be fixed to a constant numerical value and functions like $ramp_0$ replaced by their values. However this is in general not possible because, at certain times, the length of the interval in which the system is stable could be smaller than the constant we choose. On the other hand, the non Zeno assumption ensures that for every time instant there exists an ε (in general a different one for each instant) such that, in a future interval of length ε, every variable and all its derivatives are continuous and every discrete variable has a constant value.

We take advantage of this fact by transforming the original specification *Spec* as follows. First, we push out every occurrence of the *next* operator. Applying theorems introduced in Section 3, we can transform for example every occurrence of $\mathit{next}(a) \circ \mathit{next}(b)$ into $\mathit{next}(a \circ b)$, where $\circ$ denotes the usual logical and arithmetic operators. Second, we expand the definition of *next*. Note that a generic formula $a \circ \mathit{next}(b)$ can be rewritten as $\exists\varepsilon\forall\alpha(0 < \alpha < \varepsilon \rightarrow (a \circ \mathit{dist}(b,\alpha))$. The same transformation applies if *next* is in the left side of the above expression. This way we can move every occurrence of "$\exists\varepsilon\forall\alpha(0 < \alpha < \varepsilon \rightarrow \ldots$" to the front of the specification formula, thus reducing it to the form: $\exists\varepsilon\forall\alpha(0 < \alpha < \varepsilon \rightarrow A_1(\alpha) \wedge A_2(\alpha)...)$.

Let $\mathit{Spec}(\alpha)$ denote the formula $A_1(\alpha) \wedge A_2(\alpha)....$ $\mathit{Spec}(\alpha)$ specifies the system state after α time instants, assuming that α is in the future interval (between 0 and ε) in which the system has an analytic behavior. $\mathit{Spec}(\alpha)$ represents a snapshot of the system in α time units after now much in the same way as the Maclaurin series expansion of an analytic function f represents the value of f around 0. Note that $\mathit{Spec}(\alpha)$ does not contain any *next* operator and every analytic function is substituted by its value after α time units.

One could derive $\mathit{Spec}(\alpha)$ from *Spec* simply by replacing every occurrence of $\mathit{next}(x)$ by $\mathit{dist}(x,\alpha)$ and every analytic function f by $f(\alpha)$ (assuming that α is a fresh variable never used before); alternatively one could write $\mathit{Spec}(\alpha)$ from scratch by considering the state of the system at the time instant α time units from now assuming that the system is stable until then.

Example 3 *For the thermostat,* $\mathit{Spec}(\alpha)$ *is composed of the following formulas (compare with those of Example 2):*

$A_{state1}(\alpha) : state = \mathit{Heat} \wedge T \geq 9 \rightarrow$
$\mathit{dist}(state,\alpha) = Heat \vee \mathit{dist}(state,\alpha) = Cool$
...
$A_{T1}(\alpha) : \mathit{dist}(state,\alpha) = \mathit{Heat} \rightarrow \mathit{dist}(T,\alpha) = T + \alpha$
$A_{T2}(\alpha) : \mathit{dist}(state,\alpha) = \mathit{Cool} \rightarrow \mathit{dist}(T,\alpha) = T - 3 \cdot \alpha$

Can we use $\mathit{Spec}(\alpha)$, a substantially simpler version of the specification formula *Spec*, to analyze the system and to prove its desired properties? Of course $\mathit{Spec}(\alpha)$ is not a correct system specification, because the assumption that

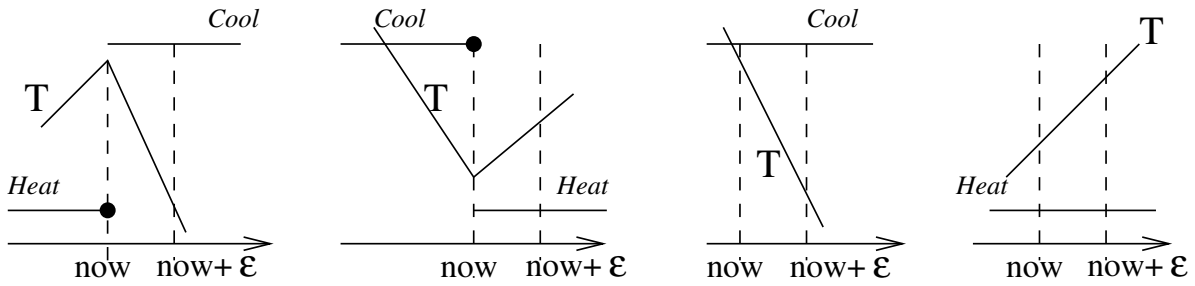

Figure 3. Possible behaviors of the thermostat

the system has an analytic behavior from the current time *now* to $now + \alpha$ may be wrong for some now and some α. However, the formula $\exists\varepsilon\forall\alpha(0 < \alpha < \varepsilon \rightarrow \mathit{Spec}(\alpha))$ holds at every time instant, therefore there always exists an ε' such that $\mathit{Spec}(\alpha)$ is a correct specification of the system for α between 0 and ε'.

Therefore the answer to the above question is positive just like in mathematical analysis one could use the Maclaurin series expansion of a function $f(x)$ to analyze the function f about 0. The following theorem introduces the use of $\mathit{Spec}(\alpha)$ instead of *Spec*.

Theorem 1 ***Using Spec***(α) ***to prove*** *next*(A)*:*
$\exists\varepsilon\forall\alpha\,(0 < \alpha < \varepsilon \rightarrow (\mathit{Spec}(\alpha) \rightarrow \mathit{Dist}(A,\alpha))) \vdash$
$\mathit{Spec} \rightarrow \mathit{next}(A)$

Notice that the schema $\Gamma \vdash \Delta$ of this theorem means that proving Γ is sufficient to prove Δ. Thanks to this theorem, if designers want to prove that $\mathit{next}(A)$ is implied by the system specification *Spec*, then they can instead use $\mathit{Spec}(\alpha)$ (which is significantly simpler than *Spec*) to prove $\mathit{Dist}(A,\alpha)$.

Example 4 *As a simplest example, assume that a system has one state variable x and Spec is $x = 0 \rightarrow \mathit{next}(x) = 3 \cdot ramp_0$. $Spec(\alpha)$ is then $x = 0 \rightarrow dist(x,\alpha) = 3 \cdot \alpha$. Let $x = 0 \rightarrow next(x < 1)$ be the desired property. Instead of proving $Spec \rightarrow x = 0 \rightarrow next(x < 1)$, one can prove that $Spec(\alpha) \rightarrow x = 0 \rightarrow Dist(x < 1, \alpha)$ for an α between 0 and a suitable ε. By substituting $Spec(\alpha)$ by its actual expression, the proof is reduced to $(x = 0 \rightarrow dist(x,\alpha) = 3 \cdot \alpha) \rightarrow x = 0 \rightarrow Dist(x < 1, \alpha)$, which trivially holds for every $\alpha < 1/3$.* □

4.1. Induction

So far we have considered only the system in a generic current time instant and we have specified by *Spec* and $\mathit{Spec}(\alpha)$ the next system state. We have assumed that our specification *Spec* holds forever and we have left an Alw implicit. From now on no outermost Alw operator will be assumed. As explained in Section 2 we assume that the properties of interest are interval-based and left continuous. We introduce now the following form of induction: if A is true now and $A \rightarrow \mathit{next}(A)$ holds in the future (now included), then A holds always in the future.

Theorem 2 $A \wedge AlwF_i(A \rightarrow \mathit{next}(A)) \vdash AlwF_i(A)$

One can use Theorem 2 to prove that a formula A is an invariant of the system by proving that *Spec* implies A and $AlwF_i(A \rightarrow \mathit{next}(A))$. However, as already noted in Section 4, the direct analysis of *Spec* may be very difficult. One could analyze $\mathit{Spec}(\alpha)$ instead and apply the following theorem which combines Theorem 1 and Theorem 2 and introduces the use of induction over $\mathit{Spec}(\alpha)$.

Theorem 3 $A \wedge \exists\varepsilon\forall\alpha(0 < \alpha < \varepsilon \rightarrow (\mathit{Spec}(\alpha) \rightarrow (A \rightarrow \mathit{Dist}(A,\alpha)))) \vdash AlwF_i(\mathit{Spec}) \rightarrow AlwF_i(A)$

This theorem constitutes the theoretical foundation of our deductive approach encoded in SAL as explained in the following section.

5. Encoding Our Approach In SAL

Although the theorems presented in the previous section can be used to prove system properties by hand, the framework was devised to be automated. In this section we show how our approach can be embedded into SAL, a tool for the automatic analysis of state machines [13]. SAL adopts a discrete time model and SAL specifications describe the evolution of state variables as a consequence of the transition from one state to the next one. SAL includes explicit-state model, symbolic and bounded model checkers. In particular the bounded model checker can analyze infinite state space using various ground decision procedures.

We encode the original specification *Spec* in SAL as follows. We add α as a new variable of type `REAL` ranging between 0 and a numeric constant ε. Every monitored or controlled variable of the system is translated into a state variable. The behavior of the system is given in terms of $\mathit{Spec}(\alpha)$, translating every $\mathit{dist}(x,\alpha)$ in x' (which represents

```
thermostat:  CONTEXT = BEGIN
 State:  TYPE = {Heat, Cool};
 alpha:{x :  REAL | x > 0 AND x < 10};
main:  MODULE = BEGIN
 OUTPUT temp :  REAL
 OUTPUT state :  State
 INITIALIZATION state = Heat; temp = 6
 TRANSITION
 state' IN
  IF state = Heat THEN
   IF temp >= 9 THEN {Cool, Heat}
    ELSE {Heat} ENDIF
  ELSE IF temp <= 6 THEN {Cool,Heat}
    ELSE {Cool} ENDIF ENDIF;
 temp' =
  IF state' = Heat THEN temp + alpha
  ELSE temp - 3 * alpha ENDIF    END;
```

Figure 4. SAL specification of the thermostat

in SAL the value of x in the next state) and every function f with its value in α .

The infinite bounded model checker (BMC) of SAL is based on induction and decision procedures over $\Re$. In order to prove $AlwF_i(A)$ we ask BMC to derive $\mathtt{G}(A)$ where G is SAL's operator corresponding to $AlwF_i$ in the discrete time. To prove $\mathtt{G}(A)$, BMC proves both A and $A \rightarrow A'$. Since the SAL specification contains the free variable α declared as a real value ranging between 0 and a numerical constant ε, BMC proves in fact that $\forall\alpha(0 < \alpha < \varepsilon \rightarrow (A \rightarrow A'))$. In our translation in SAL A' is equivalent to $\mathit{Dist}(A, \alpha)$, and ε is a constant, hence BMC proves that $\exists\varepsilon\forall\alpha\,(0 < \alpha < \varepsilon \rightarrow (A \rightarrow \mathit{Dist}(A, \alpha)))$, which constitutes the fundamental part of the premise of Theorem 3. Therefore, thanks to Theorem 3, the proof of $\mathtt{G}(A)$ in SAL by induction on a discrete set of states, ensures that $AlwF_i(\mathit{Spec}) \rightarrow AlwF_i(A)$ holds in continuous time.

Proving Properties. We encoded the linear thermostat in SAL. The first version of the SAL specification is shown in Figure 4.

In the SAL encoding, $\mathit{next}(state)$ becomes `state'` and $\mathit{next}(T)$ becomes `temp'`. The non determinism in the change of state is specified by {`Cool, Heat`}, that denotes any value in {`Cool, Heat`}. We started assuming that `alpha` is a real number greater than 0 and less than 10. Notice that this assumption may not suffice for some properties, as it will be discussed next.

This initial specification does not contain the invariants yet. To formalize the state invariants we introduced the following formula:

```
stateInv:  OBLIGATION main |-
G((state = Heat => temp <=10) AND (state =
Cool => temp >= 5));
```

We had to introduce the invariants in this way, because unfortunately SAL does not allow the introduction of system invariants as axioms in the specification.

The first property we tried to prove states that in `Heat` the temperature is always less then 20.

```
th1:  THEOREM main |-
      G( state = Heat => temp <=20);
```

To prove this property we ran the command: sal-inf-bmc -i –lemma=stateInv thermostat.sal th1, where the option -i forces SAL to use induction, and –lemma=stateInv includes the lemma stateInv as axiom in the proof. SAL is not able to prove this theorem and provides a counterexample: alpha + alpha + alpha + alpha + alpha + alpha + alpha + alpha + alpha + 6 > 20. Analyzing the counterexample we understand that if alpha can be great enough, then the temperature in one or more steps could overcome the value of 20. This counterexample can no longer be found if we introduce an upper-bound for alpha that is small enough. We modified the SAL specification and in particular the definition of alpha, as follows:

```
alpha:  { x:  REAL | x > 0 AND x < 1/10 };
```

With this bound, proving the same property `th1` was immediate. SAL also proved other theorems and the main system invariant stating that the temperature is always between 5 and 10:

```
th5:  THEOREM main |-
      G((temp >=5) and (temp <=10));
```

6. Related work

We can classify our approach as *deductive and symbolic* because we mainly use decision procedures to prove that a desired property holds starting from a set of requirements given as specification. The most used technique for the verification of timed systems is the *algorithmic* analysis of hybrid timed automata [1]. For a survey and an assessment of the state of the art in this field see [22]. An unifying theory of this approach that shows how a hybrid system with infinite state space can be abstracted to a purely discrete system preserving all the properties of interest is presented in [3]. This technique has been implemented in several tools and successfully applied to timed automata [2], i.e. hybrid automata where all continuous variables are clocks that advance with derivative 1 and guards are of type $x \leq c$ where c are numerical constants. For such automata there exist a very compact representation of the state space which allows efficient algorithmic analysis by means of automatic tools. For instance, Uppaal is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc.) [9]. Note that for timed automata the reachability problem is decid-

able, tough it is PSPACE-complete [2, 20]. The algorithmic approach has been extended to linear hybrid automata [1] and to hybrid automata with linear differential equations [4], also called linear hybrid systems. Although for these systems the reachability problem becomes in general undecidable, there exist many classes of automata for which the reachability problem is still decidable. Some strong constraints about the linear equations are necessary to compute exactly and efficiently the state space. For example, for multirate automata and for rectangular automata, where all analog variables follow trajectories within piecewise-linear envelopes and are reinitialized whenever the envelope changes, there exists an algorithm that can solve the reachability problem [20]. Another approach is to approximate the set of reachable states by polyhedrons or ellipsoids. This approach is for instance implemented in the tool **d/dt** [6] or in the tool HyTech [19]. Although approximation algorithms may not terminate or may not solve the verification problem, they have been successfully applied to the verification of real hybrid systems.

The clocked transition systems (CTS) [10] are another interesting extension of timed automata. CTS allow a finite number of discrete transitions which do not modify the values of clocks (i.e. time does not progress). In addition to algorithmic verification of finite-state systems, the CTS model supports also *deductive* verification and it can be extended to hybrid systems. In this case, the verification is mainly done by means of a rule similar to our induction theorem and it is supported by an experimental tool STeP. In CTS, time is replaced by a general (*master*) clock which is incremented by a specific action (*tick*) to be fired when the designer wants to advance the time. Since in general this allows time not to progress, Zeno behaviors are possible; they can however be excluded by proving that eventually *tick* must fire after a finite number of actions. In our approach we assume non-Zeno behavior, Zeno sequences are excluded by construction and there is no need to prove non-Zenoness. Moreover in our approach, there exists an unique system state at every time instant, leading to a more intuitive notion of *current* and *next* state of the system.

On the other hand, discrete time remains the most appealing temporal domain, since it allows enumerative and symbolic techniques (like BDDs), which however are not applicable to a dense time. Many authors have tried to represent continuous change with discrete time and to reduce verification problems for hybrid automata to verification problems for discrete systems which are decidable [7]. In [8] the authors show how the discretization can introduce a small error, which can be taken into account when reasoning and how the time can be discretized not in a fixed way but based on different granularities to avoid errors. In [11] the authors show how the verification of continuous systems can be performed better by using a discrete time and discrete timed automata (whenever possible) than by using the classical algorithmic analysis of hybrid systems in dense time.

There exist several deductive and symbolic approaches for timed system verification, generally based on temporal logics or other temporal formalisms. [16] introduces a framework for the verification of real time systems based on TRIO and the theorem prover PVS. Another temporal logic widely used for specification and verification is the Duration Calculus (DC) [12]. DC is an interval temporal logic based on Moszkowski's (discrete time) interval logic. DC was the first to introduce the concept of an integrating (duration) operator, which is convenient for reasoning about intermittent system behavior. Time is dense and variables keep their value in temporal intervals. The paper [23] presents the DC encoding in the theorem prover PVS, called PC/DC. Particular strategies are implemented to apply in a user friendly way the rules as defined in PVS. Some approaches prefer to embed directly the timed systems in the deductive tool. These approaches require skill and ingenuity in specifying and particularly in proving system correctness. The approach presented in this paper is similar but strives to simplify the framework, and hence the proofs, to favor their automation by means of theorem provers or model checkers.

This paper was partially inspired by the work done in SAL by Dutertre and Sorea [14]. In their approach a timed automaton encoded in SAL can alternatively perform two types of transactions either a discrete transaction (`regular`) which changes the system state without time progress or a time-progress transaction (`elapse`) which increases the time by δ by adding δ to each clock of the system. The value of δ to be added in a elapse transaction is constrained in a finite range to avoid that invariants become false. In our approach we prefer to permit time progress and discrete transaction simultaneously, leading to a more unified treatment of the system transactions and actions. Our approach could be described as "merging" together elapse and regular transactions. The approach of [14] is similar to the one taken by the I/O automata and Lynch-Vaandrager timed automata. For the verification of properties of these types of automata, there exists a powerful deductive proving system called TAME embedded in the theorem prover PVS [5]. The main goal of TAME is not automatic proving but helping the designer to perform proofs by means of intuitive powerful commands (assisted by the tool) similar to the classical steps normally found in proofs done by hand.

7. Conclusions and Future Work

We have presented a framework to demonstrate properties for hybrid systems in a continuous time by using techniques typically used for discrete time systems like the tem-

poral induction. This technique relies on the fact that the system is non-Zeno, that all specification items have a regular behavior, and that they are modeled in terms of a few predefined simple but very general entities (e.g. left continuous interval variables, ...).

We plan to realize a small tool that translates hybrid systems directly to ICS bypassing SAL, and providing a library of facts that ICS is not able to prove and are generally necessary to complete the proofs. Then we intend to apply our approach to further and more complex examples (like those in [1]) and compare our results with the algorithmic approach. We intend to consider alternatives to ICS for decision procedures, especially for non linear equations.

We also plan to select a meaningful set of examples of hybrid systems, for instance the benchmark of [15], and compare the expressive power and the tool support of the approach presented in this paper and the approaches and techniques presented in Section 6.

Our framework could be applied to prove the correctness of a discrete controller, provided that it works at a rate which is greater than $1/\varepsilon$, where ε is the numerical constant for which we have proved the correctness of the continuous system.

We finally point out that, by using decision procedures, we allow parametric specification and verification. Such parametric analysis is very useful (the HyTech feature of parametric verification is quite often used in practice) and our symbolic approach may be better suited to it than the algorithmic approach.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Comp. Science*, 138(1):3–34, feb 1995.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971 – 984, 2000.

[4] R. Alur, T. Henzinger, and H. Wong-Toi. Symbolic analysis of hybrid systems. In *37-th IEEE Conference on Decision and Control*, 1997.

[5] M. Archer, C. Heitmeyer, and E. Riccobene. Proving invariants of I/O automata with TAME. *Automated Software Engineering*, 9(3):201–232, Aug 2002.

[6] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *HSCC*, volume LNCS 1790, pages 20–31. Springer, 2000.

[7] E. Asarin and O. Maler. On discretization of delays in timed automata and digital circuits. In *CONCUR*, volume 1466 of *LNCS*, pages 470–484, 1998.

[8] F. Barber and S. Moreno. Representation of continuous change with discrete time. In *TIME '97*, 1997.

[9] G. Behrmann, A. David, K. G. Larsen, O. Möller, P. Pettersson, and W. Yi. UPPAAL - present and future. In *Proc. of 40*th *IEEE Conference on Decision and Control*. IEEE Computer Society Press, 2001.

[10] N. Bjørner, Z. Manna, H. Sipma, and T. E. Uribe. Deductive verification of real-time systems using STeP. *Theor. Comput. Sci*, 253(1):27–60, 2001.

[11] M. Bozga, O. Maler, and S. Tripakis. Efficient verification of timed automata using dense and discrete time semantics. In L. Pierre and T. Kropf, editors, *Charme*, LNCS 1703, pages 125–141, 1999.

[12] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, Dec. 1991.

[13] L. de Moura, S. Owre, H. Rueß, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari. SAL 2. In *CAV*, LNCS 3114, 2004.

[14] B. Dutertre and M. Sorea. Timed systems in SAL. Technical Report SDL-04-03, SRI Int., 2004.

[15] A. Fehnker and F. Ivancic. Benchmarks for hybrid systems verification. In R. Alur and G. J. Pappas, editors, *HSCC*, volume 2993 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2004.

[16] A. Gargantini and A. Morzenti. Automated deductive requirements analysis of critical systems. *ACM TOSEM*, 10(3):255–307, July 2001.

[17] A. Gargantini and A. Morzenti. Automated verification of continuous time systems by discrete temporal induction. Technical report, Dip. di Ing. gest. e dell'inf. - University of Bergamo, 2006.

[18] I. A. Goralwalla, Y. Leontiev, M. T. Ozsu, and D. Szafron. Modeling temporal primitives: back to basics. In *Proceedings of the sixth international conference on Information and knowledge management*, pages 24–31. ACM Press, 1997.

[19] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A Model Checker for Hybrid Systems. In O. Grumberg, editor, *CAV*, volume 1254 of *LNCS*, pages 460–463. Springer-Verlag, 1997.

[20] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, Aug 1998.

[21] A. Morzenti, D. Mandrioli, and C. Ghezzi. A model parametric real-time logic. *ACM Trans. Program. Lang. Syst.*, 14(4):521–573, 1992.

[22] B. Silva, O. Stursberg, B. Krogh, and S. Engell. An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In *Decision and Control*, 2001.

[23] J. U. Skakkebæk and N. Shankar. Towards a Duration Calculus proof assistant in PVS. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *LNCS*, pages 660–679. Springer, Sept. 1994.