Assertion-based Analysis of Hybrid Systems with PVS[⋆]

Erika Ábrahám-Mumm[1], Ulrich Hannemann[2], and Martin Steffen[1]

[1] Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Preußerstraße 1–9, D-24105 Kiel, Deutschland
`{eab|ms}@informatik.uni-kiel.de`
[2] Computing Science Department, University of Nijmegen
P.O. Box 9010, NL - 6500 GL Nijmegen, The Netherlands
`ulrichh@cs.kun.nl`

Abstract. Hybrid systems are a well-established mathematical model for embedded systems. Such systems, which combine discrete and continuous behavior, are increasingly used in safety-critical applications. To guarantee safe functioning, formal verification techniques are crucial. While research in this area concentrates on model checking, deductive techniques attracted less attention.

In this paper we use the general purpose theorem prover PVS for the rigorous formalization and analysis of hybrid systems. To allow for machine-assisted proofs, we implement a deductive assertional proof method within PVS. The sound and complete proof system allows modular proofs in that it comprises a proof rule for the parallel composition. Besides hybrid systems and the proof system, a number of examples are formalized within PVS.

Keywords: hybrid systems, deductive methods, machine-assisted verification.

1 Introduction

Embedded systems interacting with the physical environment pervade everyday's life and are increasingly used in safety-critical applications, for instance for automotive control, avionics, telematics, chemical process control systems, etc. To guarantee safe functioning, rigorous, i.e., formal arguments are crucial. Their formal analysis is challenging, as well, since such systems are notoriously complex. Capturing the *discrete* finite-state behavior of the digital device as well as the *continuous*, infinite-state behavior of the physical environment, *hybrid systems* [2] provide an appropriate and well-studied formal model. To deal with the complexity of the verification task and to ensure the necessary rigor for fail-safe arguments, computer support is in demand, where two major approaches —enumerative and deductive— of complementary strengths and weaknesses can be distinguished.

Enumerative techniques like model checking promise fully automatic system verification. Based on state-exploration, however, they are limited by the size of the model,

[⋆] The work was supported by the Dutch Technology Foundation STW, project EIF 3959, "Formal Design of Industrial Safety-Critical Systems" and further by the German Research Council (DFG) within the special program KONDISK under grant LA 1012/5-1.

especially when dealing with the parallel composition of systems. This phenomenon is known as the notorious *state-explosion problem.* Furthermore, for hybrid systems as *a priori* infinite-state models, one has to face the fact that they cannot be dealt with automatically in their full generality. Already the computational properties of timed-automata [3], an important subclass of hybrid systems, are undecidable. Therefore, in the field of model checking, research concentrates on identifying tractable subclasses, for instance *linear* hybrid systems [2] as the most prominent subclass (cf. for instance [21, 16, 28]). Instead of restricting the class of models, one can also resort to *approximative* techniques at the expense of information loss (cf. for instance [15, 14]).

In contrast, *deductive* methods do not support fully automatic verification, but are applicable to the full model of hybrid systems. While there are some theoretical investigations on deductive proof methods for hybrid systems (cf. for instance [22] for an overview), work on computer assistance is scarce. See the concluding section for further discussion of related work in this field.

Classical deductive verification techniques use induction over the system's computation steps to prove invariance of properties. First introduced for sequential programs, these *assertional methods* have been extended for more complex models of computation, especially for various forms of parallel and communicating programs (cf. [9] for an extensive monograph on the topic).

In this paper we describe an assertion-based deductive proof method for hybrid systems. To assure rigorous formal reasoning, we employ the interactive theorem prover PVS [25]. PVS is based on higher-order logic, includes extensive libraries of data-structures and theories, offers powerful strategies to assist in routine verification tasks, as well as modularization facilities. We furthermore use PVS to rigorously reason about different examples.

The remainder of the paper is organized as follows. We start in Section 2 briefly surveying the pertinent features of PVS and highlighting the use of the tool for our formalization. In Section 3, we review the definition of hybrid systems, their transition semantics, and their parallel composition. Section 4 describes the proof method for verifying safety properties of hybrid systems, based on assertion networks: After introducing the basic definitions in Section 4.1, we extend them in Section 4.2 to deal with the parallel composition of hybrid systems. After describing in more detail the PVS formalization of hybrid systems including one of the treated examples in Section 5, we conclude in Section 6 with a discussion of related and future work. The library of PVS-theories formalizing the hybrid system model, together with the proof methods and the examples is available via `http://www.informatik.uni-kiel.de/~eab`.

2 The PVS Theorem Prover

Theorem provers offer mechanized support for logical reasoning in general and for program verification in particular. Unlike verification systems for fully automated reasoning such as model checkers [6], theorem provers provide machine-*assistance,* i.e., an interactive proof environment. Interactive means that the user is requested to organize the proof, for instance to come up with an induction hypothesis, to split the proof in appropriate lemmas, etc. While doing so, the verification environment takes care of tedious details like matching and unifying lemmas with the proof goals and assists in the proof organization by keeping track of open proof goals, the collected lemmas and properties. Last but not least it offers a range of automatic decision or semi-decision procedures in special cases. Well-known examples of theorem provers are Isabelle [27], Coq [7], PVS [25, 26] and HOL [11].

To formalize hybrid systems and their theories, we use the theorem prover PVS (Prototype Verification System) developed at SRI International Computer Science Laboratory. PVS is written in Common Lisp and has been used for a wide range of applications; cf. [29] for an extensive bibliography.

PVS's built-in specification language is a typed higher-order logic. Type declarations, their operations and properties are bundled together into so-called *theories* which can be organized hierarchically using the `IMPORTING`-construct. Theories may contain declarations, definitions, axioms, lemmas, and theorems, and can be parameterized with type or value parameters. PVS has a extensive prelude with many predefined types such as lists, sets, natural numbers, integers, reals, relations, functions, etc., and associated lemmas about their properties. Type construction mechanisms are available for building complex types, e.g., lists, function types, records, and recursively defined abstract data types. Being based on a typed logic, PVS automatically performs type-checking to ensure consistency of the specification and the proof-in-progress. Furthermore, the type checking mechanism generates new proof obligations, so-called *Type-Correctness Conditions*, which are often very useful for an early detection of inconsistencies.

Besides the typed internal logic, the PVS-environment supports the interactive verification by predefined and user-definable proof strategies. It offers facilities for proof maintenance, such as editing and rerunning (partial) proofs, easy reuse of already existing proofs, and the like. PVS notation will be introduced when used in the examples; for a complete description of PVS we refer to the PVS manual [26]. In the sequel, the `typewriter`-font indicates formalization in the PVS language.

3 Hybrid Systems

3.1 Basic definitions

Hybrid systems [2] are a well-known formal model for discrete systems acting in a continuous environment. The system's discrete part is represented as a finite set of locations or modes Loc, connected by discrete transitions or edges. The continuous part is given by a finite set $Var \subseteq Var_g$ of variables ranging over the real numbers $\mathbb{R}$, where Var_g is a countably-infinite variable set. A mapping $\nu : Var \to \mathbb{R}$ of variables to real values is called a *valuation*; the set of all valuations is denoted by V. A location-valuation pair $\sigma = (l, \nu) \in Loc \times V$ constitutes a *state* of a hybrid system. Let $\Sigma = Loc \times V$ denote the set of all states. A state set $Ini \subseteq \Sigma$ characterizes the initial states of the system.

As states consist of a discrete and a continuous part, so do the transitions of a hybrid system. A discrete state change is captured by the edges of the graph: leading from one location to another, a transition changes the discrete part of the state; besides that, in going from one location to the next, it may alter non-deterministically the values of the variables. To cater for synchronization between parallel components, the edges come decorated with a synchronization label from a finite label set Lab. The set of labels contains a specific stutter label τ denoting internal moves, not eligible for synchronization. Each location l is assumed to be able to perform a *stutter* transition labeled by τ. Such a transition stands, as usual, for a "do-nothing" step and denotes that other hybrid systems involved in the parallel composition take some discrete transitions. To distinguish between variables the component has under its control in a stutter transition and those it cannot actively influence, the variable set is split into *control* and *non-control* variables. The distinction is drawn per location by a function $Con : Loc \rightarrow 2^{Var}$. Stutter transitions leave the valuations for control variables of the given location unchanged, while putting no restriction on the effect concerning the non-control variables, as they are considered as being influenced solely by the outside.

For the continuous part, the values of the variables may evolve over time, where the corresponding behavior is described, per location, by a set of *activities.* An activity is a continuous function, describing the variables' change starting from the moment the location is entered. Since the specific entrance point in time should not influence the behavior relative to that moment, the set of activities for a location is required to be insensitive against shift in time, or time-invariant. Let $\mathcal{F}$ denote the set of all continuous functions in $\mathbb{R}^{\geq 0} \rightarrow V$. A set $F \subseteq \mathcal{F}$ of activities is called *time-invariant,* if for all $f \in F$ and $t \in \mathbb{R}^{\geq 0}$, also $f + t \in F$, where $f + t$ denotes the function which assigns to each $t' \in \mathbb{R}^{\geq 0}$ the value $f(t + t')$. An *invariant* finally is attributed to each location, i.e., a predicate over the set of valuations V, where the system is allowed to enter or stay in a location only as long as the invariant evaluates to true.

Before giving the formal definition of a hybrid system, let us fix some notations. We write $f|_{A'} : A' \rightarrow B$ for the restriction of a function $f : A \rightarrow B$ to a sub-domain $A' \subseteq A$; the same notation is used for the extension of the restriction operator to sets of functions, as well. For $f \in \mathbb{R}^{\geq 0} \rightarrow V$ and $x \in Var$, we denote by f^x the function in $\mathbb{R}^{\geq 0} \rightarrow \mathbb{R}$ such that $f^x(t) = f(t)(x)$ for all $t \in \mathbb{R}^{\geq 0}$. We call a function $f \in \mathbb{R}^{\geq 0} \rightarrow V$ continuous, if for all $x \in Var$, f^x is continuous. The following definition corresponds to the one encoded in PVS; to avoid overly baroque notation, we allowed ourselves to elide type declarations present in PVS within the definitions in the paper, in case the type can unambiguously be inferred from the context. This convention applies to all the following definitions.

Definition 1 (Hybrid system). *A* hybrid system H *is a tuple* $(Loc, Var, Con, Ini, Lab, Edg, Act, Inv)$*, where Loc is a finite, non-empty set of* locations *and* Var *a finite, non-empty set of* variables. *The function* $Con : Loc \rightarrow 2^{Var}$ *defines the* control variables *in each state, the set* $Ini \subseteq \Sigma = Loc \times V$ *the* initial states. *The* transitions *are given by* $Edg \subseteq Loc \times Lab \times (2^V \times (V \rightarrow 2^V)) \times Loc$*, where* Lab *denotes a finite set of* labels *containing the stutter label* τ*. For all* $l \in Loc$ *there is a stutter transition* $(l, \tau, (\phi, h), l) \in Edg$ *such that* $\phi = V$ *and* $h(\nu) = \{\nu' \mid \nu|_{Con(l)} = \nu'|_{Con(l)}\}$*. The* activities *are given by* $Act : Loc \rightarrow 2^{\mathcal{F}}$ *such that* $Act(l)$ *is time-invariant for each location* $l \in Loc$*. The function* $Inv : Loc \rightarrow 2^V$ *specifies the* invariants.

For a discrete transition $(l_1, \alpha, (\phi, h), l_2) \in Edg$, $\phi \subseteq V$ is called the *guard* and $h : V \rightarrow 2^V$ its *effect*. Depending on various restrictions on the form of the invariants, the guards, the activities etc., a score of variants and simplifications of this model have been investigated, especially to obtain decidable and automatically checkable subclasses of the general definition (cf. for instance [2, 3, 21, 16, 28]). As in this paper we are concerned with formulating a proof method within a deductive framework, we will stick to the general definition.

Representing the above definition in PVS is straightforward. The hybrid system tuple is represented by the type `hys`, a record type, i.e., a product type with named fields written as `[# ... #]` (the record value is denoted by `(# ... #)`). The PVS theory of the same name `hys`, partly shown below, contains the type definition of hybrid systems.

Pvs

```
hys: THEORY
BEGIN
   ...
   IMPORTING invariant_func

   hys: TYPE = [#
      Loc:  location_set,
      Vari: variable_set,
      Lab:  label_set,
      Cont: control_variable_func[Loc,Vari],
      Ini:  state_set[Loc,Vari],
      Edg:  edge_set[Loc,Vari,Lab],
      Act:  activity_func[Loc,Vari],
      Inv:  invariant_func[Loc,Vari]  #]
END hys
```

The component types of the above PVS-definition are implemented and grouped into separate theories and imported into `hys` by the `IMPORTING`-construct. For example, the type of an invariant function, which assigns to each location an invariant (i.e., a valuation set), is implemented as a parameterized theory, since its type depends on the location and the variable sets:

Pvs

```
invariant_func[(IMPORTING location) Loc :location_set,
               (IMPORTING variable) Vari:variable_set]: THEORY
BEGIN
   IMPORTING valuation[Vari]
   invariant_func : TYPE = [(Loc)->valuation_set]
END invariant_func
```

3.2 Semantics

As mentioned before, a system's state can change in two ways: either by discrete transitions or by time delay. Hence there are two kinds of transitions between states: an instantaneous, *discrete* step, written $\rightarrow^{\alpha}$, follows an edge $(l_1, \alpha, (\phi, h), l_2)$ of the system, thereby moving from location l_1 to l_2 and possibly changing the values of the variables according to (ϕ, h):

$$\frac{\phi(\nu_1) = \mathit{true} \quad \nu_2 \in h(\nu_1) \quad \nu_1 \in \mathit{Inv}(l_1) \quad \nu_2 \in \mathit{Inv}(l_2) \quad (l_1, \alpha, (\phi, h), l_2) \in \mathit{Edg}}{(l_1, \nu_1) \rightarrow^{\alpha} (l_2, \nu_2)}$$

Time steps, written $\rightarrow^{f,t}$, describe the evolution of the values of the variables in a given location and according to an activity in that location:

$$\frac{f(0) = \nu_1 \quad f(t) = \nu_2 \quad \forall 0 \leq t' \leq t.\ f(t') \in \mathit{Inv}(l) \quad t \in \mathbb{R}^{\geq 0} \quad f \in \mathit{Act}(l)}{(l, \nu_1) \rightarrow^{f,t} (l, \nu_2)}$$

For both relations, control may stay in a location (i.e., time can progress in a location), resp. enter a location through a discrete state change, only if the invariant is not violated.

The *one-step* relation $\rightarrow$ is defined by $\rightarrow^{\alpha} \cup \rightarrow^{f,t}$. A *run* of the hybrid system H is a (finite or infinite) sequence $\rho = \sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \cdots$, with $\sigma_0 = (l_0, \nu_0) \in \mathit{Ini}$ and $\nu_0 \in \mathit{Inv}(l_0)$. We denote the set of runs of H by $[\![H]\!]$. A state $\sigma \in \Sigma$ is *reachable* in H, if there exists a run $\rho = \sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \cdots \rightarrow \sigma_n$ of H with $\sigma_n = \sigma$. We write $R(H)$ for the set of all reachable states of H.

We use $\rightarrow^n$, $\rightarrow^*$, and $\rightarrow^+$ to denote respectively the n-step relation, the reflexive-transitive closure, and the transitive closure of the one-step relation.

The semantics of hybrid systems is defined in PVS as a parameterized theory `semantics`. We list the core of this theory containing the definition of initial states, and discrete and time step relations, but elide ancillary definitions which should be clear from the context (for the full definitions we have to refer to the web resources):

PVS

```
semantics[(IMPORTING hys) H:hys]: THEORY
BEGIN
   ini_step(sigma:state[Loc(H),Vari(H)]): bool =
         Ini(H)(sigma) AND Inv(H)(loc(sigma))(val(sigma))

   disc_step(sigma1,sigma2:state[Loc(H),Vari(H)],
             e:edge[Loc(H),Vari(H),Lab(H)]): bool =
         trrel(e)((#  pre := val(sigma1),
                     post := val(sigma2) #))                AND
         Inv(H)(sourceloc(e))(val(sigma1))                  AND
         Inv(H)(targetloc(e))(val(sigma2))                  AND
         Edg(H)(e)                                          AND
         sourceloc(e) = loc(sigma1)                         AND
         targetloc(e) = loc(sigma2)

   time_step(sigma1,sigma2:state[Loc(H),Vari(H)],
             f:activity[Vari(H)], t:nonneg_real): bool =
         f(0) = val(sigma1)                                 AND
         f(t) = val(sigma2)                                 AND
         (FORALL (t1: {t1:nonneg_real|t1<=t}):
            Inv(H)(loc(sigma1))(f(t1)))                     AND
         Inv(H)(loc(sigma1))(val(sigma2))                   AND
         Act(H)(loc(sigma1))(f)                             AND
         loc(sigma2) = loc(sigma1)
   ...
END semantics
```

Before giving an example, let us fix some conventions to specify the components of the hybrid system. The standard way to describe the activities is as solutions of differential equations $\dot{\boldsymbol{x}} = g(\boldsymbol{x})$ resp. differential inclusions $\dot{\boldsymbol{x}} \in g(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$ is a vector of variables from Var and g a function from V to V, resp. from V to 2^V. We will write subsets of valuations, like the invariants of the locations,

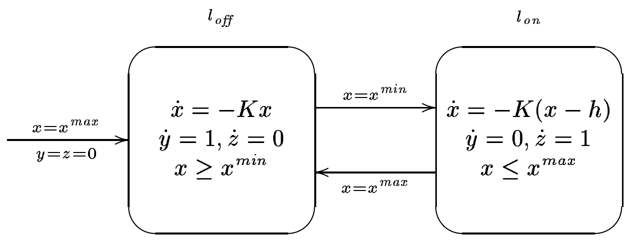

Fig. 1. Thermostat

in form of boolean predicates $\varphi : V \to Bool$. In such formulas we write short x for the evaluation $\nu(x)$. In a transition relation (ϕ, h), the effect h will be written in the form of a simultaneous, non-deterministic assignment $x_1, \ldots, x_n := g_1, \ldots, g_n$, where $x_1, \ldots, x_n \in Var$, and $g_1, \ldots, g_n$ are set-valued functions from V to $2^{\mathbb{R}}$. The relation h is then defined as the set of all valuation pairs $(\nu, \nu') \in V^2$ such that $\nu'(x_i) \in g_i(\nu)$ for all $i = 1, \ldots, n$, and $\nu(y) = \nu'(y)$ for all $y \in Var \setminus \{x_1, \ldots, x_n\}$.

Let's illustrate the definitions so far on a simple, well-tried example, the thermostat.

Example 2 (Thermostat). The temperature x of a room is controlled by a thermostat, which continuously senses the temperature and turns a heater on and off if the threshold values x^{min} and x^{max} are reached, where $x^{min} < x^{max}$ and $x^{min}, x^{max} \in \mathbb{R}^{>0}$. When the heater is off, the temperature decreases according to the function $x(t) = x_0 e^{-Kt}$, where x_0 is the initial temperature, t the time, and $K \in \mathbb{R}^{>0}$ a room constant. With the heater turned on, the temperature follows the function $x(t) = (x_0 - h)e^{-Kt} + h$, where $h > x^{min} + x^{max}$ is a real-valued constant which depends on the power of the heater. The initial temperature is x^{max} degrees and the heater is off initially. Two variables y and z serve to record the duration of time spent in the heating and the non-heating mode. The resulting hybrid system is shown in Fig. 1. By convention, trivial components of an edge $(l, \alpha, (\phi, h), l')$, i.e., $\alpha = \tau$, $\phi = true$, or $f = Id$ are not shown, and neither are stutter transitions. The same simplification is done for trivial invariants in locations.

The thermostat example is implemented by the theory `thermostat`:

PVS

```
thermostat: THEORY
BEGIN
   ...
   Loc: setof[location] =
      LAMBDA (l:location): l = l_off OR l = l_on
   Vari: setof[variable] =
      LAMBDA (k:variable): k=x OR k=y OR k=z
   Lab: setof[label] = LAMBDA (la:label): la=tau
   ...
   Inv : invariant_func[Loc,Vari] =
      LAMBDA (l:(Loc)): LAMBDA (nu:valuation[Vari]):
      IF l=l_off THEN x_min <= nu(x) ELSE nu(x) <= x_max ENDIF

   H: complete_hys =
(# Loc:=Loc, Vari:=Vari, Cont:=Cont,
           Ini:=Ini, Lab:=Lab, Edg:=Edg, Act:=Act, Inv:=Inv #)
   ...
END thermostat
```

3.3 The parallel composition of hybrid systems

Complex systems are often built from smaller components working in parallel. The parallel composition of two hybrid systems H_1 and H_2 is given by a standard product construction and written as $H_1 \times H_2$. Locations are paired and the set of variables combined. The two partners can take a common discrete step, either by synchronizing on the same label, or in that one of the contributors performs a discrete non-synchronizing transition while its partner stutters. Besides synchronizing on the label in a common discrete step, the conjunction of the actions on the variables is taken, i.e., a common step is possible only if both guards are true and if the outcome on the variables coincides. On variables it does not control, a component cannot block non-synchronizing transitions of its partner, since stutter transitions, available at each location, don't restrict the behavior of non-controlled variables. On control variables, on the other hand, stuttering is allowed only without changing the variables' values. Time transitions of the composed systems are time transitions in both systems, i.e., the activities of the composed system, restricted to the local variable sets, are activities of the component systems. Invariants of the composition finally are conjunctions of the component invariants.

Definition 3 (Parallel composition). *Let H_1 and H_2 be two hybrid systems of the forms $(Loc_i, Var_i, Con_i, Ini_i, Lab_i, Edg_i, Act_i, Inv_i)$. The* product $H_1 \times H_2$ *is the hybrid system $H = (Loc_1 \times Loc_2, Var_1 \cup Var_2, Con, Ini, Lab_1 \cup Lab_2, Edg, Act, Inv)$ such that for all $l_1, l_1' \in Loc_1$, $l_2, l_2' \in Loc_2$, $\alpha \in Lab$, $\phi \subseteq V$, $h : V \to 2^V$ and $f \in \mathcal{F}$:*

1. *$((l_1, l_2), \nu) \in Ini$ iff. $(l_i, \nu|_{Var_i}) \in Ini_i$, for $i = 1, 2$;*
2. *$Con((l_1, l_2)) = Con_1(l_1) \cup Con_2(l_2)$;*
3. *$(l_1, l_2) \longrightarrow^{\alpha}_{(\phi,h)} (l_1', l_2') \in Edg$, iff. there exist $l_i \longrightarrow^{\alpha_i}_{(\phi_i,h_i)} l_i' \in Edg_i$, such that*
   (a) $\alpha = \alpha_1 = \alpha_2$, or $\alpha_1 = \alpha \notin Lab_2$ and $\alpha_2 = \tau$, or $\alpha_1 = \tau$ and $\alpha = \alpha_2 \notin Lab_1$,
   (b) $\phi(\nu) = \phi_1(\nu|_{Var_1}) \wedge \phi_2(\nu|_{Var_2})$, and
   (c) $\nu' \in h(\nu)$, iff. $\nu'|_{Var_1} \in h_1(\nu|_{Var_1})$ and $\nu'|_{Var_2} \in h_2(\nu|_{Var_2})$;
4. *$f \in Act((l_1, l_2))$, iff. for both $i = 1$ and $i = 2$, there exist $f_i \in Act_i(l_i)$, such that $\lambda t.f(t)|_{Var_i} = f_i$;*
5. *$Inv((l_1, l_2))|_{Var_i} = Inv_i(l_i)$ for $i = 1, 2$.*

Note that by construction the set of activities $Act((l_1, l_2))$ for a composed location is time invariant, since $Act_1(l_1)$ and $Act_2(l_2)$ are. It is routine albeit tedious to show that parallel composition is associative and commutative. For a parallel composition $H_1 \times \ldots \times H_n$ with $n > 0$ and $j \in \{1, \ldots, n\}$, we call the composition system without H_j the *context* of H_j. Let Σ_H denote the state space of H. For the product system $H = H_1 \times H_2$, and a state $\sigma = ((l_1, l_2), \nu)$ of H, we write $\sigma\downarrow_{H_1} = (l_1, \nu|_{Var_1})$

and $\sigma\downarrow_{H_2} = (l_2, \nu|_{Var_2})$ for the projections on the respective components; we will use the same notation for sets of states, and analogously for runs. A basic property of the product system is that all runs of the product projected to one of the component systems are runs of that component system:

Lemma 4. *Let $H = H_1 \times H_2$ be the parallel composition of two hybrid systems H_1 and H_2. Then $[\![H]\!]\downarrow_{H_i} \subseteq [\![H_i]\!]$ and $R(H)\downarrow_{H_i} \subseteq R(H_i)$, for $i = 1, 2$.*

4 Proof System

Our approach and formalization to analyze the behavior of hybrid systems is based on Floyd's *inductive assertion method* [10]. In this classical state-based verification method one associates an assertion, i.e., a predicate over the current values of variables, with each control location of the underlying program. This gives a finite number of verification conditions to check for proving the given correctness criteria of that program. While originally developed in the context of sequential programs, the inductive assertion method serves also as fundamental technique in the analysis of concurrent programs [9]. We extend the inductive assertion method to hybrid systems.

4.1 Inductive assertional method

Let $(Loc, Var, Con, Ini, Lab, Edg, Act, Inv)$ be a hybrid system. An *assertion* on a location l is a boolean predicate over V, and an *assertion network* a predicate over Σ. For a given assertion network Q of H and a location l, let the assertion $Q_l \subseteq V$ be defined by $Q_l = \{\nu \mid (l, \nu) \in Q\}$, i.e., $\nu \in Q_l$ iff. $(l, \nu) \in Q$. Considering subsets of states as predicates on or properties of the states, we say Q_l holds for a valuation ν, in case $\nu \in Q_l$, and correspondingly for states and assertion networks. By the same token, we will speak of an assertion network implying a property etc. In connection with the system's transition semantics, an assertion network Q is *invariant*, if it holds for all reachable states, it is called *inductive*, if it holds for all initial states and is preserved under the transition relation, i.e., if for all states σ and σ' from Σ:

$$\{(l, \nu) \in Ini \mid \nu \in Inv(l)\} \subseteq Q \tag{1}$$

$$\text{if } \sigma \in Q \text{ and } \sigma \rightarrow^{\alpha} \sigma', \text{ then } \sigma' \in Q, \text{ and} \tag{2}$$

$$\text{if } \sigma \in Q \text{ and } \sigma \rightarrow^{f,t} \sigma', \text{ then } \sigma' \in Q. \tag{3}$$

Obviously, each inductive network is invariant, while the converse will, in general, not hold. The definitions of inductiveness and invariance and their connection are represented as follows:

PVS

```
verification[(IMPORTING hys) H:hys]: THEORY
BEGIN
   IMPORTING semantics[H]

   assertion: TYPE = valuation_set[Vari(H)]
   assertion_network: TYPE = [(Loc(H))->assertion]
   invariant(phi:assertion_network): bool =
      FORALL (sigma:state[Loc(H),Vari(H)]):
      reachable(sigma) IMPLIES phi(loc(sigma))(val(sigma))
   ...
   induct(Q:assertion_network): bool =
      induct_ini(Q) AND induct_edge(Q) AND induct_time(Q)

   inv_rule: LEMMA
      FORALL (Q:assertion_network):
      induct(Q) IMPLIES invariant(Q)
END verification
```

To verify a property φ for all reachable states, one can do so by finding a stronger invariant, i.e., an inductive assertion network Q which implies φ. This proof principle, known as *inductive assertion method,* is summarized in the following rule:

$$\frac{Q \to \varphi \qquad Q \text{ inductive for } H}{R(H) \to \varphi}\ \textsc{Ind}$$

In PVS, the proof method looks as follows:

PVS

```
verification_methods  : THEORY
  BEGIN
    IMPORTING ...
    simple_method: LEMMA
      FORALL (H:hys,Q,phi:assertion_network[H]):
          induct[H](Q) AND
          (FORALL (l:location): Loc(H)(l) IMPLIES
            (FORALL (nu:valuation[Vari(H)]): Q(l)(nu) IMPLIES phi(l)(nu)))
          IMPLIES invariant[H](phi)
    ...
  END verification_methods
```

It is standard to show that the rule IND is sound and complete. Note that its PVS representation contains the corresponding soundness proof as the proof of the `LEMMA simple_method`. We have to refer to the technical report [1] for the soundness and completeness proofs.

4.2 Inductive assertional proofs for parallel composition

When analyzing the parallel composition of hybrid systems, it is always possible to apply the inductive assertion method of the previous section directly on the composed system. Neglecting the structure of the composed system, however, often leads to a proliferation of the verification conditions, which reflects the state-explosion problem.

The basic idea for an improvement over the plain product of assertions for the classical programming concepts is a two-level approach, where first *local* assertion networks are checked for local consistency, and then some global consistency test (the interference freedom and the cooperation test) relates these local networks, reducing the amount of verification conditions (cf. to [9] for an exhaustive treatment).

In contrast to most applications of assertional methods, which are based on an *discrete, interleaving* model of concurrency, our method has to capture the *continuous* system evolution as well as the *synchronous* nature of hybrid systems' composition: the context of a single component cannot act independently from that component in the synchronous model, local assertions need not be shown invariant under context actions (i.e., an interference freedom test is redundant). As hybrid systems do not communicate via message passing, no cooperation test is needed, either.

An important technique, commonly called *augmentation*, which allows to speak about the peer processes of a component, is the extension of the system by fresh, otherwise unused *auxiliary* variables. As auxiliary variables are added for the sole purpose of verification, their addition must not influence the system's behavior in any way.

In the following, we will write $H' \geq H$, when H' is an augmentation of H (see [1] for detailed description). For a state set Q' of the augmented system we define the projection $Q'{\downarrow_H} = \{(l,\nu) \in \Sigma_H \mid \exists (l,\nu') \in Q'.\ \nu = \nu'|_{Var}\}$.

As the control flow and the activities for variables of H are not influenced by the auxiliary variables, the set of reachable states of H' restricted to the original variable set Var in the valuation component equals the reachable states of the original system, i.e., $R(H'){\downarrow_H} = R(H)$. Thus, a property whose satisfaction does not depend on the values for the auxiliary variables, holds for all reachable states of H', iff. it holds for all reachable states of H.

Let H_1 and H_2 be hybrid systems, $H = H_1 \times H_2$ with variable set Var their parallel composition, and Q_1 and Q_2 assertion networks for H_1 and H_2, respectively. We define the composition of the local assertion networks as $Q_1 \times Q_2 = \{\sigma \in \Sigma_H | \sigma{\downarrow_{H_1}} \in Q_1 \wedge \sigma{\downarrow_{H_2}} \in Q_2\}$. Note that $Q_1 \times Q_2$ is an assertion network of H. Now let $\varphi \subseteq \Sigma_H$ a predicate on the set of H's states. Then φ is an invariant of H if and only if there exists an auxiliary variable set Var_{aux}, hybrid systems H_1' and H_2', such that $H' = H_1' \times H_2'$ is an augmentation of H with Var_{aux}, and inductive assertion networks Q_1' and Q_2' of H_1' and H_2', respectively, such that $(Q_1' \times Q_2'){\downarrow_H} \subseteq \varphi$. With these conventions, we can formulate the proof rule to deal with the parallel composition of systems.

$$\frac{\begin{array}{c} H_1' \times H_2' \geq H_1 \times H_2 \\ Q_1' \ \mathit{inductive\ for}\ H_1' \quad Q_2' \ \mathit{inductive\ for}\ H_2' \\ (Q_1' \times Q_2'){\downarrow_{H_1 \times H_2}} \rightarrow \varphi \end{array}}{R(H) \rightarrow \varphi}\ \textsc{Comp}$$

Proposition 5. *The proof rule* (COMP) *is sound and complete.*

For the proof of soundness and completeness we refer to the technical report [1].

5 Verification in PVS

Next we sketch the hierarchical structure of the main theories in the PVS implementation of our proof methods and give an overview of the examples verified within PVS.

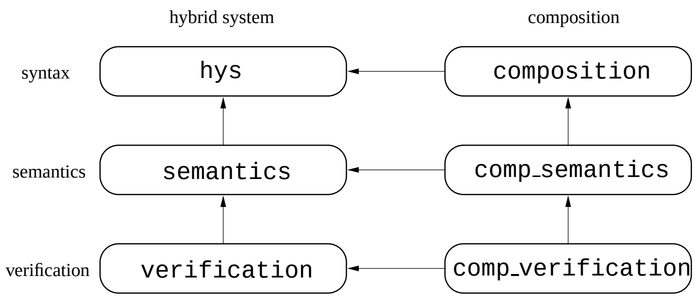

Fig. 2. Structure of the proof system in PVS

5.1 Structure of the proof system in PVS

In general, the dependencies of the modules mirror the order of definitions and lemmas as presented in the previous sections (or rather the paper follows the structure of the PVS-theories). Fig. 2 gives a overview of the main components.

The basis of the formalization are the theories containing the definition of hybrid systems and their parallel composition. These modules are imported into the definition for the semantics, both for hybrid systems and their parallel composition. The semantics of one instance of a hybrid system is defined as a separate theory parameterized in this instance (cf. the code fragment in Section 3). The theories defining the proof rules for hybrid systems and their parallel composition import the above basic definitions.

5.2 Example

Besides formalizing the proof rules in PVS, we applied the method to a number of examples, e.g., non-linear variations of the water level monitor [2], or a modified clock synchronization of the MPEG4 standard. The PVS formalization of these examples and the verified properties are available on the web-site and in [1]. In the following, we describe in more detail a simple example of a non-linear, composed hybrid system, which computes a linear approximation of a non-linear function.

The approximator is the parallel composition of two hybrid systems, H_{input} and H_{approx}. The first one changes the value of a variable x according to activities with derivation in $[-\alpha, \alpha]$, where $\alpha > 0$. The second one reads the value of x periodically after each time interval dT and approximates the value of x linearly, based on the last two received values. The approximated value is represented by the variable y. Variables x_0 and y_0 store the value of x and y respectively at the time instance of the last synchronization point. For measuring the time we introduce the clock variable z. Fig. 3 shows the resulting hybrid system.

The property we are interested in is that the approximation error does not exceed a certain margin, i.e., for each state the invariant $|x - y| \leq \alpha \cdot dT$ holds.

In order to verify this property we define an assertion (network) for H_{input}, which expresses some boundaries for the value of x:

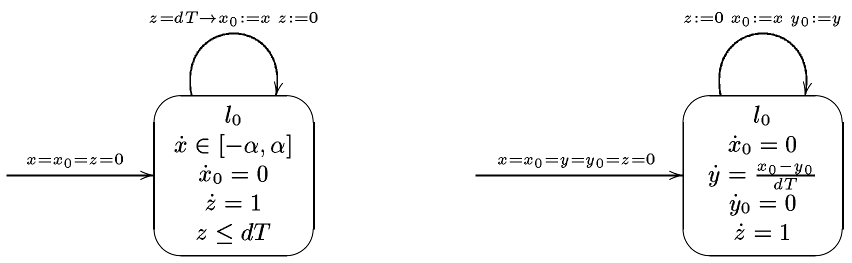
$z{=}dT \to x_0{:=}x \; z{:=}0$
$x{=}x_0{=}z{=}0$
l_0
$\dot{x} \in [-\alpha, \alpha]$
$\dot{x}_0 = 0$
$\dot{z} = 1$
$z \leq dT$
$z{:=}0 \; x_0{:=}x \; y_0{:=}y$
$x{=}x_0{=}y{=}y_0{=}z{=}0$
l_0
$\dot{x}_0 = 0$
$\dot{y} = \frac{x_0 - y_0}{dT}$
$\dot{y}_0 = 0$
$\dot{z} = 1$

Fig. 3. Linear approximation

$$Q_{input}(l_0) = x_0 - \alpha z \leq x \leq x_0 + \alpha z.$$

With H_{approx}, we associate a predicate which expresses that the invariant was valid for the last synchronization point and defines an upper boundary for the actual approximation error:

$$\begin{aligned} Q_{approx}(l_0) = \;& y = y_0 + \tfrac{x_0 - y_0}{dT} z \wedge \\ & |x_0 - y_0| \leq \alpha \cdot dT \wedge \\ & |x - y| \leq \tfrac{|x_0 - y_0|}{dT}(dT - z) + \alpha z. \end{aligned}$$

6 Conclusion

As the main line of research on hybrid systems focuses on model checking techniques for appropriately restricted subclasses, there are less investigations on deductive methods for their verification. In this paper we present an assertional deductive proof method for the verification of hybrid systems. Especially for the verification of composed systems, we give a complete proof rule to reduce the complexity introduced by the parallel composition. To facilitate the tedious verification of those system without restricting the model artificially, we embedded the proof system into the PVS theorem prover. Beside offering the full power of higher-order logic, a further advantage of such a deductive verification environment is that it allows a straightforward rigorous formalization of the mathematical definitions, without the need to resort to any specific logic. Furthermore, PVS comes equipped with a wide range of automated proof-strategies and heuristics.

Related Work Closest in spirit to our work is [5], which embed timed automata into PVS and apply their approach to the steam boiler example. The same example is treated in [33], with the goal of deriving an implementation of a real-time program in a number of refinement steps [19]. The PVS theorem prover is also used in [17] in combination with model checking using HyTech [4] for the reachability analysis for various classes of linear hybrid automata. For the verification of safety properties of hybrid systems, [20] employ hybrid temporal logic HTL, an extension of interval temporal logic. They give a number of proof-rules which they prove sound. Likewise building upon temporal logic, [24] use the Stanford theorem prover STeP as proof environment. Besides the verification of safety and liveness properties, [31] contains a deeper discussion of the connection of hybrid systems and the field of control theory and presents proof concepts for stability and attraction properties of hybrid systems (cf. also the contribution [32] in this volume). [22] surveys various deductive and algorithmic approaches for the verification of hybrid systems.

Future Work As for future work, we intend to apply our method to larger case studies, especially to extend the control example based on MPEG4 of [8], and further a laser steering system for mass spectroscopy. To improve the specification structure of hybrid systems, the interface information can be extended, for instance separating the variable set into input and output variables like in [23]. Such a cleaner separation is a necessary prerequisite for the development of an assume-guarantee reasoning scheme [30]. Furthermore, we expect that the verification will benefit from an alternative semantics allowing for compositional proofs [13].

References

1. E. Ábrahám-Mumm, U. Hannemann, and M. Steffen. Verification of hybrid systems: Formalization and proof rules in PVS. Technical Report TR-ST-01-1, Lehrstuhl für Software-Technologie, Christian-Albrechts-Universität Kiel, Jan. 2001.
2. R. Alur, C. Courcoubetis, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
3. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:252–235, 1994.
4. R. Alur, T. A. Henzinger, and P. Ho. Automatic symbolic verification of embedded systems. In *Proc. 14th Annual Real-Time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993.
5. M. Archer and C. Heitmeyer. Verifying hybrid systems modeled as timed automata: A case sudy. In O. Maler, editor, *Hybrid and Real-Time Systems (HART'97)*, LNCS 1201, pages 171–185. Springer, 1997.
6. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
7. The Coq project. http://pauillac.inria.fr/coq/, 1998.
8. J. B. de Meer and E. Ábrahám-Mumm. Formal methods for reflective system specification. In J. Grabowski and S. Heymer, editors, *Formale Beschreibungstechniken für verteilte Systeme*, pages 51–57. Universität Lübeck/Shaker Verlag, Aachen, Juni 2000 2000.
9. W.-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Proof Methods*. Cambridge University Press, 2001. to appear.
10. R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Proc. Symp. in Applied Mathematics*, volume 19, pages 19–32, 1967.
11. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*. Cambridge University Press, 1993.
12. R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors. *Hybrid Systems*, LNCS 736. Springer, 1993.

13. U. Hannemann. *Semantic Analysis of Compositional Proof Methods for Concurrency*. PhD thesis, Utrecht Universiteit, Oct. 2000.
14. T. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. In R. Alur, T. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pages 377–388, 1996.
15. T. A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In P. Wolper, editor, *Proceedings of CAV '95*, LNCS 939, pages 225–238. Springer, 1995.
16. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata. In *27th Annual ACM Symposium on Theory of Computing*. ACM Press, 1995.
17. T. A. Henzinger and V. Rusu. Reachability verification for hybrid automata. In Henzinger and Sastry [18], pages 190–204.
18. T. A. Henzinger and S. Sastry, editors. *Proceedings of the First International Workshop on Hybrid Systems: Computation and Control (HSCC'98)*, LNCS 1386. Springer, 1998.
19. J. Hooman. A compositional approach to the design of hybrid systems. In Grossman et al. [12], pages 121–148.
20. A. Kapur, T. A. Henzinger, Z. Manna, and A. Pnueli. Proving safety properties of hybrid systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Proceedings of FTRTFT '94*, LNCS 863, September 1994. Springer.
21. Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: a class of decidable hybrid systems. In Grossman et al. [12], pages 179–208.
22. S. Kowalewski, P. Herrmann, S. Engell, H. Krumm, H. Treseler, Y. Lakhnech, R. Huuck, and B. Lukoschus. Approaches to the formal verification of hybrid systems. *at-Automatisierungstechnik. Special Issue: Hybrid Systems II: Analysis, Modeling, and Verification*, 49(2):66–74, Feb. 2001.
23. N. A. Lynch, R. Segala, and F. W. Vaandrager. Hybrid I/O automata revisited. In M. D. D. Benedetto and A. Sangiovanni-Vincentelli, editors, *Proceedings of HSCC 2001*, LNCS 2034. Springer, 2001.
24. Z. Manna and H. B. Sipma. Deductive verification of hybrid systems using STeP. In Henzinger and Sastry [18].
25. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *Proceedings of CADE '92*, LNCS 607, pages 748–752. Springer, 1992.
26. S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Manual*. Computer Science Laboratory, SRI International, Menlo Park, CA, Sept. 1999.
27. L. C. Paulson. The Isabelle reference manual. Technical Report 283, University of Cambridge, Computer Laboratory, 1993.
28. O. Roux and V. Rusu. Uniformity for the decidability of hybrid automata. In R. Cousot and D. A. Schmidt, editors, *Proceedings of SAS '96*, LNCS 1145, pages 301–316. Springer, 1996.
29. J. M. Rushby, 2001. available electronically at http://www.csl.sri.com/papers/pvs-bib/.
30. N. Shankar. Lazy compositional verification. In W.-P. de Roever, H. Langmaack, and A. Pnueli, editors, *Compositionality: The Significant Difference (Compos '97)*, LNCS 1536, pages 541–564. Springer, 1998.
31. T. Stauner. Properties of hybrid systems - a computer science perspective. Technical Report TUM-I0017, Technische Universität München, Nov. 2000.
32. T. Stauner. Hybrid systems' properties – classification and relation to computer science. In *Proceedings of Eurocast '01)*, Lecture Notes in Computer Science. Springer, 2001.
33. J. Vitt and J. Hooman. Assertional specification and verification using PVS of the steam boiler system. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control System*, LNCS 1165, pages 453–472. Springer, 1996.