Verifying safety properties of a nonlinear control by interactive theorem proving with the Prototype Verification System*

Cinzia Bernardeschi Andrea Domenici†

March 29, 2019

Abstract

Interactive, or *computer-assisted*, theorem proving is the verification of statements in a formal system, where the proof is developed by a logician who chooses the appropriate inference steps, in turn executed by an automatic theorem prover. In this paper, interactive theorem proving is used to verify safety properties of a nonlinear (hybrid) control system.

Keywords Theorem Proving, Verification, Nonlinear Control, Prototype Verification System

1 Introduction

Many technical systems fall in the class of *hybrid systems*, i.e., nonlinear systems having both analog and digital components. Such systems are typically composed of an analog plant, described by linear or nonlinear equations, and a digital control, intrinsically nonlinear. In industrial practice, hybrid systems are usually analyzed by simulation. An executable model of the system is built with graphical block-based languages such as those offered by the Simulink(TM), Scilab, or ScicosLab environments [45, 11], or textual languages such as Modelica [20] or CIF [44], or a combination of the two, and the model is executed to simulate the system under various conditions.

While simulation is a mainstay of system development and is a necessary tool for validation, it cannot provide developers with the confidence afforded by formal verification. Formal verification of nonlinear systems may be difficult, but automatic or semiautomatic tools can provide valuable support to this task.

Schupp *et al.* [38] recently published an overview of hybrid systems verification, with short outlines of tools and techniques for reachability analysis,

*Postprint. Published in:Information Processing Letters Volume 116, Issue 6, June 2016, Pages 409-415. DOI: https://doi.org/10.1016/j.ipl.2016.02.001.

†C. Bernardeschi and A. Domenici are with the Department of Information Engineering, University of Pisa, Italy. E-mail: {cinzia.bernardeschi,andrea.domenici}@ing.unipi.it

examples of benchmark problems, and current challenges. A survey of works on formal verification of hybrid systems was published by Alur [1], who identifies some broad areas of research, including *symbolic reachability analysis* and *deductive verification*.

In the area of symbolic reachability analysis, research is focused on algorithms to compute or approximate a system's *reach*(*ability*) *set*, i.e., the set of states reachable from any of the admissible initial states, with the goal of verifying whether the set contains unsafe states. For example, Tiwari and Khanna [43] propose techniques to approximate reach sets for different classes of hybrid automata, based on qualitative abstraction [42], which in turns relies on model checking. Model checking is also used by Cimatti *et al.* [15], who implement a quantifier-free encoding of hybrid automata with the NuSMV [14] model checker.

Many tools have been developed to support the analysis of hybrid systems, including UPPAAL [3] for timed automata, HybridSAL [41] based on the SAL [4] model checker, ARIADNE [5], and HSOLVER [37]. In particular, ARIADNE has been used for nonlinear hybrid system verification based on an assume-guarantee method, and HSOLVER has been applied to safety verification with constraint propagation and abstraction refinement.

In the area of deductive verification, KeYmaera [36] is an interactive theorem-proving environment based on sequent calculus and tailored to the differential dynamic logic d$\mathcal{L}$ [35]. KeYmaera has been developed specifically for hybrid systems, unlike other general-purpose theorem provers, such as Coq [34], based on the calculus of inductive constructions and intuitionistic logic, and Isabelle [28], based on higher-order logic and functional programming.

In this paper, the PVS (Prototype Verification System) theorem prover is used to prove basic properties of a typical case study, the level control of a storage tank. This simple example shows that a higher-order theorem-proving tool can support developers in expressing and verifying a natural line of reasoning rooted on domain knowledge.

This paper is structured as follows: In Sec. 2, essential information on the PVS language and deduction system is provided; Sec. 3 introduces the case study; Sec. 4 describes the formalization of the case study and how the PVS is used to prove that certain constraints guarantee safe operation of the system; Sec. 5 discusses the case study and relates it to the general topic of hybrid system analysis; and Sec. 6 concludes the paper.

2 The Prototype Verification System

The PVS is an interactive theorem prover developed at Computer Science Laboratory, SRI International, by S. Owre, N. Shankar, J. Rushby, and others [31, 30] and it has been applied to many fields, including formal verification of hardware and safety-critical systems [40, 12, 13]. Its formal system is based on sequent calculus [24, 22, 23], together with a typed higher-order language.

A PVS user writes a *theory* in the PVS language [33], then uses the PVS theorem proving environment [39] to prove selected formulas of the theory.

2.1 The PVS Language

In a PVS theory, one can declare *types*, *constants*, *variables*, and *formulas*. The PVS type system is very flexible, providing users with standard mathematical types (e.g., naturals, integers, and reals) and allowing them to define *uninterpreted* types, to build *record* and *tuple* types similar to records in programming languages, and to define *function* types (e.g., "*the set of functions from integers to reals*"). In particular, functions returning Boolean values are called *predicates*. It is also possible to define *subtypes* by adding constraints to previously defined types. One can then declare constants and variables (including function constants and variables) and write formulas. A formula is a named logical statement composed of atomic formulas, logical connectives, and quantifiers.

Each formula is identified by a name and qualified by a keyword specifying if the formula is an *axiom* or not. The PVS prover takes axioms as proved statements, whereas it requires the other formulas to be proved. Axioms are recognized by the `AXIOM` keyword, the other formulas by such keywords as `LEMMA`, `THEOREM`, or other synonyms. Examples of PVS declarations are found in Sec. 4.

The PVS environment includes a large number of pre-packaged fundamental theories, called the *prelude* [32]. An even larger number of theories, covering, e.g., mathematical analysis, algebra, or probability, is available in additional libraries, such as the NASA Langley PVS Library [19, 25].

2.2 The PVS Deduction System

As previously mentioned, PVS is based on the sequent calculus. A *sequent* is an expression with this structure:

$$A_1, A_2, \ldots, A_m \vdash B_1, B_2, \ldots, B_n$$

where the A_i's are the *antecedents* and the B_i's are the *consequents*. The '$\vdash$' symbol is called a *turnstile* and may be read as "*yields*". Each antecedent or consequent is a formula built with atomic formulas, connectives, and quantifiers.

A sequent is true if any formula occurs both as an antecedent and as a consequent, or any antecedent is false, or any consequent is true. Proving a formula (a *goal*) consists in expressing it as a sequent without antecedents and applying inference rules until one of the previous conditions for truth is met.

The PVS prover presents the user with the initial sequent corresponding to the formula to be proved. The user applies a series of inference steps, invoking a prover command at each step. A prover command may result in the application of a single inference rule of the sequent calculus, or a combination of several rules, possibly chosen and iterated according to some pre-packaged *strategy*. Some of the manipulations made available by the PVS prover include: (i) *Instantiating variables*, in particular by introducing fresh *Skolem* constants; (ii) *decomposing formulas* into simpler ones; (iii) *introducing lemmas*; and (iv) *applying substitutions*. Some commands transform the current goal into two or more *subgoals*: For example, the `split` command transforms a goal of the form $A \Rightarrow B \vdash C$ into two subgoals $B \vdash C$ and $\vdash A, C$.

Usually, a PVS user directs the proof by making informed choices about the main steps (such as introducing the appropriate lemmas) and lets the prover deal with low-level, tedious and error-prone manipulations. The prover, however, supports also high-level proof strategies, such as induction.

3 Water level control

The problem of controlling the level of a liquid (say, water) in a tank is a well-known case study in control theory. There are several versions of this problem, and in this paper the one presented in [10] is considered. The problem is stated as follows:

> A cylindrical storage tank receives water with a maximum volume flow rate C. Water can be drained out with the same maximum flow rate C. The incoming flow rate $w_{\mathrm{i}}(t)$ may vary in time arbitrarily (within the mentioned limit), while the outgoing flow rate w_{o} is regulated by a valve according to the law $w_{\mathrm{o}}(t) = \mathrm{C}v(t)$, where $v \in [0, 1]$ is the valve position, with $v = 0$ when the valve is fully closed and $v = 1$ when fully open. A level control must ensure that the water level remains between the minimum and maximum levels L_1 and L_2, respectively. The control consists in a level sensor and a valve actuator. The sensor output $k(t)$ is -1, 0, or 1 if the level is below, equal to, or above the reference level $\mathrm{L} = (\mathrm{L}_1 + \mathrm{L}_2)/2$. The actuator opens or closes the valve according to the law $v'(t) = k(t)$, where the prime symbol (') stands for derivation.

In addition to the above description, it is assumed for simplicity that the outgoing flow does not depend on the water level, and that the outgoing pipe is always completely filled.

In this case study, the value of the water level as a function of time is the solution of the differential equation $l'(t) = w_{\mathrm{i}}(t) - w_{\mathrm{o}}(t)$, where $w_{\mathrm{i}}(t)$ is arbitrary, and $w_{\mathrm{o}}(t)$ is nonlinear, as it depends on the sensor output $k(t)$, which can be defined as:

$$k(t) = \begin{cases} -1 & \text{if } l(t) < L \\ 0 & \text{if } l(t) = L \\ 1 & \text{if } l(t) > L \end{cases}$$

This system is physically simple, but hard to analyze with the standard approaches of linear control theory, so in a practical setting it would most likely be studied by simulation. In [21], for example, it has been described and simulated with the Modelica language. The next section will show how a simple reasoning supported by computer-assisted theorem proving enables developers to prove a relationship among system parameters (namely, initial level, maximum flow rate, and level bounds) that ensures correct operation. This relationship is proved symbolically, therefore it has a general validity.

4 Safety property verification

The procedure to verify the safety properties of the water level control can be summarized as follows:

1. As a preliminary step, the worst-case situations possibly leading to violation of safety requirements are identified, the law of continuous evolution for the system (or a safe approximation, in more complex cases) in those situations is defined, and constraints ensuring satisfaction of the safety requirements are found.
2. A theory is defined in the PVS language, and the safety theorems are expressed, with the conjunction of the above law and constraints taken as hypotheses, and the safety requirements taken as theses.
3. Intermediate lemmas are found and proved, recurring to axioms of the specific theory and of general theories (e.g., math analysis).
4. The theorems are proved from lemmas and axioms.

The safety property to be verified is that the water level remains within the specified limits at all times, provided that the water intake satisfies the specified constraint. More precisely, a relationship among the system parameters is found, which guarantees the safety property.

The above steps are described in the following subsections.

4.1 Step 1: Safety requirements

The problem statement (Sec. 3) specifies the laws governing the system and its physical constraints. The constraint $\mathrm{L}_1 \leq l(t) \leq \mathrm{L}_2$ is the conjunction of two requirements, one forbidding depletion of the tank, and one forbidding overflow. In order to find relationships that guarantee satisfaction of the requirements, the worst-case situations that could lead to depletion or overflow are considered: (i) No incoming flow, initial level below reference, and fully open valve in case of depletion; and (ii) maximum incoming flow, initial level above reference, and fully closed valve in case of overflow. These situations may arise if the initial level and the valve position are set before activation of the sensor at time $\mathrm{t_i} = 0$.

Let us consider situation (i). In this case, the valve will close linearly wrt to time, the outgoing flow will decrease linearly, and the water level will then vary quadratically. The requirement that the level does not fall below L_1 reduces to a quadratic inequality, and elementary algebra provides the sought assumption on the initial level, i.e., $\mathrm{L_i} > \mathrm{L}_1 + \mathrm{C}/2$.

4.2 Step 2: Logic modeling

The next step is the definition of a theory modeling the system. First, the system parameters, declared as constants, the variable representing time, and the water level function:

```
C:    posreal              % max flowrate
L1:   posreal              % minimum level
L2:   posreal              % maximum level
L:    posreal = (L1+L2)/2  % reference level
L_i:  posreal              % initial level
V_i:  nnreal               % initial valve posn
t:    VAR real             % time
l(t): real                 % water level
```

Types `posreal` and `nnreal` are the positive and non-negative real numbers, respectively.

The sensor and valve specifications follow:

```
signum(x: real): integer =
  COND
  x < 0 -> -1,
  x = 0 ->  0,
  x > 0 ->  1
  ENDCOND
k(t): integer         % sensor output
    = signum(l(t)-L)
v(t): real            % valve position
valve_law: AXIOM
  deriv(v) = k        % derivative of v(t)
```

where the `COND/ENDCOND` block is the PVS case selection statement.

Then, the storage tank specifications:

```
w_in(t):  real              % input flowrate
w_out(t): real = C*v(t)     % output flowrate
level_law: AXIOM
  deriv(l) = w_in - w_out   % derivative of l(t)
```

Finally, some axioms (not shown) on the mathematical well-behavedness of the various functions, and the basic relationships among system parameters:

```
level_bounds: AXIOM
  L1 < L2
init_level: AXIOM
  l(0) = L_i
init_valve_posn: AXIOM
  v(0) = V_i
```

The above definitions are the theory against which the safety property must be verified, under some assumptions on the initial water level. It is then possible to verify satisfaction of the requirement against tank depletion by interactively proving the following theorem:

```
no_depletion: THEOREM
  forall (t: real):
    (v = (lambda (x): (-1)*x + 1) % (1)
     and w_in = const_fun(0)      % (2)
     and L_i > L1 + C/2           % (3)
     IMPLIES
     l(t) >= L1)                  % (4)
```

We observe how the lambda notation and function `const_fun(c)` (identically equal to parameter `c`) are used to distinguish the definitions of functions v (`v`) and w_{i} (`w_in`) from their application to arguments.

In the above theorem, line (1) asserts that the valve position has the form $1 - x$, line (2) asserts that the incoming flow is identically zero, line (3) is the assumption relating the initial level to the lowest tolerated level and the maximum possible flow, and line (4) is the safety requirement. By similar reasoning, the following theorem can be formulated for the requirement against overflow:

```
no_overflow: THEOREM
  forall (t: real):
    (v = (lambda (x): x)
     and w_in = const_fun(C)
     and L_i < L2 - C/2
     IMPLIES
     l(t) <= L2)
```

4.3 Step 3: Intermediate lemmas

The proof of Theorem `no_depletion` requires that two main lemmas (plus a few secondary ones) are preliminarily proved.

Lemma `level_fun` proves that $l(t)$ is a quadratic form:

```
level_fun: LEMMA
  v = (lambda (x): (-1)*x + 1)
  and w_in = const_fun(0)
  IMPLIES
  l(t) = (C/2)*t^2 - C*t + l(0)
```

The lemma is proved by invoking the `level_law` axiom, plus a few simple lemmas on integration from the NASA Langley library.

Lemma `neg_discr` proves that the discriminant of $l(t) - \mathrm{L}_1$ is negative if condition (3) of the theorem is satisfied:

```
neg_discr: LEMMA
  L_i > L1 + C/2
  IMPLIES
  discr(C/2, -C, L_i-L1) < 0
```

Also this lemma is proved with lemmas from the NASA Langley library, and commands from the PVS *manip* package, which extends the prover with algebraic manipulation steps [17, 18].

4.4 Step 4: Proving properties

The proof of the theorem can then begin by instantiating the initial sequent with a Skolem constant (`t!1`, automatically introduced by the prover) and decomposing it into antecedent and consequent formulas:

```
{-1}  v = (lambda (x): (-1)*x + 1)
{-2}  w_in = const_fun(0)
{-3}  L_i > L1 + C/2
  |-------
{1}   l(t!1) >= L1
```

Note that, in the PVS user interface, antecedent and consequent formulas are separated by a dashed line, stacked vertically, and labeled numerically. Antecedents have negative labels. The label of a formula may change as formulas are rearranged in the course of a proof, and curly braces highlight newly introduced or transformed formulas.

Then, lemma `level_fun` is introduced and instantiated (Formula {-1}):

```
{-1}  v = (lambda (x): (-1)*x + 1)
      AND w_in = const_fun(0)
      IMPLIES
      l(t!1) = (C/2)*t!1^2 - C*t!1 + l(0)
[-2]  v = (lambda (x): (-1)*x + 1)
[-3]  w_in = const_fun(0)
[-4]  L_i > L1 + C/2
  |-------
[1]   l(t!1) >= L1
```

Splitting the implication in Antecedent {-1}, three subgoals are produced, the first one being the following:

```
{-1}  l(t!1) = (C/2)*t!1^2 - C*t!1 + l(0)
[-2]  v = (lambda (x): (-1)*x+ 1)
[-3]  w_in = const_fun(0)
[-4]  L_i > L1 + C/2
  |-------
[1]   l(t!1) >= L1
```

By substitution and elementary manipulations, it is then possible to express the theorem's thesis as a quadratic inequality (Formula {1}):

```
[-1]  v = (lambda (x): (-1)*x + 1)
[-2]  w_in = const_fun(0)
[-3]  L_i > L1 + C/2
  |-------
{1}   (C/2)*t!1^2 - C*t!1 + l(0) - L1 >= 0
```

This sequent is proved by invoking Lemma `neg_discr` and Axiom `init_level`. The remaining two subgoals introduced by implication splitting are immediately recognized as true by the PVS prover:

```
[-1]  v = (lambda (x): (-1)*x + 1)
[-2]  w_in = const_fun(0)
[-3]  L_i > L1 + C/2
  |-------
{1}   v = (lambda (x): (-1)*x + 1)
[2]   l(t!1) >= L1
```

and

```
[-1]  v = (lambda (x): (-1)*x + 1)
[-2]  w_in = const_fun(0)
[-3]  L_i > L1 + C/2
  |-------
{1}   w_in = const_fun(0)
[2]   l(t!1) >= L1
```

Theorem `no_overflow` is proved along the same lines.

In the two proofs, the developer's task was to understand the overall structure of the proof and select the relevant axioms and lemmas. Long and repetitious sequences of small inference steps have been dealt with by single commands, relieving the developer of their burden and potential mistakes. Also, it may be interesting to know that a first attempt was made to prove the safety property under weaker assumptions, namely, $L_i \geq L_1$ for Theorem `no_depletion` and $L_i \leq L_2$ for Theorem `no_overflow`, but failed proofs led to reconsidering the assumed behavior of the system and finding the right assumptions, which take the maximum flow rate C into account.

5 Discussion

This proof-of-concept example has been treated with a heuristic (not to say naïve) approach. In particular, no use has been made of the well-established theory of hybrid systems.

The first step in the above section represents the safety analysis part of the development process, carried out independently of the later verification phase and its tools. The safety properties have been defined only on the two states identified as worst-case situations, instead of expressing them, as is usually done,

as global constraints on the whole set of reachable states. Clearly, this has been made possible by the simplicity of the physical system. In more realistic cases, it would be difficult to find such a set of "extreme" states, whose safety guarantees safety in all other states. In fact, the algorithmic methods used for hybrid system verification are based on the *automatic* generation of the state space (or of safe approximations), thus relieving developers of the need to figure out the most relevant states for safety analysis.

This simple example, however, shows that a higher-order theorem-proving tool can support developers in expressing and verifying a natural line of reasoning rooted on domain knowledge. In more complex, but still tractable problems, this kind of reasoning could lead to a better insight of the physical problem, which might not be gained through the use of automatic tools, more machine-intensive and less human-intensive.

The hybrid-systems theory becomes indispensable when system complexity makes the heuristic approach impracticable. In this case, both a description of the system as a hybrid automaton and its safety properties can be formalized in higher-order logic in a simple and uniform way. For example, Masci *et al.* [8] have modeled an implantable pacemaker as a system of timed automata. The automata are defined by their locations, the guards and invariants, and the clock variables. All these are expressed in PVS with uniform patterns: locations are represented by values of an enumerated type, states are represented as records whose members return the current location and the current values of clock variables, and so on (in PVS, record members are just another syntax for functions). This method of representing timed automata in logic can be extended to hybrid automata, and it can be mechanized to various extent, depending on the type of automaton. A system model in a graphical block language, or a textual one, could be translated into a logic model. PVSio-web [29] is an example of a tool that produces a logical model of state machines (not yet hybrid automata) from a graphical language [26] derived from Stateflow(TM).

While generation of logic models from other languages can be almost completely automatized in a straightforward way, using such models is a more complex issue.

The two main hurdles for the application of interactive theorem proving are learning the language and learning proof strategies. Formal logic is rarely included in engineering syllabi, and the PVS language is quite complex. However, much of its complexity is due to its wide-ranging applicability and to the richness (or complexity) of its syntax, which is very precise and often offers a few variant forms to express a given meaning. In the authors' experience, the main difficulty was in learning the subtleties of the rigorous higher-order type system. It may be argued, however, that a systems developer does not have to learn the whole language and all its possible usages. If a standard formalization of hybrid systems is adopted, someone using interactive theorem proving for safety analysis could learn just the concepts and notations needed to understand and use that formalization.

The harder issue of proving theorems can be tackled in a similar way. To most engineers, applying theorems is much easier than proving them, but math-

ematicians know that there are standard patterns of demonstration for many classes of theorems. Similarly, standard proof patterns can be found for classes of hybrid systems and for classes of properties to be verified.

Actually, one of the main goals in the authors' work is finding useful patterns both for system modeling and for system verification, and packaging the latter in the form of proof strategies that can be programmed into the PVS prover [2] and invoked with simple commands.

Another issue related to proof strategies concerns the different proof styles needed for the discrete and the continuous parts of hybrid systems. Most reported applications of the PVS concern digital systems, but applications to continuous systems have also been made, for example in air traffic control [27]. Proofs in the area of digital systems typically rely on library theories for Boolean and integer algebra and on induction, whereas in the area of continuous systems, proofs rely on library theories for mathematical analysis. The NASA Langley library offers a large collection of theories on analysis, but at the time of writing no collection of theories specifically devoted to differential equations was available. This means that differential problems must be solved applying the basic theorems on integration and differentiation provided by the library, which may require some creativity. When analytical solutions cannot be found, approximation methods can be formalized using other library theories, such as those on interval arithmetic [16].

Interactive theorem proving can be seen as a complement to the better established techniques based on algorithmic construction of abstractions or approximations of a system, in particular of its reachability set. Its main advantage is generality, in terms both of results and of applicability. Results are general since they are usually expressed in terms of symbolic quantities. For example, in the case discussed in this paper, the safety requirement against depletion was found to hold for any triple of free parameters L_i, L_1, C satisfying $L_i > L_1 + C/2$. Applicability is general because the laws of logical deduction are intrinsically general. As explained above, it is highly desirable to have pre-packaged theories and proof strategies available, specialized for particular classes of problems, but when a problem does not fit into anyone of those classes, it is always possible to (try to) devise new ways to solve it, at the price of a harder effort, whereas more automatic techniques are often capable of dealing only with some particular types of system.

Another feature of theorem-proving environments, and of the PVS in particular, is modularity. Separate theories can be defined for different subsystems or for different aspects of a (sub)system, and each theory can be referred to by other theories. So, the overall model of a system can be decomposed in (or built from) a number of theories for each subsystem, plus a co-ordinating theory for the overall system. Or, a separate theory can be defined to solve one hard differential equation, or to simplify a set of constraints. Specific theorems can be proved in each subtheory, and then be used as lemmas to prove the main verification goals. It is also likely that such intermediate lemmas can be reused for different systems.

A more technical feature is that in the deductive approach it is possible to

prove properties of a system's state space, but the state space does not have to be generated, or approximated, or abstracted, as is the case for the tools based on reachability analysis, which generally are limited in their applicability by the problem of state-space explosion.

The issue remains of the intrinsic difficulties of learning language and techniques for interactive theorem proving, as discussed earlier. It should be borne in mind, however, that almost every verification tool has its own special language and requires some experience to be used proficiently, and also that many tools lack the generality of theorem proving.

6 Conclusions

Formal verification has long been advocated as an important tool in the development of control systems, and recommended by safety standards. However, its adoption as a standard industrial practice is lagging behind model-based simulation, due in part to the perceived complexity of its tools and methodologies. This paper shows how a state-of-the-art theorem proving environment can be an effective tool, providing control systems developers with the ability to prove in a rigorous but natural way some general properties that simulation can validate only for specific cases.

This work is part of an effort aimed at the application of formal methods to modeling and verification of safety- or mission-critical systems, including control logics [7, 6], electromedical devices [8], and integrated clinical environments [9]. Further work will focus on investigating methodologies and proof techniques tailored to diverse application domains, still in the area of safety-critical systems.

Acknowledgments

The authors wish to thank the anonymous referees for their valuable comments and suggestions.

References

[1] Rajeev Alur. Formal verification of hybrid systems. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, pages 273–278, New York, NY, USA, 2011. ACM.

[2] Myla Archer and Ben Di Vito. Developing user strategies in pvs: A tutorial. In *In: Proceedings of the First International Workshop on Design and Application of Strategies/Tactics in Higher Order Logics (STRATA*, 2003.

[3] G. Behrmann, A. David, K.G. Larsen, J. Hakansson, P. Petterson, Wang Yi, and M. Hendriks. UPPAAL 4.0. In *Third International Conference on Quantitative Evaluation of Systems (QEST 2006)*, pages 125–126, Sept 2006.

[4] Saddek Bensalem, Vijay Ganesh, Yassine Lakhnech, Cesar Muñoz, Sam Owre, Harald Rueß, John Rushby, Vlad Rusu, Hassen Saïdi, N. Shankar, Eli Singerman, and Ashish Tiwari. An Overview of SAL. In *Proceedings of the Fifth NASA Langley Formal Methods Workshop (LFM 2000)*, pages 187–196, 2000.

[5] Luca Benvenuti, Davide Bresolin, Pieter Collins, Alberto Ferrari, Luca Geretti, and Tiziano Villa. Assume-guarantee verification of nonlinear hybrid systems with ARIADNE. *Int. J. of Robust and Nonlinear Control*, (24):699–724, 2014.

[6] Cinzia Bernardeschi, Luca Cassano, Andrea Domenici, and Paolo Masci. Debugging PVS specifications of control logics via event-driven simulation. In *Computation Tools 2010*, Lisbon, Portugal, November 21–26 2010. IARIA.

[7] Cinzia Bernardeschi, Luca Cassano, Andrea Domenici, and Paolo Masci. Simulation and Test-Case Generation for PVS Specifications of Control Logics. *International Journal on Advances in Software*, 4(3 & 4):327–341, 2011.

[8] Cinzia Bernardeschi, Andrea Domenici, and Paolo Masci. Integrated simulation of implantable cardiac pacemaker software and heart models. In *CARDIOTECHNIX 2014, 2d International Congress on Cardiovascular Technology*, pages 55–59. SCITEPRESS, 2014.

[9] Cinzia Bernardeschi, Andrea Domenici, and Paolo Masci. Towards a formalization of system requirements for an integrated clinical environment. In *5th EAI International Conference on Wireless Mobile Communication and Healthcare (MOBIHEALTH 2015)*. Springer, 2015. In press.

[10] Simon Bliudze and Daniel Krob. Modelling of complex systems: Systems as dataflow machines. *Fundam. Inf.*, 91(2):251–274, April 2009.

[11] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*. Springer-Verlag New York, 2010.

[12] Victor Carreño and César Muñoz. Aircraft trajectory modeling and alerting algorithm verification. In Mark Aagaard and John Harrison, editors, *Theorem Proving in Higher Order Logics*, volume 1869 of *Lecture Notes in Computer Science*, pages 90–105. Springer Berlin Heidelberg, 2000.

[13] Victor Carreño and César Muñoz. Safety verification of the Small Aircraft Transportation System concept of operations. In *Proceedings of the AIAA 5th aviation, technology, integration, and operations conference (AIAA-2005-7423)*, 2005.

[14] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[15] A. Cimatti, S. Mover, and S. Tonetta. A quantifier-free SMT encoding of non-linear hybrid automata. In *Formal Methods in Computer-Aided Design (FMCAD), 2012*, pages 187–195, Oct 2012.

[16] Marc Daumas, David Lester, and César Muñoz. Verified real number calculations: A library for interval arithmetic. *IEEE Transactions on Computers*, 58(2):226–237, February 2009.

[17] Ben Di Vito. Strategy-enhanced interactive proving and arithmetic simplification for PVS. In *Proceedings of the 1st International Workshop on Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003)*, pages 43–55, 2003.

[18] Ben Di Vito. Manip User's Guide, Version 1.3, 2011.

[19] Bruno Dutertre. Elements of mathematical analysis in PVS. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Joakim von Wright, Jim Grundy, and John Harrison, editors, *Theorem Proving in Higher Order Logics*, volume 1125 of *Lecture Notes in Computer Science*, pages 141–156. Springer Berlin Heidelberg, 1996.

[20] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley, 2014.

[21] Sébastien Furic. Enforcing Reliability of Discrete-Time Models in Modelica. In *Proceedings of the 8th International Modelica Conference*, pages 638–649, 2011.

[22] Gerhard Karl Erich Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift*, 39(2):176–210, 1934.

[23] Gerhard Karl Erich Gentzen. Untersuchungen über das logische Schließen. II. *Mathematische Zeitschrift*, 39(3):176–210, 1934.

[24] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.

[25] Hanne Gottliebsen. Transcendental Functions and Continuity Checking in PVS. In Mark Aagaard and John Harrison, editors, *Theorem Proving in Higher Order Logics*, volume 1869 of *Lecture Notes in Computer Science*, pages 197–214. Springer Berlin Heidelberg, 2000.

[26] Paolo Masci, Yi Zhang, Paul Jones, Patrick Oladimeji, Enrico D'Urso, Cinzia Bernardeschi, Paul Curzon, and Harold Thimbleby. Combining PVSio with stateflow. In *Proceedings of the 6th NASA Formal Methods Symposium (NFM2014)*, Berlin, Heidelberg, April–May 2014. Springer-Verlag.

[27] César Muñoz, Anthony Narkawicz, George Hagen, Jason Upchurch, Aaron Dutle, and María Consiglio. DAIDALUS: Detect and Avoid Alerting Logic for Unmanned Systems. In *Proceedings of the 34th Digital Avionics Systems Conference (DASC 2015)*, Prague, Czech Republic, September 2015.

[28] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.

[29] Patrick Oladimeji, Paolo Masci, Paul Curzon, and Harold Thimbleby. PVSio-web: a tool for rapid prototyping device user interfaces in PVS. In *FMIS2013, 5th International Workshop on Formal Methods for Interactive Systems, London, UK, June 24, 2013*, 2013.

[30] S. Owre, S. Rajan, J. Rushby, N. Shankar, and M. Srivas. PVS: combining specification, proof checking, and model checking. In R. Alur and T.A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in LNCS, pages 411–414. Springer-Verlag, 1996.

[31] S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Automated Deduction — CADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer Berlin Heidelberg, 1992.

[32] S. Owre and N. Shankar. The PVS Prelude Library. Technical report, SRI International Computer Science Laboratory, 333 Ravenswood Avenue, Menlo Park CA 94025, USA, 2001.

[33] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. PVS Language Reference, Version 2.4. Technical report, SRI International Computer Science Laboratory, 333 Ravenswood Avenue, Menlo Park CA 94025, USA, 2001.

[34] Christine Paulin-Mohring. Introduction to the Coq Proof-Assistant for Practical Software Verification. In Bertrand Meyer and Martin Nordio, editors, *Tools for Practical Software Verification*, volume 7682 of *Lecture Notes in Computer Science*, pages 45–95. Springer Berlin Heidelberg, 2012.

[35] André Platzer. Differential dynamic logics. *KI*, 24(1):75–77, 2010.

[36] André Platzer and Jan-David Quesel. KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning*, volume 5195 of *Lecture Notes in Computer Science*, pages 171–178. Springer Berlin Heidelberg, 2008.

[37] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Trans. Embed. Comput. Syst.*, 6(1), February 2007.

[38] Stefan Schupp, Erika Ábrahám, Xin Chen, Ibtissem Ben Makhlouf, Goran Frehse, Sriram Sankaranarayanan, and Stefan Kowalewski. Current challenges in the verification of hybrid systems. In Mohammad Reza Mousavi and Christian Berger, editors, *Cyber Physical Systems. Design, Modeling, and Evaluation*, volume 9361 of *Lecture Notes in Computer Science*, pages 8–24. Springer International Publishing, 2015.

[39] N. Shankar, S. Owre, J. M. Rushby, and D. W. J. Stringer-Calvert. PVS Prover Guide, Version 2.4. Technical report, SRI International Computer Science Laboratory, 333 Ravenswood Avenue, Menlo Park CA 94025, USA, 2001.

[40] Mandayam Srivas, Harald Rueß, and David Cyrluk. Hardware verification using PVS. In Thomas Kropf, editor, *Formal Hardware Verification: Methods and Systems in Comparison*, volume 1287 of *Lecture Notes in Computer Science*, pages 156–205. Springer-Verlag, 1997.

[41] Ashish Tiwari. HybridSAL Relational Abstracter. In *Proceedings of the 24th International Conference on Computer Aided Verification*, CAV'12, pages 725–731, Berlin, Heidelberg, 2012. Springer-Verlag.

[42] Ashish Tiwari and Gaurav Khanna. Series of abstractions for hybrid automata. In *Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control*, HSCC '02, pages 465–478, London, UK, UK, 2002. Springer-Verlag.

[43] Ashish Tiwari and Gaurav Khanna. Nonlinear systems: Approximating reach sets. In Rajeev Alur and GeorgeJ. Pappas, editors, *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 600–614. Springer Berlin Heidelberg, 2004.

[44] D.A. Van Beek, Michel Reniers, J.E. Rooda, and Ramon R.H. Schiffelers. Concrete syntax and semantics of the compositional interchange format for hybrid systems. In *IFAC World Congress (IFAC 2008)*, pages 7979–7986, Chung, Myung Jin and Misra, Pradeep, 2008.

[45] Alain Vande Wouwer, Philippe Saucez, and Carlos Vilas Fernández. *Simulation of ODE/PDE Models with MATLAB, OCTAVE and SCILAB.* Springer International Publishing, 2014.