Logic and Compositional Verification of Hybrid Systems*
(Invited Tutorial)

André Platzer

Carnegie Mellon University, Computer Science Department,
Pittsburgh, PA, USA
aplatzer@cs.cmu.edu

Abstract. Hybrid systems are models for complex physical systems and have become a widely used concept for understanding their behavior. Many applications are safety-critical, including car, railway, and air traffic control, robotics, physical-chemical process control, and biomedical devices. Hybrid systems analysis studies how we can build computerised controllers for physical systems which are guaranteed to meet their design goals. The continuous dynamics of hybrid systems can be modeled by differential equations, the discrete dynamics by a combination of discrete state-transitions and conditional execution. The discrete and continuous dynamics interact to form hybrid systems, which makes them quite challenging for verification.

In this tutorial, we survey state-of-the-art verification techniques for hybrid systems. In particular, we focus on a coherent logical approach for systematic hybrid systems analysis. We survey theory, practice, and applications, and show how hybrid systems can be verified in the hybrid systems verification tool KeYmaera. KeYmaera has been used successfully to verify safety, reactivity, controllability, and liveness properties, including collision freedom in air traffic, car, and railway control systems. It has also been used to verify properties of electrical circuits.

1 Introduction

Hybrid systems are a common model for systems where both discrete and continuous behavior are important [2, 8, 10, 20]. Hybrid systems arise, for instance, when a computer controls a physical process. Then the computer will cause discrete transitions and digital switching at various discrete points in time, while the physical process keeps evolving continuously.

As a common mathematical model for such complex physical systems, *hybrid systems* are dynamical systems [28] where the system state evolves over time

* This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181, and under Grant No. CNS-0931985, and by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

G. Gopalakrishnan and S. Qadeer (Eds.): CAV 2011, LNCS 6806, pp. 28–43, 2011.
© Springer-Verlag Berlin Heidelberg 2011

according to interacting laws of discrete and continuous dynamics [1, 9, 10, 16, 20, 43].

One canonical example is a train driving on a railway track. The train moves continuously on the track while its behavior is controlled by several computer control systems supporting the train conductor even up to full automation. One of the most crucial safety-critical correctness properties of a train is that we want to ensure that the train controller prevents all train collisions. A study of the correctness of the train cannot be split into an isolated study of the software and an isolated study of the mechanical parts. They work together and need to be verified together. We cannot determine whether a software controller for a part of a train is correct unless we understand enough of the physics of the train that it controls. We cannot fully understand how a train moves physically without understanding how its digital controllers, control programs, sensors, and actuators affect its behavior. We need to look at both, i.e., the hybrid system dynamics, to find out.

Hybrid systems are equally important in the automotive, aviation, railway, and robotics industry for instance. They occur in factory automation problems and biological, chemical, and physical process control. Most of these applications are safety-critical, because badly controlled processes can have a huge impact on the system environment, especially when the processes operate close to humans. Hybrid systems verification is a very challenging but important problem for which a range of techniques have been developed [10, 12–14, 16, 17, 20–22, 24–26, 41, 42].

In this tutorial, we survey a number of state-of-the-art verification techniques for hybrid systems, especially a logical approach for hybrid systems analysis [30–32]. This approach forms the basis for the differential invariants as fixed points procedure [37] that computes the invariants and differential invariants required for verification in a fixed point loop. This logic-based verification approach has been implemented in the verification tool KeYmaera[1] for hybrid systems [39]. KeYmaera has been used successfully to verify several safety-critical properties, including collision freedom, of the cooperation protocol of the European Train Control System [40] and of aircraft roundabout maneuvers [31] and the flyable aircraft roundabout maneuver [38]. More details about the hybrid systems verification techniques surveyed in this tutorial can be found in the book *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*[2] [32].

The approach presented in this tutorial and the verification tool KeYmaera is also very instructive for teaching hybrid systems verification and the use of logic and formal methods for complex physical systems. The sophisticated graphical user interface of KeYmaera makes it easier to work with the system and learn how hybrid systems verification works. It also makes it easier to understand proofs that KeYmaera found automatically. KeYmaera's interaction capabilities, which are based on those of KeY [4], also help solving very complex verification questions and system design questions interactively that are still beyond the capabilities of today's automation techniques. The author has taught two graduate

[1] `http://symbolaris.com/info/KeYmaera.html`
[2] `http://symbolaris.com/lahs/`

courses on hybrid systems verification using the approach presented here. Course material is available at the web page[2] of the book [32].

Even though we do not focus on these extensions in this tutorial, the approach taken in this tutorial can be extended to logic and verification techniques for distributed hybrid systems [33, 34], i.e., systems that are both distributed systems and hybrid systems. These distributed hybrid systems include multi-agent hybrid systems, reconfigurable hybrid systems, and hybrid systems with an evolving and unbounded number of agents. The approach also extends to logic and verification techniques for stochastic hybrid systems [35].

2 Hybrid Systems

There is a range of models for hybrid systems [2, 5–7, 10, 15, 20, 27, 30, 31, 43]. We focus on hybrid programs [30, 32], and the related model of hybrid automata [2, 20].

Hybrid system models allow the user to specify the continuous dynamics by differential equations. Continuous dynamics results, e.g., from the continuous movement of a train along the track (train position z evolves with velocity v along the differential equation $z' = v$ where z' is the time-derivative of z) or from the continuous variation of its velocity over time ($v' = a$ with acceleration a). Other behavior can be modelled more naturally by discrete dynamics, for example, the instantaneous change of control variables like the acceleration (e.g., the changing of a by setting $a := -b$ with braking force $b > 0$) or change of status information in discrete controllers. Both kinds of dynamics interact, e.g., when measurements of the continuous state affect decisions of discrete controllers (the train switches to braking mode when velocity v is too high). Likewise, they interact when the resulting control choices take effect by changing the control variables of the continuous dynamics (e.g., changing the acceleration control variable a in $z'' = a$). The combination of continuous dynamics with analog or discrete control causes complex system behavior, which can neither be verified by purely continuous reasoning (because of the discontinuities caused by discrete transitions) nor by considering discrete change in isolation (because safety depends on continuous states).

2.1 Undecidability of Numerical Image Computation

Verification of hybrid systems is a very challenging problem. The verification problem is the problem to decide whether a given hybrid system satisfies a given correctness property (e.g., safety, liveness, and so on). Unfortunately, this problem is undecidable even for very simple hybrid systems [11, 20].

Even for absurdly limited models of hybrid systems, the verification problem is neither semidecidable nor co-semidecidable numerically, even for a bounded number of transitions and when tolerating arbitrarily large error bounds in the decision [36]. The numerical image computation problem plays a role that is almost as central as that of the halting problem for Turing machines. We refer to the literature [36] for a formal statement and proof. The basic intuition behind

the undecidability result for the numerical image computation problem is shown in Fig. 1. Suppose an algorithm could decide safety of a system numerically by evaluating the value of the system flow φ at points. If the algorithm is a decision algorithm, it would have to terminate in finite time, hence, after evaluating a finite number of points, say x_1, x_2, x_3 in Fig. 1. But from the information that the algorithm has gathered at a finite number of points, it cannot distinguish the good behavior φ (solid flow safely outside B) from the bad behavior g (dashed flow reaching bad region B). The same undecidability result still holds even when restricting the flow φ to very special classes of functions and when assuming that its derivatives could be evaluated and even when tolerating arbitrarily large error bounds in the decision [36]. There is a series of extra assumptions and bounds that make the problem (approximately) decidable again by imposing extra constraints on the system; see [36]. Yet, by the general undecidability result, these extra bounds (and several other bounds that have been proposed in related work) cannot be computed numerically. Because of this strong numerical undecidability result, it is surprisingly difficult but not impossible to get hybrid systems verification techniques sound [17, 41].

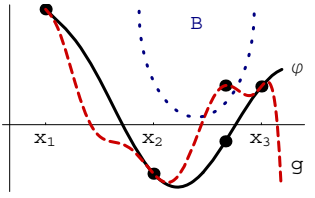


Fig. 1. Indistinguishable

Consequently, sound verification of hybrid systems needs some symbolic part. In the remainder of this tutorial, we focus on a purely symbolic and logical approach that is formally sound, i.e., the verification result is always correct.

2.2 Hybrid Programs

Hybrid programs are program models for hybrid systems and are formed using the statements and operations in Table 1.

Discrete jump sets. Discrete transitions are represented as instantaneous assignments of values to state variables. They can express resets like $a := -b$ or adjustments of control variables like $a := A$. To handle simultaneous changes of multiple variables, discrete jumps can be combined to sets of jumps with simultaneous effect. For instance, the discrete jump set $a := a + 5, A := 2a^2$ expresses that a is increased by 5 and, simultaneously, variable A is set to $2a^2$, which is evaluated *before* a receives its new value $a + 5$.

Table 1. Statements and effects of hybrid programs (HPs)

HP Notation	Operation	Effect
$x_1 := \theta_1, \ldots, x_n := \theta_n$	discrete jump	simultaneously assign θ_i to variables x_i
$x_1' = \theta_1, x_2' = \theta_2, \ldots$ $\ldots, x_n' = \theta_n \,\&\, H$	continuous evo.	differential equations for x_i within evolution domain H (first-order formula)
$?H$	state test	test first-order formula H at current state
$\alpha; \beta$	seq. composition	HP β starts after HP α finishes
$\alpha \cup \beta$	nondet. choice	choice between alternatives HP α or HP β
α^*	nondet. repetition	repeats HP α n-times for any $n \in \mathbb{N}$

Differential equation systems. Continuous evolution in the system dynamics is represented using differential equation systems as evolution constraints. For example the (second-order) differential equation $z'' = -b$ describes deceleration with braking force b and $z' = v, v' = -b \,\&\, v \geq 0$ expresses that the evolution only applies as long as the speed is $v \geq 0$. This is an evolution along the differential equation system $z' = v, v' = -b$ that is restricted (written &) to remain within the evolution domain region $v \geq 0$, i.e., to stop braking before $v < 0$. Such an evolution can stop at any time within $v \geq 0$, it could even continue with transient grazing along the border $v = 0$, but it is never allowed to enter $v < 0$. The second-order differential equation $z'' = -b$ itself is equivalent to the first-order differential equation system $z' = v, v' = -b$, in which the velocity v is explicit.

Control structure. Discrete and continuous transitions—represented as jump sets or differential equations, respectively—can be combined to form a hybrid program with interacting hybrid dynamics using regular expression operators $(\cup, ^*, ;)$ of regular programs [19] as control structure. For example, the hybrid program $q := accel \cup z'' = -b$ describes a train controller that can choose to either switch to acceleration mode ($q := accel$) or brake by the differential equation $z'' = -b$, by a nondeterministic choice ($\cup$). The nondeterministic choice $q := accel \cup z'' = -b$ expresses that either $q := accel$ or $z'' = -b$ happens, nondeterministically. The system can choose one of the two options. The sequential composition $a := -b\,;\, z'' = a$, instead, expresses that first, the acceleration a is updated by $a := -b$, and then the system follows the differential equation $z'' = a$ with the updated acceleration (hence brakes). In conjunction with other regular combinations, control constraints can be expressed using tests like $?z \geq SB$ as guards for the system state. This test will succeed if, indeed, the current state of the system satisfies $z \geq SB$; otherwise the test will fail and execution cannot proceed. In that respect, a test is like an assert statement in conventional programs and cuts the system run if the test is not successful.

Other control structures can easily be defined from the basic operations in Table 1. See Table 2 for a list of common additional statements that can be defined [32] from those in Table 1. For instance,

$$\text{if } H \text{ then } \alpha \text{ else } \beta \equiv (?H;\alpha) \cup (?\neg H;\beta)$$

Example 1 (Natural hybrid program for simple train). As a much simplified example of a train controller, consider the following hybrid program:

$$\big(((?z < SB;\ a := A) \cup (a := -b));\ \ z' = v, v' = a \,\&\, v \geq 0\big)^* \tag{1}$$

First, the discrete controller executes and then, after the sequential composition (;), the train follows the differential equation system $z' = v, v' = a$ that is restricted to (written &) the evolution domain $v \geq 0$. The discrete controller consists of a nondeterministic choice ($\cup$) between two options. The left option

Table 2. Additional statements and control structures definable as abbreviations

HP Notation	Operation	Effect
$x := *$	nondet. assign.	assigns any real value to x
if H then α else β	if-then-else	executes HP α if H holds, otherwise HP β
if H then α	if-then	executes HP α if H holds, otherwise no effect
while H do α	while loop	repeats α if H holds, stops if $\neg H$ holds at end
repeat α until H	repeat until	repeats α (at least once) until H holds at end
skip	do nothing	no effect and does not change the state space
abort	aborts run	blocks current run and allows no transition

performs a test ($?z < SB$) to check whether the current position is left of the start braking point SB and then, after the left-most sequential composition (;), assigns the positive acceleration A to a by $a := A$. The right option does not perform a test, but just assigns the braking force $-b$ to the acceleration a by $a := -b$. In particular, the first control option (acceleration) is only available when the train has not yet passed the start braking point SB, while the second control option (braking) is always available. The train would choose between both options nondeterministically when both are possible. Otherwise, it can only choose the options that successfully pass their respective tests (the right option in (1) is always available because it has not tests). Finally, the repetition operator (*) at the end of hybrid program (1) expresses that the controller-plant-loop can repeat indefinitely. This pattern $(\mathit{ctrl}; \mathit{plant})^*$ is a very common use case for hybrid programs, but by far not the only useful form of a system model.

The effect of the discrete jump set $x_1 := \theta_1, \ldots, x_n := \theta_n$ is to simultaneously change the interpretations of the x_i to the respective θ_i by a discrete jump in the state space. The new values θ_i are evaluated before changing the value of any variable x_j. The effect of $x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, H$ is an ongoing continuous evolution respecting the differential equation system $x_1' = \theta_1, \ldots, x_n' = \theta_n$ that is restricted to remain within the evolution domain region H. The evolution is allowed to stop at any point in H. It is, however, required to stop before it leaves H. For unconstrained evolutions, we write $x' = \theta$ in place of $x' = \theta \,\&\, \mathit{true}$.

The test action or state check $?H$ is used to define conditions. Its semantics is that of a no-op if the formula H is true in the current state; otherwise, like *abort*, it allows no transitions. That is, if the test succeeds because formula H holds in the current state, then the state does not change, and the system execution continues normally. If the test fails because formula H does not hold in the current state, then the system execution cannot even continue. Thus, the effect of a test action is similar to an *assert* statement in Java.

The nondeterministic choice $\alpha \cup \beta$, sequential composition $\alpha; \beta$, and nondeterministic repetition α^* of programs are as in regular expressions but generalised to a semantics in hybrid systems. Choices $\alpha \cup \beta$ are used to express behavioral alternatives between the transitions of α and β. That is, the hybrid program $\alpha \cup \beta$ can choose nondeterministically to follow the transitions of the hybrid program α, or, instead, to follow the transitions of the hybrid program β. The sequential composition $\alpha; \beta$ says that the hybrid program β starts executing after α has

finished (β never starts if α does not terminate). In $\alpha;\beta$, the transitions of α take effect first, until α terminates (if it does), and then β continues. Repetition α^* is used to express that the hybrid process α repeats any number of times, including zero times. When following α^*, the transitions of hybrid program α can be repeated over and over again, any nondeterministic number of times (≥ 0). Hybrid programs form a regular-expression-style Kleene algebra with tests [23].

The formal transition semantics of hybrid programs is defined in [30, 32].

2.3 Hybrid Automata

Hybrid automata are an automaton representation of hybrid systems [2, 20]. The basic idea is to have one differential equation per mode of continuous evolution of the system with an automaton structure on top that defines how and under which condition the system switches between the various modes.

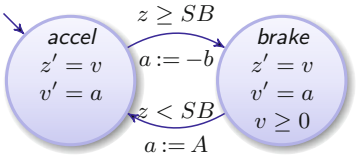
A *hybrid automaton* is a finite directed graph with a set of nodes V and a set of edges E, where

- $x_1, \ldots, x_n$ are the continuous state variables and n is the (fixed) dimension of the continuous state space.
- Each node $v \in V$ is labeled with a differential equation $x_1' = \theta_1, .., x_n' = \theta_n$ and an evolution domain constraint H, which is a quantifier-free formula of real arithmetic. The differential equation specifies how the variables $x_1, \ldots, x_n$ evolve while the system is in node v and the evolution domain constraint H has to be true all the time while in mode v.
- Each edge $e \in E$ is labeled with a guard H, which is a quantifier-free formula of real arithmetic, and a discrete jump set $x_1 := \theta_1, \ldots, x_n := \theta_n$. The guard H determines when edge e can be taken. The discrete jump set (called *reset*) $x_1 := \theta_1, \ldots, x_n := \theta_n$ determines how the variables are reassigned when the system follows edge e.

Each of the transitions of a hybrid automaton is either a discrete or a continuous transition. A continuous transition within one node is a continuous evolution along the differential equation of that node without leaving the evolution domain constraint. A discrete transition along an edge is possible if the guard H is satisfied in the current state and then the state will be reset according to the discrete jump set $x_1 := \theta_1, \ldots, x_n := \theta_n$ when following the edge. The hybrid automaton itself repeats discrete and continuous transitions indefinitely. See, e.g., [20, 32], for a formal definition of the transitions of a hybrid automaton.

We examine the relationship between hybrid programs and hybrid automata in the following example where we consider hybrid automaton and hybrid program side by side.

Example 2 (Hybrid automata versus hybrid programs). With the operations in Table 1, hybrid systems can be represented naturally as hybrid programs. For example, the right of Fig. 2 depicts a hybrid program of an (overly) simplified train control. The hybrid automaton on the left of Fig. 2 shows a corresponding hybrid automaton. Line 1 represents that, in the beginning, the current node q of the



1 $q := accel;$ /* *initial mode is node accel* */
2 $(\ (?q = accel;\ \ z' = v, v' = a)$
3 $\cup\ (?q = accel \wedge z \geq SB;\ a := -b;\ q := brake;\ ?v \geq 0)$
4 $\cup\ (?q = brake;\ \ z' = v, v' = a\ \&\ v \geq 0)$
5 $\cup\ (?q = brake \wedge z < SB;\ \ a := A;\ \ q := accel))^*$

Fig. 2. Hybrid automaton and hybrid program for a much simplified train control

system is the initial node *accel*. We represent each discrete and continuous transition of the automaton as a sequence of statements with a nondeterministic choice ($\cup$) between these transitions. Line 4 represents a continuous transition of the automaton. It tests if the current node q is *brake*, and then (i.e., if the test was successful) follows the differential equation system $z' = v, v' = a$ restricted to the evolution domain $v \geq 0$. Line 3 characterises a discrete transition of the automaton. It tests the guard $z \geq SB$ when in node *accel*, and, if successful, resets $a := -b$ and then switches q to node *brake*. By the semantics of hybrid automata [1, 20], an automaton in node *accel* is only allowed to make a transition to node *brake* if the evolution domain restriction of *brake* is true when entering the node, which is expressed by the additional test $?v \geq 0$ at the end of line 3. Observe that this test of the evolution domain region generally needs to be checked as the last operation after the guard and reset, because a reset like $v := v - 1$ could affect the outcome of the evolution domain region test. In order to obtain a fully compositional model, hybrid programs make all these implicit side conditions explicit. Line 2 represents the continuous transition when staying in node *accel* and following the differential equation system $z' = v, v' = a$. Line 5 represents the discrete transition from node *brake* of the automaton to node *accel*.

Lines 2–5 cannot be executed unless their tests succeed. In particular, at any state, the nondeterministic choice ($\cup$) among lines 2–5 reduces de facto to a nondeterministic choice between either lines 2–3 or between lines 4–5. At any state, q can have value either *accel* or *brake* (assuming these are different constants), not both. Consequently, when $q = brake$, a nondeterministic choice of lines 2–3 would immediately fail the tests in the beginning and not execute any further. The only remaining choices that have a chance to succeed are lines 4–5 then. In fact, only the single successful choice of line 4 would remain if the second conjunct $z < SB$ of the test in line 5 does not hold for the current state. Note that, still, all four choices in lines 2–5 are available, but at least two of these nondeterministic choices will always be unsuccessful. Finally, the repetition operator (*) at the end of Fig. 2 expresses that the transitions of a hybrid automaton, as represented by lines 2–5, can repeat indefinitely, possibly taking different nondeterministic choices between lines 2–5 at every repetition.

The hybrid program on the right of Fig. 2 directly corresponds to the hybrid automaton on the left of Fig. 2. This translation is simple and systematic. The same translation principle works for all hybrid automata and can represent them faithfully as hybrid programs [32], just like finite automata can be implemented in a conventional while-programming language. This direct translation, however,

blows up the representation. A much more natural hybrid program can usually be found when directly representing the hybrid system as a hybrid program right away. More natural representations also have computational advantages for verification. This is the preferred way for designing systems.

Example 3 (Natural hybrid program corresponding to Fig. 2). The natural hybrid program corresponding to the system in Fig. 2 is the following hybrid program:

$$\big((\texttt{if}(z \geq SB)\, a := -b\, \texttt{else}\, a := A);\quad z' = v, v' = a \,\&\, v \geq 0\big)^* \tag{2}$$

This hybrid program is almost identical to that in (1), except that it has an extra test specifying that the braking option can only be chosen if the position z is after the start braking point SB. Contrast the natural hybrid program in (2) with the hybrid program on the right of Fig. 2 that has been constructed from a hybrid automaton. The natural hybrid program has the same behavior as the hybrid automaton and its corresponding hybrid program, but the natural hybrid program in (2) is significantly easier to understand and also simplifies verification. Finally, the natural hybrid program in (2) is the same as the following hybrid program when resolving abbreviations according to Table 2.

$$\big(((?z \geq SB;\ a := -b) \cup (?z < SB;\ a := A));\quad z' = v, v' = a \,\&\, v \geq 0\big)^*$$

This representational flexibility gives hybrid programs an edge over hybrid automata. The same system can be represented in many ways and a representation that is most natural to a problem often makes the verification easier. It should be noted that there is more than one hybrid automaton describing the same hybrid system, too. Nevertheless, the representation of hybrid systems as hybrid programs is more flexible, because discrete, continuous, and switching dynamics are not restricted to a specific pattern, but can be combined freely using regular expression style operators.

3 Logic for Hybrid Systems

Hybrid programs are a flexible behavioral model for hybrid systems. As a specification and verification language for hybrid systems, we have introduced the *differential dynamic logic* $\mathsf{d}\mathcal{L}$ [29, 30, 32]. In $\mathsf{d}\mathcal{L}$, operational models of hybrid systems are internalized as first-class citizens, so that correctness statements about the transition behavior of hybrid systems can be expressed as formulas. That is, correctness statements about systems can be combined into bigger formulas with arbitrary propositional operators or quantifiers, and even into nestings of formulas. As a basis, $\mathsf{d}\mathcal{L}$ includes (nonlinear) real arithmetic for describing concepts like safe regions of the state space. Further, $\mathsf{d}\mathcal{L}$ supports real-valued quantifiers for quantifying over the possible values of system parameters or durations of continuous evolutions. For talking about the transition behavior of hybrid systems, $\mathsf{d}\mathcal{L}$ provides modal operators such as $[\alpha]$ or $\langle\alpha\rangle$ that refer to the states reachable by following the transitions of hybrid program α. The logical operators of $\mathsf{d}\mathcal{L}$ are summarized in Table 3.

Table 3. Operators of differential dynamic logic for hybrid systems ($\mathsf{d}\mathcal{L}$)

Notation	Operator	Meaning
$\theta_1 = \theta_2$	equality	value of θ_1 is equal to that of θ_2
$\theta_1 \geq \theta_2$	comparison	value of θ_1 is greater or equal that of θ_2
$\theta_1 > \theta_2$	comparison	value of θ_1 is greater than that of θ_2
$\theta_1 \leq \theta_2$	comparison	value of θ_1 is less or equal that of θ_2
$\theta_1 < \theta_2$	comparison	value of θ_1 is less than that of θ_2
$\neg\phi$	negation/not	true if ϕ is false
$\phi \wedge \psi$	conjunction/and	true if both ϕ and ψ are true
$\phi \vee \psi$	disjunction/or	true if ϕ is true or if ψ is true
$\phi \rightarrow \psi$	implication	true if ϕ is false or ψ is true
$\phi \leftrightarrow \psi$	equivalence	true if ϕ and ψ are both true or both false
$\forall x\, \phi$	for all quantifier	true if ϕ is true for all values of variable x
$\exists x\, \phi$	exists quantifier	true if ϕ is true for some values of variable x
$[\alpha]\phi$	$[\cdot]$ modality	true if ϕ true after all runs of HP α
$\langle\alpha\rangle\phi$	$\langle\cdot\rangle$ modality	true if ϕ true after at least one run of HP α

Within a single specification and verification language, $\mathsf{d}\mathcal{L}$ combines operational system models with means to talk about the states that are reachable by system transitions. The logic $\mathsf{d}\mathcal{L}$ provides parametrized modal operators $[\alpha]$ and $\langle\alpha\rangle$ that refer to the states reachable by hybrid program α and can be placed in front of any formula. The formula $[\alpha]\phi$ expresses that all states reachable by hybrid program α satisfy formula ϕ. Likewise, $\langle\alpha\rangle\phi$ expresses that there is at least one state reachable by α for which ϕ holds. These modalities can be used to express necessary or possible properties of the transition behavior of α in a natural way. They can be nested or combined propositionally. The $\mathsf{d}\mathcal{L}$ logic supports quantifiers like $\exists p\, [\alpha]\langle\beta\rangle\phi$ which says that there is a choice of parameter p (expressed by $\exists p$) such that for all possible behaviors of hybrid program α (expressed by $[\alpha]$) there is a reaction of hybrid program β (i.e., $\langle\beta\rangle$) that ensures ϕ. Likewise, $\exists p\, ([\alpha]\phi \wedge [\beta]\psi)$ says that there is a choice of parameter p that makes both $[\alpha]\phi$ and $[\beta]\psi$ true, simultaneously, i.e., that makes the conjunction $[\alpha]\phi \wedge [\beta]\psi$ true, saying that formula ϕ holds for all states reachable by α executions and, independently, ψ holds after all β executions. This gives a flexible logic for specifying and verifying even sophisticated properties of hybrid systems, including the ability to refer to multiple hybrid systems at once.

The semantics of differential dynamic logic and more details about it can be found in [30, 32].

Example 4 (Safety in train control). Let *train* denote the hybrid program for the simple train control dynamics in (2). Consider the following $\mathsf{d}\mathcal{L}$ formula

$$v \geq 0 \wedge z < m \;\rightarrow\; [\mathit{train}]z < m \tag{3}$$

It expresses that, when the system starts in an initial state where $v \geq 0 \wedge z < m$ is true, i.e., with nonnegative velocity and with a train position z within the movement authority limits m, then, when following the dynamics of the hybrid program *train*, then the system will always be a in a state where $z < m$ is true.

It turns out that formula (3) is a bit naive and needs additional assumptions on the parameters to be valid. For instance, the train will not remain safe, even if it starts safely within $z < m$ if its initial velocity is so high that it cannot brake in time before leaving $z < m$. Similarly, the start braking point parameter SB in (2) needs to be chosen carefully to ensure that (3) is valid. But under corresponding additional constraints, the following $\mathsf{d}\mathcal{L}$ formula can be proven to be valid, i.e., true under all interpretations for all the variables and parameters:

$$v^2 < 2b(m-z) \wedge b > 0 \wedge A \geq 0 \;\rightarrow$$
$$\big[\big(SB := m - \frac{v^2}{2b} - (\frac{A}{b}+1)(\frac{A}{2}\varepsilon^2 + \varepsilon v);\;\; \texttt{if}(z \geq SB)\, a := -b \,\texttt{else}\, a := A;$$
$$t := 0;\; z' = v, v' = a, t' = 1 \,\&\, v \geq 0 \wedge t \leq \varepsilon\big)^*\big]\,(z < m) \quad (4)$$

Variable t is a clock that evolves by $t' = 1$. The bound $t \leq \varepsilon$ gives an upper bound on the time of the continuous evolution until the discrete controllers have a chance to react to situation changes again. See [32, 40] for details.

4 Compositional Deductive Verification

The verification problem for hybrid systems is a very challenging problem. It is not even semidecidable numerically [36]. In the fully symbolic domain of differential dynamic logic, however, we can do better. There is a sound compositional proof system that works fully symbolically [30, 32]. It can be used to prove interesting properties of hybrid programs including safety, reactivity, controllability, and liveness. The fact that this proof system is compositional is also important for scalability purposes. Because it proves properties of complex hybrid systems by reducing them to properties about simpler systems, this compositional verification approach can scale to complex systems.

Furthermore, the proof system is a *complete axiomatization of hybrid systems relative to differential equations* [30]. That is, every true statement about a hybrid system can be proven from elementary properties of differential equations.

Theorem 1 (Relative completeness [30]). *Hybrid systems can be axiomatized completely relative to differential equations.*

The proof of this theorem is a tricky 15page proof, but the theorem has important consequences. It proves that all true properties of hybrid systems can be decomposed successfully into properties of their parts. This theorem also explains the practical verification successes with this approach in air traffic [38] and railway control [40] and shows that other systems can be verified with the approach. The reason is that the decomposition of the entire verification problem into elementary properties of the dynamical aspects makes the verification problem tractable.

Theorem 1 has another important consequence. It gives a formal reason why the handling of differential equations is at the heart of hybrid systems

verification. Moreover, the proof calculus in [30, 32] completely lifts every verification technique for differential equations to a verification technique for hybrid systems. In general, it may not always be clear how verification techniques for continuous systems generalize to hybrid systems. But Theorem 1 gives a formal proof showing *that* and *how* to generalize any verification technique for differential equations to full hybrid systems completely.

A prime example of such advanced and powerful verification techniques are *differential invariants* for differential equations [31]. Differential invariants have been instrumental in enabling the verification of complex hybrid systems, including air traffic control [38], train control with disturbances in the dynamics [40], and electrical circuits [32]. Differential invariants turn the following intuition into a formally sound proof procedure. If the vector field of the differential equation always points into a direction where the differential invariant F, which is a logical formula, is becoming "more true" (see Fig. 3), then the system will always stay save if it initially starts save. This principle can be understood in a simple but formally sound way using the logic $\mathsf{d}\mathcal{L}$ [31, 32]. Differential invariants have been introduced in [31] and later refined to a procedure that computes differential invariants in a fixed-point loop [37].

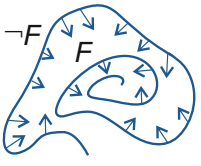


Fig. 3. Differential invariant F

5 Verification Tool KeYmaera

The approach surveyed in this tutorial is implemented in KeYmaera[3], which is a hybrid verification tool for hybrid systems. KeYmaera has a very powerful graphical user interface for conducting proofs and for looking at the proofs that KeYmaera found automatically; see Fig. 4.

This user interface is based on that of the prover KeY [4], from which KeYmaera also inherits its name[4]. KeYmaera has powerful automatic proof procedures that have been used to prove a number of interesting collision avoidance properties in systems including air traffic control and railway control fully automatically [37]. These automation procedures and fixedpoint loops for generating invariants and differential invariants are described in detail in [32, 37].

Nevertheless, the possibility of interacting with KeYmaera can be extremely powerful for verifying complex systems that cannot be handled automatically by any verification tool yet. A good practice for complex physical systems is to combine automatic proof search in KeYmaera with selective user guidance after inspecting the intermediate stage of a partial proof that KeYmaera found in its graphical user interface. KeYmaera also supports annotations such as `@invariant(F)` and `@candidate(F,G)` to annotate problems with possible proof hints about invariant and/or differential invariant formulas `F,G` that could help KeYmaera in finding computationally difficult proofs.

[3] `http://symbolaris.com/info/KeYmaera.html`

[4] KeYmaera is pronounced similar to the hybrid Chimaera from Greek mythology.

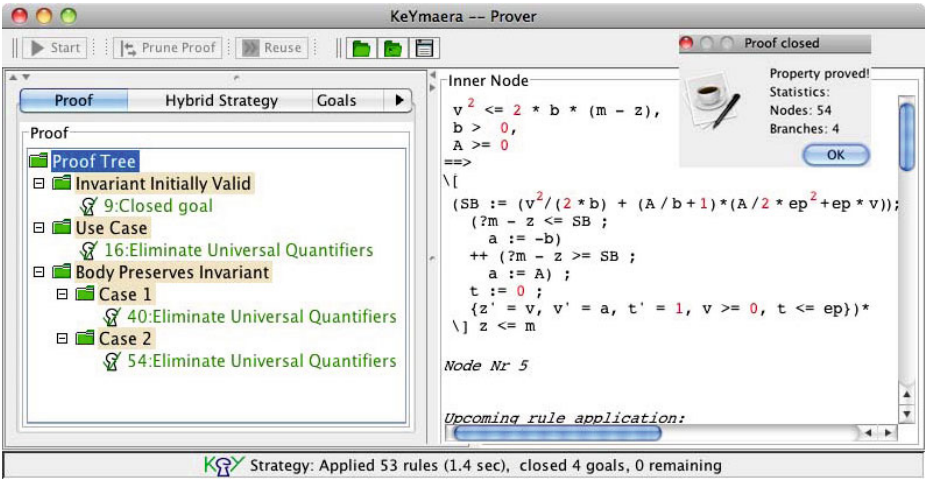


Fig. 4. KeYmaera verification tool for hybrid systems

```
\problem {
  \[ R ep,b,A, SB, a, v, z, t, m; \] (
    v^2 < 2*b*(m-z) & b > 0 & A>=0
  ->
    \[(
    SB := m - (v^2)/(2*b) - ((A/b) + 1) * ((A/2)*ep^2 + ep*v);
    if (z >= SB) then
        a := -b
    else
        a := A
    fi;
    t:=0; {z'=v, v'=a, t'=1, (v >= 0 & t <= ep)}
    )*@invariant(2*b*(m-z)-v^2>0)  // loop @annotation optional
    \] (z < m)
  )
}
```

Fig. 5. KeYmaera problem description for simple train control

The KeYmaera notation for the dℒ formula (4) is shown in Fig. 5. The second line declares the variables ep,b,A, SB, a, v, z, t, m of type real. The annotation @invariant(2*b*(m-z)-v^2>0) gives a proof hint that KeYmaera should use $2b(m-z)-v^2>0$ as a loop invariant. This proof hint is unnecessary, because KeYmaera will automatically discover an invariant that proves the formula in Fig. 5 anyhow.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theor. Comput. Sci. 138(1), 3–34 (1995)
2. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman, et al. (eds.) [18], pp. 209–229
3. Alur, R., Sontag, E.D., Henzinger, T.A. (eds.): HS 1995. LNCS, vol. 1066. Springer, Heidelberg (1996)
4. Beckert, B., Hähnle, R., Schmitt, P.H. (eds.): Verification of Object-Oriented Software. LNCS (LNAI), vol. 4334. Springer, Heidelberg (2007)
5. van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H.: Syntax and consistent equation semantics of hybrid Chi. J. Log. Algebr. Program. 68(1-2), 129–210 (2006)
6. van Beek, D.A., Reniers, M.A., Schiffelers, R.R.H., Rooda, J.E.: Concrete syntax and semantics of the compositional interchange format for hybrid systems. In: 17th IFAC World Congress (2008)
7. Bergstra, J.A., Middelburg, C.A.: Process algebra for hybrid systems. Theor. Comput. Sci. 335(2-3), 215–280 (2005)
8. Branicky, M.S.: General hybrid dynamical systems: Modeling, analysis, and control. In: Alur, et al. (eds.) [3], pp. 186–200
9. Branicky, M.S.: Studies in Hybrid Systems: Modeling, Analysis, and Control. Ph.D. thesis, Dept. Elec. Eng. and Computer Sci. Massachusetts Inst. Technol. Cambridge, MA (1995)
10. Branicky, M.S., Borkar, V.S., Mitter, S.K.: A unified framework for hybrid control: Model and optimal control theory. IEEE T. Automat. Contr. 43(1), 31–45 (1998)
11. Cassez, F., Larsen, K.G.: The impressive power of stopwatches. In: CONCUR, pp. 138–152 (2000)
12. Chaochen, Z., Ji, W., Ravn, A.P.: A formal description of hybrid systems. In: Alur, et al. (eds.) [3], pp. 511–530
13. Chutinan, A., Krogh, B.H.: Computational techniques for hybrid system verification. IEEE T. Automat. Contr. 48(1), 64–75 (2003)
14. Clarke, E.M., Fehnker, A., Han, Z., Krogh, B.H., Ouaknine, J., Stursberg, O., Theobald, M.: Abstraction and counterexample-guided refinement in model checking of hybrid systems. Int. J. Found. Comput. Sci. 14(4), 583–604 (2003)
15. Cuijpers, P.J.L., Reniers, M.A.: Hybrid process algebra. J. Log. Algebr. Program. 62(2), 191–245 (2005)
16. Davoren, J.M., Nerode, A.: Logics for hybrid systems, vol. 88(7), pp. 985–1010. IEEE, Los Alamitos (2000)
17. Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. STTT 10(3), 263–279 (2008)
18. Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.): Hybrid Systems, LNCS, vol. 736. Springer (1993)
19. Harel, D., Kozen, D., Tiuryn, J.: Dynamic logic. MIT Press, Cambridge (2000)
20. Henzinger, T.A.: The theory of hybrid automata. In: LICS, pp. 278–292. IEEE Computer Society, Los Alamitos (1996)

21. Jifeng, H.: From CSP to hybrid systems. In: Roscoe, A.W. (ed.) A classical mind: essays in honour of C. A. R. Hoare, pp. 171–189. Prentice Hall, Hertfordshire (1994)
22. Kesten, Y., Manna, Z., Pnueli, A.: Verification of clocked and hybrid systems. Acta Inf. 36(11), 837–912 (2000)
23. Kozen, D.: Kleene algebra with tests. ACM Trans. Program. Lang. Syst. 19(3), 427–443 (1997)
24. Manna, Z., Sipma, H.: Deductive verification of hybrid systems using STeP. In: Henzinger, T.A., Sastry, S.S. (eds.) HSCC 1998. LNCS, vol. 1386, pp. 305–318. Springer, Heidelberg (1998)
25. Mitchell, I., Bayen, A.M., Tomlin, C.: A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. IEEE T. Automat. Contr. 50(7), 947–957 (2005)
26. Mysore, V., Piazza, C., Mishra, B.: Algorithmic algebraic model checking II: Decidability of semi-algebraic model checking and its applications to systems biology. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 217–233. Springer, Heidelberg (2005)
27. Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: An approach to the description and analysis of hybrid systems. In: Grossman, et al. (eds.) [18], pp. 149–178
28. Perko, L.: Differential equations and dynamical systems. Springer, New York (1991)
29. Platzer, A.: Differential dynamic logic for verifying parametric hybrid systems. In: Olivetti, N. (ed.) TABLEAUX 2007. LNCS (LNAI), vol. 4548, pp. 216–232. Springer, Heidelberg (2007)
30. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. 41(2), 143–189 (2008)
31. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. 20(1), 309–352 (2010)
32. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010)
33. Platzer, A.: Quantified differential dynamic logic for distributed hybrid systems. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 469–483. Springer, Heidelberg (2010)
34. Platzer, A.: Quantified differential invariants. In: Frazzoli, E., Grosu, R. (eds.) HSCC, pp. 63–72. ACM Press, New York (2011)
35. Platzer, A.: Stochastic differential dynamic logic for stochastic hybrid programs. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE. LNCS. Springer, Heidelberg (2011)
36. Platzer, A., Clarke, E.M.: The image computation problem in hybrid systems model checking. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 473–486. Springer, Heidelberg (2007)
37. Platzer, A., Clarke, E.M.: Computing differential invariants of hybrid systems as fixedpoints. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 176–189. Springer, Heidelberg (2008)
38. Platzer, A., Clarke, E.M.: Formal verification of curved flight collision avoidance maneuvers: A case study. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 547–562. Springer, Heidelberg (2009)
39. Platzer, A., Quesel, J.-D.: KeYmaera: A hybrid theorem prover for hybrid systems (System description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 171–178. Springer, Heidelberg (2008)

40. Platzer, A., Quesel, J.-D.: European Train Control System: A Case Study in Formal Verification. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 246–265. Springer, Heidelberg (2009)
41. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation-based abstraction refinement. Trans. on Embedded Computing Sys. 6(1), 8 (2007)
42. Rönkkö, M., Ravn, A.P., Sere, K.: Hybrid action systems. Theor. Comput. Sci. 290(1), 937–973 (2003)
43. Tavernini, L.: Differential automata and their discrete simulators. Non-Linear Anal. 11(6), 665–683 (1987)