Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

How to Prove Complex Properties of Hybrid Systems with KeYmaera: A Tutorial[⋆]

Jan-David Quesel, Stefan Mitsch, Sarah Loos, Nikos Aréchiga, André Platzer

Carnegie Mellon University, Pittsburgh, PA, USA e-mail: {jquesel|smitsch|sloos|arechiga|aplatzer}@cmu.edu

The date of receipt and acceptance will be inserted by the editor

Abstract. This paper is a tutorial on how to model and prove complex properties of complex hybrid systems in KeYmaera, an automatic and interactive formal verification tool for hybrid systems implementing differential dynamic logic. Hybrid systems can model highly nontrivial controllers of physical plants, whose behaviors are often safety critical such as trains, cars, airplanes, or medical devices. Formal methods can help design systems that work correctly. This paper illustrates how KeYmaera can be used to systematically model, validate, and verify hybrid systems. We develop tutorial examples that illustrate challenges arising in many real-world systems. In the context of this tutorial, we identify the impact that modeling decisions have on the suitability of the model for verification purposes. We show how the interactive features of KeYmaera can help users understand their system designs better and prove complex properties for which the automatic prover of KeYmaera still takes an impractical amount of time. We hope this paper is a helpful resource for designers of embedded and cyber-physical systems and that it illustrates how to master common practical challenges in hybrid systems verification.

1 Introduction

Hybrid systems [2, 15, 24] feature both discrete and continuous dynamics. Hybrid systems are important for modeling and understanding systems with computerized or embedded controllers for physical systems. Prime examples of hybrid systems include cars [17, 30], aircraft [44, 52, 53], trains [46], robots [36], and even audio protocols [25]. The design of any controller for these systems is critical, because malfunctions may have detrimental consequences to the system operation. A number of formal verification techniques have been developed for hybrid systems, but verification is still challenging for more complex applications [1]. Experience can make a big difference when making trade-offs to decide on a modeling style, on the most suitable properties to consider, and on the best way to approach the verification task.

This article introduces hybrid system modeling with differential dynamic logic [37, 38, 40]. Furthermore, we explain how to prove complex properties of hybrid systems with our theorem prover KeYmaera. We intend this paper to be a valuable resource for system designers and researchers who face design challenges in hybrid systems and want to learn how they can successfully approach their verification task. Formal verification is a challenging task, but we argue that it is of utmost importance for safety-critical designs and the coverage benefits compared to traditional incomplete system testing far outweigh the cost. Especially, the possibility of checking and dismissing designs early in the development cycle reduces the risk of design flaws causing costly downstream effects.

Even though some of our findings affect other verification tools, we focus on KeYmaera [45] in this paper. KeYmaera implements differential dynamic logic [37, 38, 40], which is a specification and verification logic for hybrid systems. KeYmaera is based on KeY [6], and is presently the premier theorem prover for hybrid systems. For formal details and more background on the approach behind KeYmaera, we refer to the literature [38–40]. It has matured to a powerful verification tool that has been used successfully to verify cars [30, 33], aircraft [44], trains [46], robots [32], and surgical robots [28]. Like

[⋆] This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181, NSF CNS-0931985, and CNS-1035800. This research was partially supported by the German Research Council (DFG) in SFB/TR 14 AVACS. The second author is supported by DOE CSGF.

with any other verification tool, some decisions in the modeling, specification, and proof approach make verification unnecessarily tedious, while others are computationally more effective. Relative completeness results [38, 40] identify exactly which decisions are critical, but even the decisions that are not can have a dramatic impact on the effectiveness of the verification process in practice [40].

We identify best practices for hybrid systems verification that help practitioners and researchers verify hybrid systems with KeYmaera more effectively. We develop a series of tutorial examples that illustrate how to master increasingly more complicated challenges in hybrid systems design and verification. These examples are carefully chosen to illustrate common phenomena that occur in practice, while being easier to understand than the full details of our specific case studies[1]: here, we illustrate hybrid systems and KeYmaera by considering motion in a series of car models. We emphasize that KeYmaera is in no way restricted to car dynamics but has been shown to work for more general dynamics, including hybrid systems with non-linear differential equations, differential inequalities, and differential algebraic constraints.

2 Introduction to Hybrid Systems Modeling

In this section we exemplify the main concepts of hybrid systems, before we introduce hybrid programs, a program notation for hybrid systems.

2.1 Hybrid Systems by Example

Hybrid systems, as already mentioned, comprise continuous and discrete dynamics. The movement of cars (i. e., their continuous dynamics) can be described by differential equations. Kinematic models based on Newton's laws of mechanics are sufficient for basic car interactions where p is the position of the car, v its velocity and a its acceleration. All these state variables are functions in time t. They observe the following ordinary differential equation (ODE):

$$\frac{\mathrm{d}p}{\mathrm{d}t} = v,\ \frac{\mathrm{d}v}{\mathrm{d}t} = a \equiv p' = v,\ v' = a \tag{1}$$

This ODE models that the position p of the car changes over time with velocity v, and that the velocity v changes with acceleration a. As time domain we use the non-negative real numbers, denoted by $\mathbb{R}_{\geq 0}$, and instead of $\frac{\mathrm{d}p}{\mathrm{d}t}$ we write p' for the time-derivative of p. Observe that equation (1) does not specify how the acceleration a evolves. We could add another differential equation $a' = j$ where j is the jerk, but then the question is how j

[1] Specific case studies include cars [30, 33], aircraft [44], trains [46], robots [32], and surgical robots [28]. The models of these case studies are included in KeYmaera.

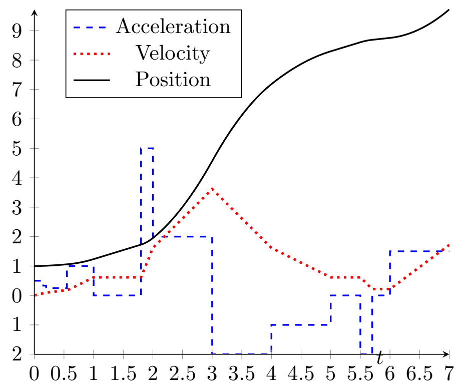


Fig. 1: One example trace of a hybrid system for car dynamics (1) with the acceleration signal changing as indicated over time

evolves. KeYmaera follows the *explicit change* principle. That is, no variable changes unless the model explicitly specifies how it changes. In particular, the absence of a differential equation for a in (1) indicates a is constant during this continuous evolution.

If we want to model an *analog controller* for a, we can replace a in (1) by a term that describes how the analog controller sets the acceleration a, depending on the current position p and velocity v. For example, if v_s is the set-value for the velocity, we could describe a simple proportional controller with gain K_p by the differential equation $p' = v, v' = K_p(v - v_s)$.

A common alternative is to use a *discrete controller*, which turns the purely continuous dynamical systems into a *hybrid system* that exhibits both discrete and continuous dynamics. A discrete controller instantaneously sets values at particular points in time. An example trajectory is shown in Fig. 1 for the car dynamics (1), controlled by a discrete controller for the acceleration a that changes its values at various instants in time. The figure traces the values of the system state variables p, v, and a over (real-valued) time t. The acceleration a changes its value instantaneously according to some discrete controller (not specified in (1)) and this effect propagates to the velocity and position according to the relations given by the differential equation (1).

Given a target speed v_s suppose we want to build a discrete controller that chooses a constant positive acceleration of A if the current speed is too low and a constant deceleration of $-B$ if it is too high. (2) shows a hybrid system that includes such a controller.

$$\big(\text{if } v \leq v_s \text{ then } a := A \text{ else } a := -B \text{ fi}; (p' = v, v' = a)\big)^* \tag{2}$$

The first statement here is a case distinction started with if and ended with fi. It first checks whether the current velocity v is less than or equal to the desired velocity v_s (i. e., whether $v \leq v_s$ holds). If that is the case then the car chooses to accelerate by executing $a := A$. This means that the value of a gets updated to the value of

A. In the following, we assume A is the maximal acceleration. Otherwise, i. e., if $v > v_s$ then the assignment $a := -B$ gets executed, assigning the maximal deceleration of $-B$ to a. The operator ; is for sequential composition. That is, after the first statement finishes (here, the if statement) the next statement is executed, here the differential equation system $(p' = v, v' = a)$. Hence after the controller chooses an acceleration, the variables evolve according to the solution of this differential equation system. During this evolution the acceleration a is constant. Operator * denotes nondeterministic repetition like in a regular expression. That is, the sequence of the discrete controller and the differential equation system are repeated arbitrarily often. The loop, in our example, enables the discrete controller to update the acceleration.

A common and useful assumption when working with hybrid systems is that discrete actions do not consume time (whenever they do consume time, it is easy to transform the model to reflect this just by adding explicit extra delays). Because discrete actions are assumed not to consume time, multiple discrete actions can occur at the same real point in time.

The model (2) does not specify when the continuous evolution stops to give the discrete controller another chance to react. This is because the number of loop iterations as well as the evolution times are chosen nondeterministically (even no repetition and evolution for zero duration are allowed). Hybrid systems with differential-algebraic equations can introduce an upper bound on the continuous dynamics. We model such an upper bound on time in (3) with a clock variable c. That is, we ensure that at least every ε time units the discrete controller might take action.

$$\big(\text{if } v \le v_s \text{ then } a := A \text{ else } a := -B \text{ fi}; \tag{3}$$
$$c := 0;\ (p' = v, v' = a, c' = 1 \ \&\ c \le \varepsilon)\big) \tag{4}$$

The clock c is reset to zero by the discrete assignment $c := 0$ before every continuous evolution and then evolves with a constant slope of $c' = 1$. The formula $c \le \varepsilon$ that is separated from the differential equation by & is an *evolution domain constraint*. Evolution domain constraints are formulas that restrict the continuous evolution of the system to stay within that domain. This means, the continuous evolution starts within the specified domain and must stop before it leaves this region. Therefore, the continuous evolution in (3) evolves for at most ε time units, i. e., the discrete controller is invoked at least every ε time units because any continuous evolution for more than ε time units violates the evolution domain constraint $c \le \varepsilon$. This model paradigm ensures that if the controller is implemented on faster hardware, then it will still have the same safety properties.

Note that the model (3) only puts an upper bound on the duration of a continuous evolution, not a lower bound. The discrete controller can react faster than ε and, in fact, in Fig. 1, it does react more often.

The next extension to our model adds nondeterministic choice of the acceleration. If, as in (5), we replace the assignment $a := A$ by $a := A \cup a := 0$ (read "a becomes A *or* a becomes 0"), then the controller can always choose to keep its current velocity instead of accelerating further. We use $\cup$ to be a nondeterministic choice, meaning the program can unconditionally follow either way.

$$\begin{aligned}\big(&\text{if } v \le v_s \text{ then } a := A \cup a := 0\\ &\qquad\text{else } a := -B \text{ fi};\\ &c := 0; (p' = v, v' = a, c' = 1 \ \&\ c \le \varepsilon)\big)^*\end{aligned} \tag{5}$$

In summary, nondeterministic choice, repetition, and assignment are important modeling constructs for safety verification purposes, because they allow us to capture the safety-critical aspects of many different controllers all within a single model.

2.2 Hybrid Programs

The program model for hybrid systems that we have illustrated by example is called *hybrid programs* (HP) [38–40]. The syntax of hybrid programs is shown together with an informal semantics in Table 1. KeYmaera also supports an ASCII variation of the notation in Table 1. The basic terms (called θ in the table) are either rational number constants, real-valued variables or (possibly nonlinear) polynomial or rational arithmetic expressions built from those.

The effect of $x := \theta$ is an instantaneous discrete jump assigning the value of θ to the variable x. For example in Fig. 1, the acceleration a changes instantaneously at time 1.8 from 0 to 5, by the discrete jump $a := A$ when A has value 5. The term θ can be an arbitrary polynomial. For a car with current velocity v the deceleration necessary to come to a stop within distance m is given by $-\frac{v^2}{2m}$. The controller could assign this value to the acceleration by the assignment $a := \frac{v^2}{2m}$.

The effect of $x' = \theta \,\&\, F$ is an ongoing continuous evolution controlled by the differential equation $x' = \theta$ that is restricted to remain within the evolution domain F, which is a formula of arithmetic. The evolution is allowed to stop at any point in F but it must not leave F. Systems of differential equations and higher-order derivatives are defined accordingly: $p' = v, v' = -B \,\&\, v \ge 0$, for instance, characterizes the braking mode of a car with braking force B that holds within $v \ge 0$ and stops any time before $v < 0$. The extension to systems of differential equations is straight forward, see [38–40].

For discrete control, the test action $?F$ is used to define conditions. It succeeds without changing the state if F is true in the current state, otherwise it aborts all further evolution. For example, a car controller can check whether the chosen acceleration is within physical limits by $? - B \le a \le A$. If a computation branch does not satisfy this condition, the branch is discontinued and aborts. From a modeling perspective, tests should

Table 1: Simple Statements of hybrid programs (F is a first-order formula, α, β are hybrid programs)

Statement	Effect
$\alpha;\ \beta$	sequential composition where β starts after α finishes
$\alpha \cup \beta$	nondeterministic choice, following either alternative α or β
α^*	nondeterministic repetition, repeating α n times for any $n \in \mathbb{N}$
$x := \theta$	discrete assignment of the value of term θ to variable x (jump)
$x := *$	nondeterministic assignment of an arbitrary real number to x
$\big(x_1' = \theta_1, \dots, x_n' = \theta_n \& F\big)$	continuous evolution of x_i along the differential equation system $x_i' = \theta_i$ restricted to evolution domain F
$?F$	test if formula F holds at current state, abort otherwise
if F then α fi	perform α if F is true at current state, do nothing otherwise
if F then α else β fi	perform α if F is true at current state, perform β otherwise

fail if a branch is not possible in the original system we are currently modeling. Therefore, during verification we consider only those branches of a system where all tests succeed.

From these basic constructs, more complex hybrid programs can be built in KeYmaera similar to regular expressions. The *sequential composition* $\alpha;\beta$ expresses that hybrid program β starts after hybrid program α finishes, as in Expression (2). The nondeterministic choice $\alpha \cup \beta$ expresses alternatives in the behavior of the hybrid system. Nondeterministic repetition α^* says that the hybrid program α repeats an arbitrary number of times, including zero. These operations can be combined to form any other control structure.

For instance, $(?v \geq v_s;\ a := A) \cup (?v \leq v_s;\ a := -B)$ says that, depending on the relation of the current speed v of some car and a given target speed v_s, a is chosen to be the maximum acceleration A if $v \leq v_s$ or maximum deceleration $-B$ if $v \geq v_s$. If both conditions are true (hence, $v = v_s$) the system chooses either way. Note that the choice between the two branches is made nondeterministically. However, the test statements abort the program execution if the left branch was chosen in a state where $v \geq v_s$ does not hold, or the right branch was chosen in a state where $v \geq v_s$ was not satisfied. In other words, only one choice works out unless $v = v_s$ in which case either $a := A$ or $a := -B$ will be run. As abbreviations, KeYmaera supports *if-statements* with the usual meanings from programming languages. The if-statement can be expressed using the test action, sequential composition and the choice operator.

$$\begin{aligned} \text{if } F \text{ then } \alpha \text{ fi } &\equiv (?F;\alpha) \cup (?\neg F) \\ \text{if } F \text{ then } \alpha \text{ else } \beta \text{ fi } &\equiv (?F;\alpha) \cup (?\neg F;\beta) \end{aligned}$$

Its semantics is that if condition F is true, the then-part α is executed, otherwise the else-part β is performed, if there is one, otherwise the statement is just skipped. Note that, even though we use nondeterministic choice in the encoding, the choice becomes deterministic as the conditions in the test actions are complementary, so exactly one of the two tests $?F$ and $?\neg F$ fails in any state.

The *nondeterministic assignment* $x := *$ assigns any real value to x that is, every time $x := *$ is run, an arbitrary real number will be put into x, possibly a different one every time. Thereby, $x := *$ expresses unbounded nondeterminism that can be used, for example, for modeling choices for controller reactions. For instance, the idiom $a := *; ?a > 0$ nondeterministically assigns any positive value to the acceleration a, because only positive choices for the value of a will pass the subsequent test $?a > 0$. Any negative assignments will fail.

2.3 Differential Dynamic Logic

KeYmaera implements *differential dynamic logic* $\mathsf{d}\mathcal{L}$ [37, 38, 40] as a specification and verification language for hybrid systems. The formulas of $\mathsf{d}\mathcal{L}$ can be used to specify the properties of the hybrid systems of interest. The logic $\mathsf{d}\mathcal{L}$ also comes with a proof calculus [37, 38, 40] that has been implemented in KeYmaera and can be used to prove these properties and, thus, verify their correctness.

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 $\mathsf{d}\mathcal{L}$ formulas are built using the operators in Table 2 where $\sim \in \{>, \geq, =, \neq, \leq, <\}$ is a comparison operator and θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals. 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. So $[\alpha]\phi$ is true in exactly those states from which running α only leads to states that satisfy 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. For example $[\alpha]\phi \wedge [\beta]\psi$ is true in those states where all executions of α lead to states satisfying ϕ and executing β only reaches states satisfying ψ. Using modalities and propositional connectives, we can express Hoare triples by $\phi \rightarrow [\alpha]\psi$. Here the formula ϕ

Table 2: Operators and (informal) meaning in differential dynamic logic (d$\mathcal{L}$)

d$\mathcal{L}$	Operator	Meaning
$\theta_1 \sim \theta_2$	comparison	true iff $\theta_1 \sim \theta_2$ with $\sim \in \{=, >, \geq, <, \leq\}$
$\neg\phi$	negation / not	true if ϕ is false
$\phi \land \psi$	conjunction / and	true if both ϕ and ψ are true
$\phi \lor \psi$	disjunction / or	true if ϕ is true or if ψ is true
$\phi \to \psi$	implication / implies	true if ϕ is false or ψ is true
$\phi \leftrightarrow \psi$	bi-implication / equivalent	true if ϕ and ψ are both true or both false
$\forall x\, \phi$	universal quantifier	true if ϕ is true for all values of variable x
$\exists x\, \phi$	existential quantifier	true if ϕ is true for some values of variable x
$[\alpha]\phi$	$[\cdot]$ modality / box	true if ϕ is true after all runs of HP α
$\langle\alpha\rangle\phi$	$\langle\cdot\rangle$ modality / diamond	true if ϕ is true after at least one run of HP α

serves as a precondition. This means that the system is required to fulfill the postcondition ψ only when the initial state is in ϕ. The logic d$\mathcal{L}$ 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, $\exists p\,([\alpha]\phi \land [\beta]\psi)$ says that there is a choice of parameter p that makes both $[\alpha]\phi$ and $[\beta]\psi$ true, simultaneously. This is, the choice makes $[\alpha]\phi \land [\beta]\psi$ true, i. e. the 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.

Note that differential equations of d$\mathcal{L}$ [38] constitute a crucial generalization compared to discrete dynamic logic [47]. Another important change is that d$\mathcal{L}$ is defined over the domain $\mathbb{R}$, not natural numbers. The formal semantics of differential dynamic logic and more details about it can be found in [38, 40].

3 Proving with KeYmaera

KeYmaera is an interactive theorem prover. It works by decomposing the verification task into several subtasks. The boolean structure of the input formula is stepwise transformed into a proof tree (where applicable). Programs are handled by symbolic execution. That is for each program construct there is a rule that calculates its effect. For instance, assignments $x := \theta$ can be handled by replacing every occurrence of x by θ in the postcondition. Choices in the program flow are explored separately. For loops KeYmaera uses (inductive) invariants. An inductive invariant is a formula that is satisfied in the current state and starting from any state satisfying the invariant executing the loop body leads into a state also satisfying the invariant. Hence we can do induction and argue that starting from the state just reached we will end up in states satisfying the invariant. In order to use this pattern for reasoning about formulas that are not inductive invariants themselves we add a third task: We have to show that the property we want to show is a consequence of the invariant.

For differential equations there are two routes to go. If the ODE happens to have a polynomial solution, we can replace it by a discrete assignment at each point in time t. That is we assign the value of its solution at time t. However, if there is no polynomial solution available this would yield formulas in an undecidable theory. Thus, we go a different route in those cases. That is, we can apply the same idea of induction in order to strengthen the evolution domain constraint. That is we apply differential induction [39] where we show that the derivative of the evolution domain candidate points inwards w.r.t. the region it characterizes. That way, we can be sure that if we start within that region we will never leave it. Thus, assuming we can show that we are initially in that region, adding it to the evolution domain constraints does not restrict the system any further. Still, it gives us insight in which regions are reachable by the system. If at some point we are able to show that our post-condition is covered by the evolution domain constraint we are sure that it will be satisfied by each run of the system.

Once we have dealt with all the modalities in the formulas we end up with a first-order formula over the reals. Validity of those can be decided by quantifier elimination [50]. The original method proposed by Tarski however has non-elementary complexity. Even worse Davenport and Heintz have shown that the worst-case complexity of such a procedure will always be doubly exponential [14]. Still, we can use quantifier eliminiation in many practical examples. Beyond that, KeYmaera interfaces with a number of tools and implements algorithms to deal with special cases more efficiently.

4 Related Tools

There has been significant research on hybrid system verification and related approaches include a number of hybrid systems verification tools under active development.

The ultimate goal of these approaches is to provide fully automated verification tools for hybrid systems. Unfortunetely, this is provably impossible. Therefore, different compromises have been made. On the one hand, fully automated tools work on restricted classes of hybrid systems, but the procedures might not terminate. On the other hand, semi-automated tools that automatically explore the state space and fall back to the user where the automated search fails. For the latter the user can then use domain knowledge to steer the tool into a promising direction within an uncountably branching space.

An interesting subclass of hybrid systems are real-time systems [35] in which all continuous veriables represent clocks instead of physical motion in space. That is the derivatives of all these variables are 1. However, no changes to the derivative are allowed and the computations are limited to resetting variables to 0. Even though this sounds limiting a number of interesting systems can be analyzed in that way. The main advantage is that reachability is decidable for real-time systems, whereas it is undecidable for hybrid systems. Toolwise, we like to point the reader to Uppaal [29], a model checker for timed automata, a common model for real-time systems basded on smart, exhaustive, set-valued simulation of the system. Uppaal has been extended to a verification tool for priced timed-automata [3, 8]. Priced timed automata extend real-time systems with variables that can have constant but arbitrary and changing slopes. However, they cannot be used in any way that influences the reachability relation. That is, they can neither restrict evolutions nor switching. However, cost optimal reachability is decidable and implement by Uppaal CORA [7].

Model checking [12] tools are applicable to restricted classes of hybrid systems, mostly those with linear dynamics. Among the first tools following this line are HyTech [25], d/dt [5], and PHAVer [20]. PHAVer, which superseded HyTech, was recently superseded itself by SpaceEx [21]. SpaceEx [21] is a fully automated tool for verification of hybrid systems with linear, affine dynamics. FOMC [13] provides methods tailored to systems with large discrete state spaces. All these tools perform an exhaustive search of the state space fully automatically. If they succeed no user interaction is necessary. However since reachability for hybrid systems is an undecidable question, termination of the automated procedures is not guaranteed and there is no way for the user to steer them into promising directions. Still, model checking is an extremely versatile approach which is especially good in finding counter examples. Thus, it can also be used as complementary approach to approaches tailored to showing the absence of errors.

For non-linear hybrid systems the tools are often based on numerical methods. That means that unlike KeYmaera, they rely on specific numbers or bounds on the range of the variables used to describe the problem. Flow* [11] can be used for bounded model checking of non-linear hybrid systems. That is given a time horizon and bound on the number of jumps, the tool constructs an overapproximation of the reachable states. The iSAT algorithm [19] tightly couples interval constraint propagation with methods from SAT solving, thereby providing a solver for boolean combinations of nonlinear constraints (including transcendental functions) over discrete and continuous variables with bounded ranges. Its most recent implementation [31] integrates—among other improvements—Cylindrical Algebraic Decomposition to boost reasoning for polynomial constaints. The iSAT-tool can be used for Bounded Model Checking (BMC) of hybrid systems by using overapproximations or exact solution functions of the differential equations in the finite unwinding of the transition system. The iSAT solver has been extended to iSAT-ODE [18] by embedding validated enclosure methods for the solution sets of non-linear ordinary differential equations (ODEs), allowing BMC of hybrid systems without manual overapproximation or solving of the ODEs. For probabilistic discrete-time hybrid systems, the SiSAT solver [51] allows the computation of reachability probabilities. For low-dimensional systems HSolver [49] offers methods for unbounded horizon reachability analysis of hybrid systems. It implements abstraction refinement based on interval constraint propagation.

In contrast to that, KeYmaera is a semi-automated tool for unbounded horizon, purely symbolic verification of hybrid systems and hybrid games, which can deal with a rich set of continuous dynamics and symbolic parameters at the same time. It performs automated proof search and allows the user to interact and steer the prover in cases where the heuristics fail. A strong point of KeYmaera is the automatated decomposition of the original verification problem into smaller subtasks while retaining a clear connection to the original problem. This allows the user to focus on the difficult cases, where interaction is necessary and let the prover take care of those cases where the necessary steps can be performed automatically. Still, in constrast to fully automated tools, some knowledge about the core ideas behind KeYmaera is necessary to apply it successfully to complex systems. The following sections are meant to provide an easy to follow introduction into these ideas based on a running example from the car domain.

5 KeYmaera Tutorial

Starting from a simple example, we develop a series of increasingly more complex systems which illustrate how modeling and verification challenges can be handled in KeYmaera. For additional and more detailed examples, as well as step-by-step instructions to follow along in KeYmaera, see `http://symbolaris.com/info/`

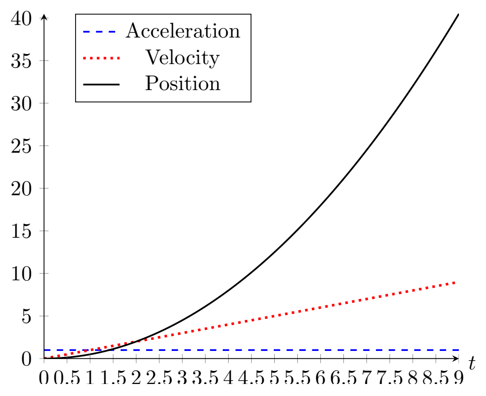


Fig. 2: Simulated trace of Example 1

`KeYmaera-tutorial.html`. The example files in this paper can also be found in the project *KeYmaera Tutorial* of KeYmaera.

5.1 *Example 1: Uncontrolled Continuous Car Model*

First we will look at a simple system in which a car starts at some positive velocity and accelerates at constant rate along a straight lane. The requirement we want to prove is that the car always travels forward in space. Example 1 captures the setup of this scenario: when starting at the initial conditions `init`, all executions of the car $[plant]$ must ensure the requirements `req`. The scenario setup is expressed using the $\mathsf{d}\mathcal{L}$ formula $\texttt{init} \rightarrow [plant](\texttt{req})$: the initial conditions are in the antecedent of a logical implication; the (hybrid) program and the requirement form its consequent. We used the box modality $[plant]$ to express that *all* states reachable by the continuous model of the system *plant* satisfy our requirements `req`.

The initial conditions are formally specified in formula (7): the velocity *and* the acceleration must both be positive initially ($v \geq 0 \wedge A > 0$). In this example, the *plant* is very simple, cf. formula (8): the derivative of position is velocity ($p' = v$) and the derivative of velocity is acceleration ($v' = A$). Finally, formula (9) states that the velocity of the car is positive $v \geq 0$ and, thus, captures our requirement that the car always travels forward in space. Note, that many different ways exist to model even such a simple system: in formulas (10)–(11) we use an additional variable to remember the initial position of the car while still using velocity in the plant.

KeYmaera proves all these models automatically. In Example 1 we modeled only continuous components in the *plant*. In the next example we will allow a discrete controller to interact with the system.

5.2 *Example 2: Safety Property of Hybrid Systems*

Example 1 had a plant but no controller. This means that, once started, the car would drive for a possibly infinite amount of time without any option to ever change its initial decisions. In this example, we introduce a discrete controller, *ctrl*, into the model of the system given in the hybrid program Example 2. The task of the controller in this example is to adjust the velocity by accelerating or braking, and still always drive forward.

Example 1 Safety property of an uncontrolled continuous car model

$$\texttt{init} \rightarrow [plant]\,(\texttt{req}) \tag{6}$$
$$\texttt{init} \equiv v \geq 0 \wedge A > 0 \tag{7}$$
$$plant \equiv p' = v, v' = A \tag{8}$$
$$\texttt{req} \equiv v \geq 0 \tag{9}$$

$$\texttt{init} \equiv v \geq 0 \wedge A > 0 \wedge p_0 = p \tag{10}$$
$$\texttt{req} \equiv p \geq p_0 \tag{11}$$

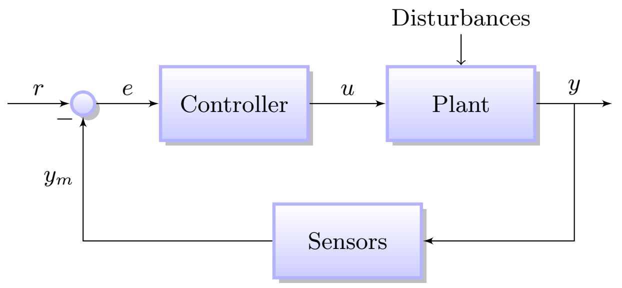


Fig. 3: Closed-loop feedback control system principle

The example follows closed-loop feedback control, which is a typical control system principle, as depicted in Fig. 3: a controller tries to minimize the error e (difference between a desired output response r and sensed output measurements y_m) by computing set values u as input for a plant. The plant, depending on some disturbances, produces an actual output response y, which is fed back into the controller through sensors as measurements.

Example 2 shows the model, which was extended from Example 1. The essential difference is the hybrid program in formula (12), whose controller *ctrl* is repeated together with the *plant* nondeterministically often in a loop. The state transition system of this hybrid program is depicted in Fig. 4a. The initial conditions in formula (13) now contain a specification for braking force $B > 0$. The controller has three simple options as stated in formula (14): it may cause the car to accelerate with rate $A > 0$, maintain velocity, or brake with rate $-B < 0$. We model the control options as a nondeterministic choice ($\cup$) in order to verify multiple concrete controllers at once. Because the hybrid program is within a box modality, whether the controller chooses to accelerate, maintain velocity, or brake, `req` must always hold.

When a real car brakes, it decelerates to a complete stop—it is not possible to drive a car backwards by braking. In order to model this, in formula (15) we extended the plant from the previous example and prevent the continuous dynamics from evolving beyond what is possible in the real world. So, even though evolving over time with $p'' = -B$ would eventually cause the car to drive backward, we disallow these traces by adding an evolution domain constraint of $v \geq 0$ in the plant (separated by &), which restricts the model of the car to realistic movement.

Example 2 Safety property of a hybrid car model

$$\texttt{init} \rightarrow [(ctrl; plant)^*]\ (\texttt{req}) \tag{12}$$
$$\texttt{init} \equiv A > 0 \wedge B > 0 \wedge v \geq 0 \tag{13}$$
$$ctrl \equiv a := A \cup a := 0 \cup a := -B \tag{14}$$
$$plant \equiv p' = v, v' = a \ \&\ v \geq 0 \tag{15}$$
$$\texttt{req} \equiv v \geq 0 \tag{16}$$

We also want the discrete controller to be able to change the acceleration of the vehicle at any time. Like in a regular expression, the nondeterministic repetition * creates a loop over the *ctrl* and *plant*. The *plant* evolves for an arbitrary amount of time (it may even evolve for zero time) as long as it satisfies the evolution domain. When the *plant* completes, the program loops back to the *ctrl* which is again allowed to choose between accelerating, maintaining velocity, or braking. All the states that are reachable by this program must satisfy the requirement `req` of formula (16), which is the same as in the previous example.

Fig. 4b shows a sequence of control choices that govern the plant for varying plant execution duration. The resulting sample trace of the continuous change of the car's velocity v and position p, which follows from these control decisions, is shown in Fig. 4c.

In order to prove properties of a loop, we need to identify an invariant, which is a formula that is true whenever the loop repeats. That is, a formula `inv` that is initially valid ($\texttt{init} \rightarrow \texttt{inv}$), that implies the postcondition (use case $\texttt{inv} \rightarrow \texttt{req}$), and where the loop body preserves the invariant (here $\texttt{inv} \rightarrow [ctrl; plant]\texttt{inv}$). Invariants are critical parts of the system design. As such, they should always be communicated as part of the system, for which KeYmaera provides annotations:

$$\texttt{init} \ \rightarrow \ [(ctrl; plant)^* \text{@invariant}(v \geq 0)](\texttt{req})$$

The @invariant(`inv`) annotation for a loop indicates that `inv` is a loop invariant candidate. KeYmaera uses this annotation to find proofs more efficiently. KeYmaera, otherwise, tries to compute an invariant automatically [43] as it does in Example 2.

As a guideline for finding such invariants manually, we may use the following heuristics inspired by model-predictive control.

Guideline 51 (Invariant) *We start at our safety condition, which states that all possible dynamics have to fulfill the safety constraint.*

$$[p' = v, v' = a \ \&\ v \geq 0]\,(v \geq 0)$$

Then we choose the worst-case branch from our controller that is most likely to violate the safety constraint, which in this example is braking:

$$[p' = v, v' = -B \ \&\ v \geq 0]\,(v \geq 0)$$

We solve the dynamics ODE and get

$$[v - Bt \geq 0 \wedge v \geq 0]\,(v \geq 0) \ .$$

Since in any case either evolution for zero time or the evolution domain ensure $v \geq 0$, this is a likely candidate for a good invariant.

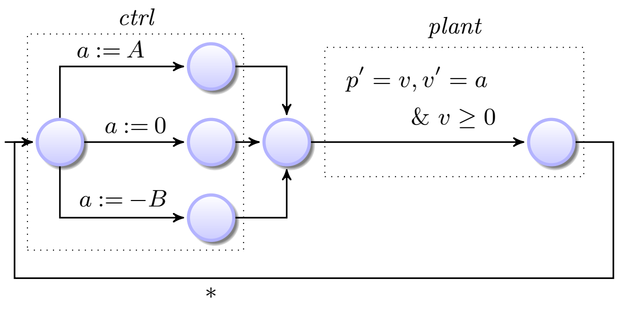
5.3 Example 3: Safety Property of an Event-Triggered Hybrid System

Now we will add some complexity to the system and the controller. We want to model a stop sign assistant: while the car is driving down the lane, the controller must choose when to begin decelerating so that it stops at or before a stop sign. This means that it is no longer sufficient to let the controller run at arbitrary points in time as in Example 2, since the controller now must brake when it approaches a stop sign. Thus, we have to change our model to prevent the plant from running an infinite amount of time. We can do this by adding an additional constraint to the evolution domain of the plant. Depending on the nature of this additional constraint we either speak of an *event-triggered system* or a *time-triggered* system. The former interrupts the plant when a particular event in the environment occurs (e. g., when the car is too close to a stop sign), while the latter interrupts the plant at periodic times (e. g., every 50 ms).

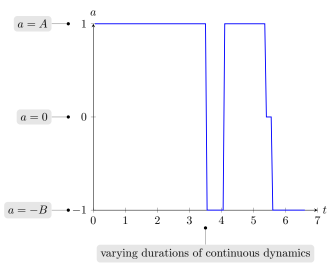
We will start with an event-triggered system, since those are often easier to prove than time-triggered systems. A time-triggered model will be discussed in Example 5 in the next section.

Example 3a Stop sign controller (event-triggered)

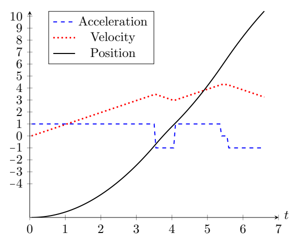
$$\texttt{init} \rightarrow [(ctrl; plant)^*](\texttt{req}) \tag{17}$$
$$\texttt{init} \equiv \texttt{Safe} \wedge A > 0 \wedge B > 0 \wedge v \geq 0 \tag{18}$$
$$\texttt{Safe} \equiv p + \frac{v^2}{2B} \leq S \tag{19}$$
$$ctrl \equiv (?\texttt{Safe}; a := A) \cup (?v = 0; a := 0) \cup (a := -B) \tag{20}$$
$$plant \equiv p' = v, v' = a \ \&\ v \geq 0 \wedge \texttt{Safe} \tag{21}$$
$$\texttt{req} \equiv p \leq S \tag{22}$$



(a) Transition system

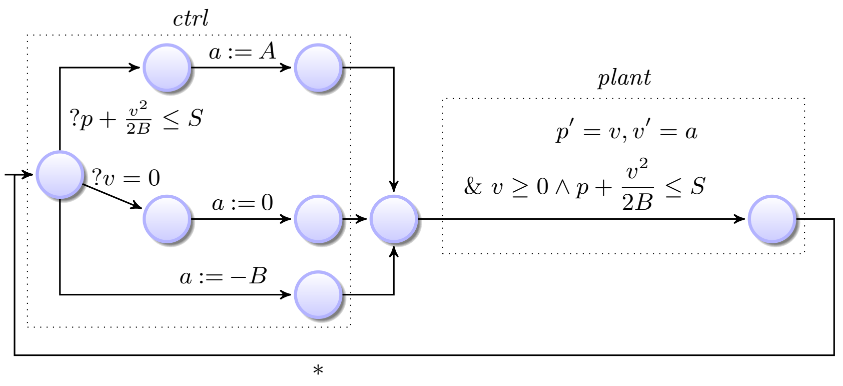


(b) Control decisions: coast, accelerate and brake; nondeterministic duration of continuous dynamics

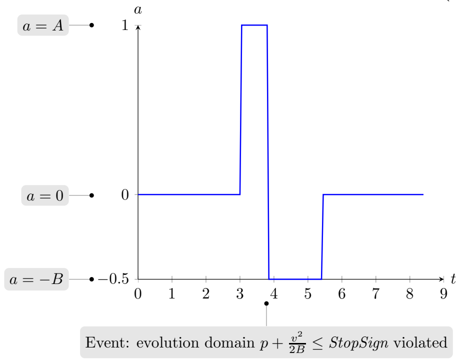


(c) Sample trace: velocity and position per acceleration choice

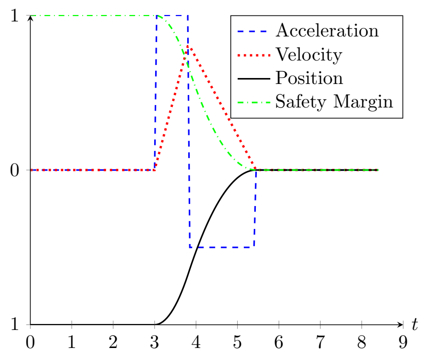
Fig. 4: A hybrid car controller (Example 2)


ctrl
$?p + \frac{v^2}{2B} \le S$
a := A
?v = 0
a := 0
a := −B
plant
$p' = v, v' = a$
$\& \; v \ge 0 \wedge p + \frac{v^2}{2B} \le S$
*

(a) Transition system



(b) Control decisions: first coast, then accelerate until evolution domain constraint is violated, brake, and finally stay stopped



(c) Sample trace: velocity and position per acceleration choice; distance to stop sign $|p - S|$ gives the remaining safety margin

Fig. 5: Event-triggered stop sign controller (Example 3a)

The stop sign assistant is modeled in Example 3a and depicted in Fig. 5a. The basic setup of the model in formula (17) is the same as in Example 2. However, we have to adapt the initial condition in (18) such that the car starts at a position that is still sufficiently distant from the stop sign (`Safe`). Intuitively, the car is at a safe position, if it can still stop before it exceeds the position of the stop sign. Using kinematic equations, we derive that the stopping distance of the car when decelerating at rate $-B$ is $\frac{v^2}{2B}$; thus, the proposition `Safe` is true when the current position plus the stopping distance of the car does not exceed the position of the stop sign, as specified in formula (19).

The controller in formula (20) still chooses between accelerating, maintaining velocity, and braking, but the first two options are not allowed if the car is too close to the stop sign. We restrict the choice of accelerating by adding the test ?`Safe`, so that the car may only accelerate if it is still sufficiently distant from the stop sign. We also only allow the car to maintain velocity ($a := 0$) when it is already stopped, since otherwise the car could coast through the intersection. The proposition `Safe` is added as event trigger to the evolution domain, cf. (21). This ensures that the controller executes if the car comes within the minimum stopping distance of the stop sign; however, the controller is free to execute at any time before this point is reached to adapt acceleration as needed. Finally, the requirement `req` in formula (22) defines that in all states, which are reachable by the event-triggered hybrid program, the position of the car must not exceed the position of the stop sign $p \leq S$.

An example for event-triggered control and its effects is shown in Fig. 5b and 5c: the car accelerates until the evolution domain constraint triggers braking, which causes the car to stop smoothly at the stop sign.

Guideline 52 (Evolution domain) *In order to derive the evolution domain for event-triggered control, again, we can rely on model-predictive control. We start at the safety condition and the kinematic equations of the car.*

$$[p' = v, v' = a \;\&\; v \geq 0]\,(p \leq S)$$

To be safe, we have to interrupt the continuous dynamics at the latest when the car can still stop in the remaining distance to the stop sign with braking power $-B$*:* $[p' = v, v' = -B \;\&\; v \geq 0]\,(p \leq S)$*. In order to determine the distance, we first need to find out how long it will take to stop from the current velocity:* $t = \frac{v}{B}$*, which follows from* $v + \int (-B)\,dt = v - Bt = 0$*. The distance that the car will travel until it comes to a full stop from its current velocity thus is* $\int_0^{v/B} (v - Bt)dt = \frac{v^2}{2B}$*, which yields our evolution domain constraint* $p + \frac{v^2}{2B} \leq S$*.*

In this model, deriving the stopping distance of the car to come up with the appropriate equation for the *ctrl* was not difficult; however, for more complex models, the solution may not be so apparent. We may get hints about what *ctrl* should be by first trying to prove safety in a system that is obviously unsafe or that we merely suspect to be safe in some scenarios.

To illustrate this method, in Example 3b we start with a simpler version of Example 3a. We remove the nondeterministic repetition (23), so that we do not yet have to worry about loop invariants. For lack of a better understanding, in `init` (24) we just strive to not violate our requirement $p \leq S$ and place the car somewhere in front of the stop sign. Finally, we replace the safety condition of Example 3a, such that *ctrl* allows the car to choose acceleration without any restrictions, which cannot always be correct. The plant (27) and the requirement (28) remain the same as in Example 3a. Attempting to prove property (23) of Example 3b results in several open goals in which there are formulas which, had they been in the antecedent, the property would have held. Some of these formulas contradict our assumptions in `init` (24), so we ignore them. However, there is one remaining formula which does not contradict any assumption: $B \geq v^2(2S + -2p)^{-1}$. The failed proof attempt indicates that we should change our design to obey this constraint. With some algebraic manipulation, we see that this constraint is almost identical to the restriction we added to the *ctrl* in Example 3a.

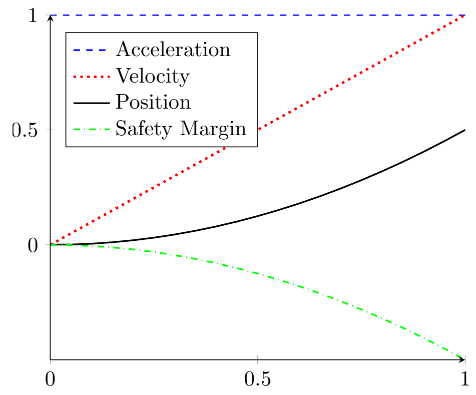


Fig. 6: Sample trace of Example 3b; safety margin becomes negative immediately due to unsafe controller

Example 3b Unsafe stop sign controller design to discover safety constraints

$$\texttt{init} \rightarrow [\mathit{ctrl}; \mathit{plant}](\texttt{req}) \tag{23}$$
$$\texttt{init} \equiv p \leq S \wedge A > 0 \wedge B > 0 \wedge v \geq 0 \tag{24}$$
$$\texttt{Safe} \equiv \texttt{true} \tag{25}$$
$$\mathit{ctrl} \equiv (?\texttt{Safe}; a := A) \cup (?v = 0; a := 0) \cup (a := -B) \tag{26}$$
$$\mathit{plant} \equiv p' = v, v' = a \;\&\; v \geq 0 \wedge \texttt{Safe} \tag{27}$$
$$\texttt{req} \equiv p \leq S \tag{28}$$

It is not uncommon for the first attempt at proving the safety of a system to be unsuccessful because the model is in fact unsafe. KeYmaera allows the user to examine a trace of the hybrid program which obeys the initial conditions, and follows the execution of the hybrid program, but violates the given safety requirement. In Example 3b, there are infinitely many such counterexamples that could be generated; however, one counterexample (which KeYmaera automatically generates) sets the position of the stop sign to be $S = 0$, the initial position and velocity of the car to be $p = -23$ and $v = 986$, and maximum acceleration $A = 38$. These assignments of values to the symbolic parameters are all permissible by the initial conditions. The transition then has the car accelerate at rate A and allows the system to evolve for .1 time steps, at which point the position of the car is $p = 75.79$, so the car has run the stop sign and the requirement $p \leq S$ has been violated, showing that the system is unsafe.

5.4 Example 4: Pitfalls when modeling event triggered systems

Note that there are some pitfalls when modeling event triggered systems. That is, we have to make sure that our model does not restrict the physical behavior unnecessarily in order to react to certain events. Consider a cruise control with the goal of reaching and maintaing a certain velocity, say v_s. A simple event triggered model for this is shown in Example 4. Certainly, we can prove that this controller ensures that the velocity never exceeds the set-value v_s as every time the car reaches velocity v_s it will set the acceleration to 0.

Example 4 Event triggered cruise-control

$$\texttt{init} \;\rightarrow\; [(\mathit{ctrl}; \mathit{plant})^*](\texttt{req}) \tag{29}$$
$$\texttt{init} \equiv v \le v_s \wedge A > 0 \tag{30}$$
$$\mathit{ctrl} \equiv \text{if } v = v_s \text{ then } a := 0 \text{ else } a := A\text{fi} \tag{31}$$
$$\mathit{plant} \equiv p' = v, v' = a \;\&\; v \le v_s \tag{32}$$
$$\texttt{req} \equiv v \le v_s \tag{33}$$

$$\texttt{init} \;\rightarrow\; [(\mathit{ctrl}_f; \mathit{plant})^*](\texttt{req}) \tag{34}$$
$$\mathit{ctrl}_f \equiv a := A \tag{35}$$

$$\texttt{init} \;\rightarrow\; [(\mathit{ctrl}; \mathit{plant}_r)^*](\texttt{req}) \tag{36}$$
$$\mathit{plant}_r \equiv (p' = v, v' = a \;\&\; v \le v_s) \cup (p' = v, v' = a \;\&\; v \ge v_s) \tag{37}$$

$$\texttt{init} \;\rightarrow\; [(\mathit{ctrl}_f; \mathit{plant}_r)^*](\texttt{req}) \tag{38}$$

Unfortunatly, it this is not the reason why we can prove this property. Replacing the controller by one that always chooses to accelerate reveals that the validity of the formula does not depend on our control choices, i.e., formula (34) is valid as well. This stems from the fact that any continuous evolution is already restricted to the domain we consider critical. Thus, there is no transition leaving this domain once we reach its border which makes the property trivially true. However, safety in real world system crucially relies on correct functioning of our controllers. Thus we have to adopt the model to reflect the fact that the car could in some scenarios exceed the velocity we want to maintain and then show that our controller makes sure that it does not do so.

Consider the plant model given in (37). Here we refine the plant in such a way that time may evolove regardless of the relation of v and v_s. Still, the controller will be evidently be invoked and able to update the acceleration once the velocity reaches v_s. However, now since there are transitions that might invalidate our requirement that we never exceed the velocity v_s we can observe a difference between our original controller (31) and the faulty one (35). That is, the formula (36) is valid whereas the formula (38) is not.

5.5 Example 5: Safety Property of a Time-Triggered Hybrid System

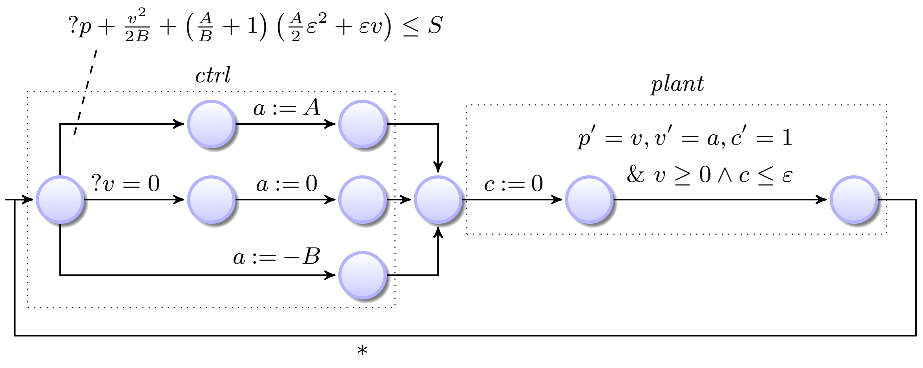
Event-triggered systems like the one in Example 3a make proving easier, but they are difficult (if not impossible) to implement. In order to implement Example 3a faithfully, it would require a sensor which would send position and velocity data continuously, so that it could notify the controller instantaneously when the car crosses the final braking point. A more realistic system is one in which the sensors take periodic measurements of position and velocity and the controller executes each time those sensor updates are taken. However, if we don't restrict the amount of time between updates, then there is no way to control the car safely, since it would essentially be driving blind. Instead, we require that the longest time between sensor updates is bounded by ε. To account for imperfect timing, the controller can also handle updates that come in before the ε deadline. In this section, we implement this system and prove it is safe.

Fig. 7b shows control decisions that follow this principle. Every ε time units the controller senses the velocity and position of the car and makes a new decision to accelerate, stay stopped, or brake. A sample trace of the continuous dynamics resulting from these control decisions is sketched in Fig. 7c.

With this change, we must create a more intelligent controller. There are two essential differences between Example 3a and Example 5 with its transition system depicted in Fig. 7a: Example 5 introduces a clock into the plant (43) that stops continuous dynamics before $c \le \varepsilon$ becomes false. (42) uses the upper bound on that clock in the safety condition that allows the car to accelerate. In Example 3a we used the formula `Safe` to determine whether it was safe for the car to accelerate at the present moment. Now, we must have a controller which not only checks that it is safe to accelerate at present, but also that doing so for up to ε time will still be safe. We use the formula $\texttt{Safe}_\varepsilon$ in Example 5, which checks that while accelerating for ε time, the car will always be able to come to a complete stop before the stop sign.

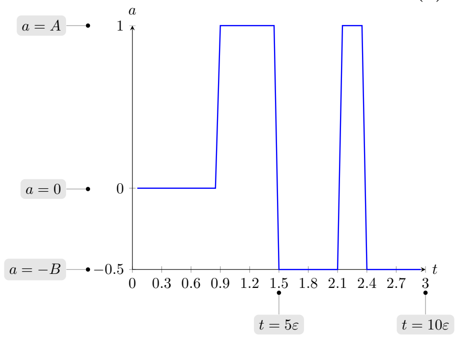
Guideline 53 ($\texttt{Safe}_\varepsilon$ for time-triggered control) *In the case of time triggered control, a decision (e. g., accelerating with A) is safe when after ε time braking is still safe.*

$$[c := 0]\,;$$
$$[p' = v, v' = A, c' = 1 \;\&\; v \ge 0 \wedge c \le \varepsilon]\,;$$
$$[p' = v, v' = -B \;\&\; v \ge 0]\,(p \le S)$$

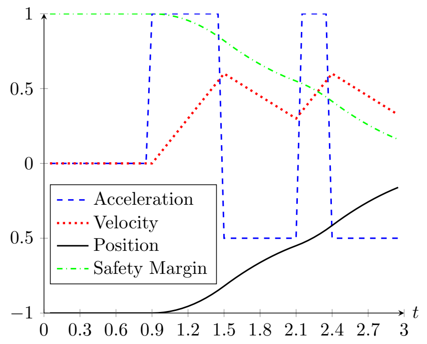


$?p + \frac{v^2}{2B} + \left(\frac{A}{B}+1\right)\left(\frac{A}{2}\varepsilon^2+\varepsilon v\right) \le S$
ctrl
plant
$a := A$
$?v = 0$
$a := 0$
$a := -B$
$c := 0$
$p' = v, v' = a, c' = 1$
$\& \; v \ge 0 \land c \le \varepsilon$
$*$

(a) State transition system



(b) Control decisions: despite nondeterministic loop duration $c_i \le \varepsilon$, the system was simulated with $c = \varepsilon$



(c) Sample trace: velocity and position per acceleration choice; distance to stop sign $|p - S|$ gives the remaining safety margin

Fig. 7: Time-triggered stop sign controller (Example 5)

First, we need to determine the distance traveled while accelerating with A for ε time:

$$\int_0^{\varepsilon} (v + At)dt = \tfrac{A}{2}\varepsilon^2 + \varepsilon v \ .$$

As a next step we need to determine the distance for braking to a full stop from the increased velocity $v + A\varepsilon$:

$$\int_0^{(v+A\varepsilon)/B} (v + A\varepsilon - Bt)dt = \tfrac{v^2}{2B} + \tfrac{A}{B}\left(\tfrac{A}{2}\varepsilon^2 + \varepsilon v\right) \ ,$$

with braking time following from $v + A\varepsilon - Bt = 0$.

Since we already know the distance for braking to a full stop from Guideline 52 ($\frac{v^2}{2B}$), we could alternatively find the distance needed to compensate the increased velocity:

$$\int_0^{A\varepsilon/B} (v + A\varepsilon - Bt)dt = \tfrac{A}{B}\left(\tfrac{A}{2}\varepsilon^2 + \varepsilon v\right)$$

with braking time following from $A\varepsilon - Bt = 0$.

When we add the distance traveled while accelerating with the distance needed to stop afterwards, we get $p + \frac{v^2}{2B} + \left(\frac{A}{B}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v\right) \le S$ as definition for $\mathtt{Safe}_\varepsilon$.

Because we have already proven a very similar system in Example 3a, it may be tempting to simply add a safety margin for how much the position of the car can change in time ε. Since the proof holds for symbolic values which can be arbitrarily large, however, there is no constant error margin large enough that is safe for all controllers.

Example 5 Stop sign controller (time-triggered)

$$\mathtt{init} \rightarrow [(ctrl; plant)^*](\mathtt{req}) \tag{39}$$

$$\mathtt{init} \equiv p + \frac{v^2}{2B} \le S \land A > 0 \land B > 0 \land v \ge 0 \land \varepsilon > 0 \tag{40}$$

$$ctrl \equiv (?\mathtt{Safe}_\varepsilon; a := A) \cup (?v = 0; a := 0) \cup (a := -B) \tag{41}$$

$$\mathtt{Safe}_\varepsilon \equiv p + \frac{v^2}{2B} + \left(\frac{A}{B}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v\right) \le S \tag{42}$$

$$plant \equiv c := 0;\ p' = v, v' = a, c' = 1 \ \&\ v \ge 0 \ \land\ c \le \varepsilon \tag{43}$$

$$\mathtt{req} \equiv p \le S \tag{44}$$

5.6 *Sequential Composition vs. Nondeterministic Choice*

In the previous models we considered a loop $(ctrl; plant)^*$ over the sequential composition of controller and plant. Using sequential composition seems natural when comparing HPs to block diagrams. However, we could use nondeterministic choice $(ctrl \cup plant)^*$ instead. This allows more execution orders, for instance, executing the controller twice without checking the evolution domain, or skipping the controller altogether. Using choice has the advantage that the resulting branches can be explored independently by KeYmaera. However, the model complexity might grow as it is necessary to ensure the controller runs sufficiently often. For Example 5 this could be done by moving the reset of the clock c into the controller. Then at some point no further time can pass until the controller was executed again. Further, in the sequential model the information about the possible control actions is propagated to the plant automatically by the symbolic execution performed by KeYmaera. In the model using choice, it is necessary to transport sufficient information about the control actions using the loop invariant.

5.7 Example 6: Guarded Nondeterministic Assignment

In previous examples, we have only represented controllers which can choose from a discrete choice of accelerations (either A, 0, or $-B$).

A more realistic controller would be able to choose any acceleration within a range of values representing the physical limits of the system. In Example 6 and Fig. 8a we introduce guarded nondeterministic assignment to represent an arbitrary choice of a real value within a given range. In this example, we only need to change the *ctrl* to introduce nondeterministic assignment, while the rest of Example 6 is identical with Example 5: Line (47) of Example 6 assigns an arbitrary real value to a ($a := *$). The subsequent test checks that the value of a is in the interval $[-B, A]$. This operation eliminates all traces which do not satisfy the test, so only traces in which a is in $[-B, A]$ are considered. As a result, when we prove the property in Example 6, we are proving safety for all values of a within $[-B, A]$. Fig. 8b shows an example sequence of the control choices made by such a controller. The resulting trace of the car's velocity and position is depicted in Fig. 8c.

5.8 Example 6: Nondeterministic Over-approximation

A good technique to prove properties that involve complicated formulas is to use nondeterministic over-approximation. If the value of a variable a_x is given by a function $a_x = f(x)$, but the value of $f(x)$ is contained entirely in some interval $[f_1, f_2]$, the proof can often be greatly simplified by omitting the expression for $f(x)$ and simply allowing a_x to nondeterministically take any value in $[f_1, f_2]$ by using guarded nondeterministic assignment as discussed in Section 5.7.

For instance, in Example 6 the car's braking mechanism is modeled simply as choosing a negative acceleration, and is always fixed. Consider a more realistic braking model, like the one outlined in [26]. There, braking is modeled as

$$v' = \frac{1}{M}\left(-c_1 T_b - f_0 - c_2 v - c_3 v^2\right)$$

where T_b is the braking torque, $c_1 T_b$ is the braking force, M is the mass of the car, f_0 is the static friction force, $c_2 v$ is the rolling friction force, and $c_3 v^2$ is aerodynamic drag. This model has five more variables than the previous one. KeYmaera uses quantifier elimination as a decision procedure for first order real arithmetic, which is doubly exponential in the number of variables. Thus, it helps to avoid unnecessary variables. In Example 7, we use a simpler model in which only one new variable is added. In this model, the car's maximum total braking capability is between some symbolic parameters b and B, as modeled in (56). This means that we can only guarantee b as the car's braking capability. In comparison to Example 6, thus, we have to adapt `init` (53) and $\mathtt{Safe}_\varepsilon$ (57) so that both consider the new braking bounds.

Since the controls of real systems are usually deterministic and often complex, it can be useful to prove that the implemented controller is a deterministic refinement of the proved nondeterministic controller. This area is rich with possibilities for future research, but for preliminary methods on refinement, see [4].

Example 6 Stop sign controller (guarded nondeterministic assignment)

$$\mathtt{init} \rightarrow [(ctrl; plant)^*](\mathtt{req}) \tag{45}$$

$$\mathtt{init} \equiv p + \frac{v^2}{2B} \le S \wedge A > 0 \wedge B > 0 \wedge v \ge 0 \wedge \varepsilon > 0 \tag{46}$$

$$ctrl \equiv (?\mathtt{Safe}_\varepsilon; a := *; ? -B \le a \le A) \tag{47}$$

$$\cup (?v = 0; a := 0) \cup (a := -B) \tag{48}$$

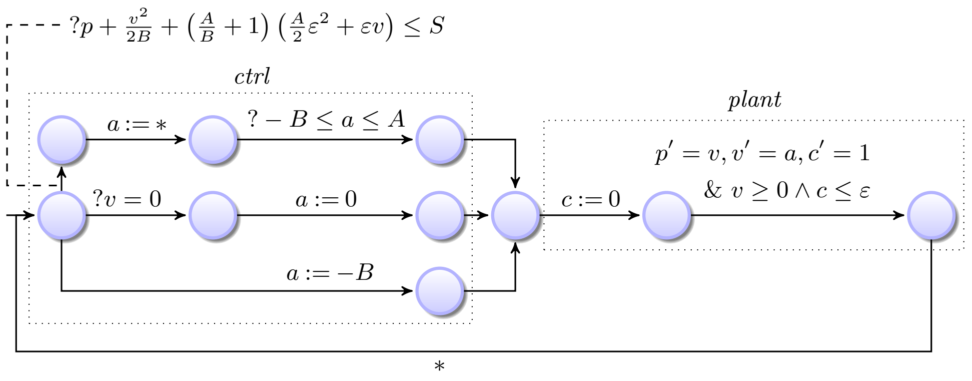
$$\mathtt{Safe}_\varepsilon \equiv p + \frac{v^2}{2B} + \left(\frac{A}{B} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v\right) \le S \tag{49}$$

$$plant \equiv c := 0;\ p' = v, v' = a, c' = 1 \ \&\ v \ge 0 \wedge c \le \varepsilon \tag{50}$$

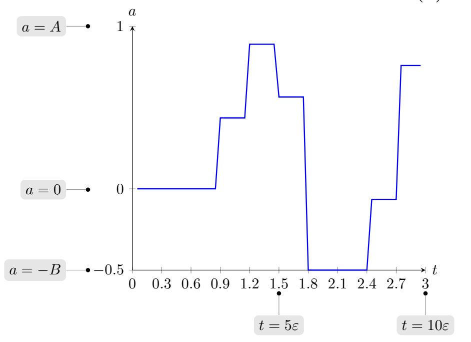
$$\mathtt{req} \equiv p \le S \tag{51}$$

5.9 Example 7: Differential Inequality Models of Disturbance

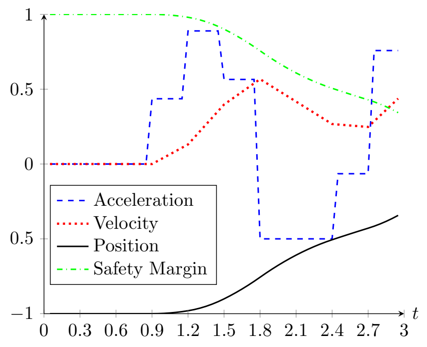
In this section we introduce differential inequality models as a technique to consider external disturbance, such as the influence of road conditions on braking. If the value of a variable v changes nondeterministically according to acceleration a, as in the previous examples, and some disturbance d, we can use differential inequality models of disturbance. For example, the differential inequality $v' \le ad$ models the effect of disturbance d on the acceleration a of our car, i. e., in the worst case the effective braking force may be reduced and the acceleration increased depending on a disturbance factor d.


$?p + \frac{v^2}{2B} + \left(\frac{A}{B}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v\right) \le S$
ctrl
plant
$a := *$
$? - B \le a \le A$
$?v = 0$
$a := 0$
$a := -B$
$c := 0$
$p' = v, v' = a, c' = 1$
$\& \; v \ge 0 \wedge c \le \varepsilon$
$*$

(a) State transition system



(b) Control decisions: nondeterministically chosen acceleration constrained with guard $-B \le a \le A$



(c) Sample trace: velocity and position per acceleration choice; distance to stop sign $|p - S|$ gives the remaining safety margin

Fig. 8: Time-triggered controller with nondeterministic assignment (Example 6)

Example 7 Stop sign controller with nondeterministic braking

$$\texttt{init} \rightarrow [(ctrl; plant)^*](\texttt{req}) \tag{52}$$

$$\texttt{init} \equiv p + \frac{v^2}{2b} \le S \wedge A > 0 \wedge b > 0 \wedge B \ge b \wedge v \ge 0 \wedge \varepsilon > 0 \tag{53}$$

$$ctrl \equiv (?\texttt{Safe}_\varepsilon; a := *; ? - B \le a \le A) \tag{54}$$

$$\cup\, (?v = 0; a := 0) \tag{55}$$

$$\cup\, (a := *; ? - B \le a \le -b) \tag{56}$$

$$\texttt{Safe}_\varepsilon \equiv p + \frac{v^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v\right) \le S \tag{57}$$

$$plant \equiv c := 0; p' = v, v' = a, c' = 1 \;\&\; v \ge 0 \wedge c \le \varepsilon \tag{58}$$

$$\texttt{req} \equiv p \le S \tag{59}$$

Example 8 Stop sign controller with braking disturbance

$$\texttt{init} \rightarrow [(ctrl; plant)^*](\texttt{req}) \tag{60}$$

$$\texttt{init} \equiv p + \frac{v^2}{2bd} \le S \wedge A > 0 \wedge b > 0 \wedge B \ge b \tag{61}$$

$$\wedge\, v \ge 0 \wedge \varepsilon > 0 \wedge d > 0$$

$$ctrl \equiv (?\texttt{Safe}_\varepsilon; a := *; ? - B \le a \le A) \tag{62}$$

$$\cup\, (?v = 0; a := 0) \tag{63}$$

$$\cup\, (a := *; ? - B \le a \le -b) \tag{64}$$

$$\texttt{Safe}_\varepsilon \equiv p + \frac{v^2}{2bd} + \left(\frac{A}{b}+1\right)\left(\frac{Ad}{2}\varepsilon^2 + \varepsilon v\right) \le S \tag{65}$$

$$plant \equiv c := 0;\; p' = v, v' \le ad, c' = 1 \;\&\; v \ge 0 \wedge c \le \varepsilon \tag{66}$$

$$\texttt{req} \equiv p \le S) \tag{67}$$

Example 8 introduces such a differential inequality model of disturbance on top of Example 7. The specific differential inequality $v' \le ad$ used in this example models that the effective braking force and the effective acceleration force are subject to disturbance d; the disturbance is negligible when the acceleration or braking force is small, but it grows with increasing force. This model avoids disturbance when the car does not accelerate ($a = 0$), which means that disturbance alone will not cause the car to move.

Example 8 uses the same loop of sequential execution of controller and plant as Example 7, cf. (60). We adapt the initial condition in formula (61) to reflect that disturbance affects the braking force of the car to $b(1+d)$, but does not exceed the braking force ($0 \le d < 1$). The controller itself remains the same as in Example 7, cf. (62)–(64). The main difference is in the controller's safety condition given in formula (65), which considers the fact that disturbance may reduce the effective braking force of the car and increase its acceleration (e. g., when driving downhill). Finally, the plant (66) replaces the differential equation of Example 7 with the differential inequality model.

5.10 System Energy and Invariants

An important challenge when proving properties of hybrid systems is proving properties of the continuous components, i. e., the part of the system specified as a system of differential equations.

One strategy to handle differential equations is to solve them and continue the proof with their solutions. This can be an effective approach in many cases, especially when the differential equations have polynomial solutions. In this case, quantifier elimination can be used to reason about the resulting polynomials. Many differential equations (even simple ones), however, do not have polynomial solutions. For example, the differential equation $x' = -x$ for exponential decay has solution $x(t) = e^{-t}$, and the equations of motion of a simple harmonic oscillator involve trigonometric functions. These functions cannot be handled directly by quantifier elimination, and it can be very challenging to prove interesting properties about them. In general, the solutions of differential equations are more complex than the equations themselves, and indeed many differential equations do not have closed-form solutions.

Instead, KeYmaera provides proof rules to reason about differential equations without solving them. These rules fit neatly with standard techniques in science and engineering for working with differential equations. An important and useful idea from physics is that an isolated system will conserve energy over time (in the absence of relativistic effects). If there are "loss" effects due to things like friction or air resistance, the energy of the system will decrease over time, until the system runs out of energy and settles into an "equilibrium".

In the example below, a car starts to cruise without accelerating, but it experiences wind resistance, which slows it down. We use an energy argument to show that its velocity cannot increase past its initial velocity. Let m be the mass of the car, and suppose it starts with velocity $v = v_0$. The initial kinetic energy of the car is then $E_0 = \frac{1}{2}mv_0^2$. At any future time t, the kinetic energy of the car will be given by $E_t = \frac{1}{2}mv_t^2$. The kinetic energy may not increase, so the condition $E_t \le E_0$ ensures that the car's velocity cannot increase. After arithmetic cancellations, the energy condition is equivalent to $v_t^2 \le v_0^2$, which we use to prove the example below. In KeYmaera, the energy condition is entered through the "differential invariant" rule.

The idea of conservation of energy can be generalized to more complex scenarios. We observe that the following characteristics of energy were important to the applications described in the section above:

Example 9 Car velocity cannot increase, proved via an energy argument

$$\texttt{init} \to [\mathit{plant}]\,(\texttt{req}) \tag{68}$$
$$\texttt{init} \equiv v = v_0 \land v_0 \ge 0 \land \delta > 0 \tag{69}$$
$$\mathit{plant} \equiv p' = v, v' = -\delta v \tag{70}$$
$$\texttt{req} \equiv v \le v_0 \tag{71}$$

1. A system cannot have negative energy. Energy must be either positive, or zero if it is at an equilibrium position.
2. The energy of the system cannot increase over time. If there are dissipation effects (such as fricition or air resistance), energy must decrease, but will otherwise remain constant.

In what follows, we will use the term *generalized energy function* to denote any function that is nonnegative over all system states and never increases as the system evolves. A related notion in the engineering literature is that of Lyapunov functions, which have a rich theory and important applications [27] [54] [23].

The following example, adapted from Example 5 of [16], is a system with switching that is proven using a generalized energy function.

The equations below represent a model of a car with automatic transmission controlled by a PI controller to track a constant velocity v_{ref}. The difference between the car's velocity and the desired velocity is $\Delta v = v - v_{ref}$. The state of the integrator of the PI controller is $\Delta T_I = T_I - 0$, since when the car is cruising at the desired velocity, we would like the integrator state to be zero. The mass of the car is represented by m, and G_p is one of four possible gears the car can be in, $G_{p(t)} \in \{G_1, G_2, G_3, G_4\}$. The angular velocity of the engine, ω, can be computed from the car's velocity from knowledge of the current gear by $\omega = G_{p(t)}v$. The proportional gain of the controller is $K_{p(t)}$, and it changes when the car changes gears, so that the car handles smoothly, $K_{p(t)} \in \{K_1, K_2, K_3, K_4\}$. T_R is a constant gain chosen so that the controller converges quickly to the desired velocity, while avoiding over corrections that could cause undesired oscillations. The car dynamics are given by

$$\dot{\Delta v} = -(G_{p(t)}K_{p(t)}/m) \cdot \Delta v - (G_{p(t)}/m) \cdot \Delta T_I \tag{72}$$
$$\dot{\Delta T_l} = (K_{p(t)}/T_R) \cdot \Delta v \tag{73}$$

The automatic transmission switches between gears according to the following rule. G_i denotes the current gear of the car, ω_{high} is an engine velocity at which the transmission decides to switch to a higher gear, and ω_{low} is an engine velocity at which the transmission decides to switch to a lower gear.

$$G_{p(t^+)} = \begin{cases} G_{i+1} & \text{if } i \ne 4 \text{ and } v \ge \frac{1}{G_i}\omega_{high} \\ G_{i-1} & \text{if } i \ne 1 \text{ and } v \le \frac{1}{G_i}\omega_{low} \\ G_i & \text{otherwise} \end{cases} \tag{74}$$

The verification task will be to ensure that when the cruise control system is engaged within some interval around the desired velocity, the engine speed ω will not exceed a certain *redline speed*, ω_{max}. If the engine runs above the redline speed, damage can result.

A hybrid program for this example is shown in .

Example 10 Tracking a constant velocity with gear shifting

$$\texttt{init} \to [(transmission; dynamics)^*]\,(\texttt{req}) \quad (75)$$
$$\texttt{init} \equiv\ \Delta T_I = 0\ \wedge\ -\Delta v_{engage} \leq \Delta v \quad (76)$$
$$\wedge\ v \leq \Delta v_{engage}\ \wedge\ G_p = G_1 \quad (77)$$
$$ctrl \equiv\ v := \Delta v + v_{ref}; \quad (78)$$
$$(?G_p \neq G_4; ?v \geq \frac{1}{G_p}\omega_{high}; \quad (79)$$
$$((?G_p = G_1; G_p := G_2) \cup (?G_p = G_2; G_p := G_3) \quad (80)$$
$$\cup\, (?G_p = G_3; G_p := G_4))) \quad (81)$$
$$\cup\, (?G_p \neq G_1; ?v \leq (\frac{1}{G_p})\omega_{low}; \quad (82)$$
$$((?G_p = G_4; G_p := G_3) \cup (?G_p = G_3; G_p := G_2) \quad (83)$$
$$\cup\, (?G_p = G_2; G_p := G_1))); K_p := \frac{187.5}{G_p} \quad (84)$$
$$plant \equiv\ \{\Delta v' = (-G_p\frac{K_p}{m})\Delta v - \frac{G_p}{m}\Delta T_I, \quad (85)$$
$$\Delta T_I' = \frac{K_p}{T_R}\Delta v \quad (86)$$
$$\&((\Delta v + v_{ref} \leq \frac{1}{G_p}\omega_{high} \vee G_p = G_4) \quad (87)$$
$$\wedge\ (\Delta v + v_{ref} \geq \frac{1}{G_p}\omega_{low} \vee G_p = G_1))\} \quad (88)$$
$$\texttt{req} \equiv\ G_p(\Delta_v + v_{ref}) \leq \omega_{max} \quad (89)$$

In [16], the gains K_1, K_2, K_3, K_4 and the constant T_R are chosen for specific gear ratios, car mass, and reference velocity. The values of the gear ratios are $G_1 = 50$, $G_2 = 32$, $G_3 = 20$, and $G_4 = 14$—note that they are unit-free because they are ratios of gear sizes. The proportional gains are chosen so that the product $K_{p(t)}G_{p(t)} = 187.5$, to ensure that the car handles smoothly across gear switches. The reference velocity is chosen $v_{ref} = 30m/s(\approx 67mph)$, and the mas of the car is assumed to be $m = 1500kg$.

For the parameter values specified above, the authors in [16] find that the following is a generalized energy function of the system.

$$40.822(\Delta T_I)^2 + 144.524\Delta T_I\Delta v + 255.589(\Delta v)^2 \quad (90)$$

For the verification task, we use the following inductive invariant

$$40.822(\Delta T_I)^2 + 144.524\Delta T_I\Delta v + 255.589(\Delta v)^2 \leq 58000 \quad (91)$$
$$\wedge\ (G_{p(t)}(\Delta v + v_{ref}) > omega_{high} -> G_{p(t)} = G_4 \quad (92)$$

This invariant specifies that the generalized system energy, as represented by the generalized energy function, never exceeds 5800. This number is chosen such that the initial energy of the system is less than it. The additional condition states that if the angular velocity is larger than ω_{high}, then the car is currently in gear G_4. This is done to ensure that the engine speed ω is computed with the correct gear when checking that the engine speed is less than the redline speed ω_{max}.

In general, it is a difficult problem to find generalized energy functions for arbitrary systems. Efficient algorithms exist, however, for the case of linear systems [10].

An additional complication that may exist is that the model may not admit the existence of a quantity that is always decreasing—in particular, some state may be always increasing. For example, a car that continuously accelerates will have unbounded position and velocity. These types of systems are called "unstable". A good treatment of of stability theory can be found in [10], [27], [54].

5.11 Differential Invariants, Differential Cuts, and Differential Induction

Differential invariants define an induction principle for differential equations: instead of solving a differential equation system, we show that some expression F is true throughout the dynamics specified in the differential equation system (i. e., is a differential invariant for the differential equation system). A simple proof rule for differential invariants is *differential weakening*. Differential weakening replaces a differential equation system with a nondeterministic assignment subject to the evolution domain constraints. This proof rule, although being obviously sound since the differential equation system cannot leave the evolution domain by definition, is only useful when the evolution domain constraints are very informative.

We can use *differential cuts* to make the evolution domain sufficiently informative. By successively applying differential cuts we can increasingly strengthen the evolution domain until eventually the differential equation can be resolved by differential weakening. Since additional evolution domain constraints actually change the system dynamics, we have to prove that these additional constraints are themselves differential invariants. We can do this by differential induction. Differential induction defines induction for differential equations much in the sense of induction for discrete loops. Intuitively, differential induction shows that a formula is getting "more true" when following the differential equation system. More in-depth information about differential weakening, cuts, and induction can be found in [42].

5.12 Example 8: Car controller for non-linear dynamics

In the previous examples we have reduced the stop-sign controller to a one-dimensional problem. In the next step, we introduce non-linear dynamics to model the behavior of a car on a two-dimensional but still straight lane. As a first safety property, we want to prove that the car controller manages to stay within the bounds of the lane when it drives a sequence of circular arcs as trajectory.

In a two-dimensional world, the car has a vector state variable describing its current position $p = (p_x, p_y)$ and scalar r that describes the radius of the car's current trajectory. The car makes a sharp turn if the radius r is small. If the radius becomes larger then the curve becomes increasingly straightened; the car drives a perfect straight line if the radius is infinite ($r = \infty$). We can specify the orientation of the car w.r.t. the center of the curve $o = (o_x = \cos\theta, o_y = \sin\theta)$. Its angular velocity is then a scalar $\omega = \theta'$. To avoid undecidable arithmetic, the orientation vector o encodes sine and cosine functions in the dynamics [32, 39]. A car follows unidirectional motion along its orientation, that is the orientation vector o gives the orientation and reduces the velocity vector to a scalar v as in the previous examples. This distinction not only represents the control choices of a car with separated steering and acceleration, it also reduces proof complexity, since we can introduce separated differential invariants. The translational and rotational velocities are linked w.r.t. the rigid body planar motion by the formula $r\omega = v$.

In our model, steering changes the rotational velocity and with it the radius of the trajectory. Note, that the constraints on the radius and on the direction vector select the drive variant [9] of our car:

Omnidirectional drive modeled with positive radius $r \geq 0$ and instantaneous orientation jumps $r := (p - p_c)^\perp$ around a trajectory center p_c.

Differential drive modeled with positive radius $r \geq 0$ and disallowing instantaneous orientation jumps.

Ackermann drive, Dubin's car modeled with strictly positive radius $r > 0$ and disallowing instantaneous orientation jumps.

The continuous dynamics of the car can then be described by the differential equation system of ideal-world dynamics of the planar rigid body motion, as in $p_x' = vo_x, p_y' = vo_y, v' = a, (r\omega)' = a, o' = \omega o^\perp$. The condition $(r\omega)' = a$ encodes the rigid body planar motion $r\omega = v$ and $o' = \omega o^\perp$ adjusts the orientation towards the trajectory center according to the rotational velocity ω with ortho-normal vector $o^\perp$.

In Ex. 11 the car is initially stopped at the center of the lane. As in the previous models, the car has three control choices (95)–(99): (i) it may choose a new trajectory when it is safe to do so, (ii) it may stay stopped, and (iii) it may brake on its current trajectory to stay on the lane.

Example 11 Car controller with non-linear dynamics

$$\texttt{init} \rightarrow [(\mathit{ctrl}; \mathit{plant})^*]\,(\texttt{req}) \tag{93}$$

$$\texttt{init} \equiv l_w > 0 \land p_y = l_y \land v = 0 \land r > 0 \land \|o\| = 1 \tag{94}$$
$$\land A > 0 \land b > 0 \land B \geq b \land \varepsilon > 0$$

$$\mathit{ctrl} \equiv (?\texttt{Safe}_\varepsilon; a := *; ?-B \leq a \leq A); \tag{95}$$
$$\omega := *; \tag{96}$$
$$r := *; ?r > 0 \land |\omega| r = v \tag{97}$$
$$\cup\, (?v = 0; a := 0; \omega := 0) \tag{98}$$
$$\cup\, (a := *; ?-B \leq a \leq -b) \tag{99}$$

$$\texttt{Safe}_\varepsilon \equiv |p_y - l_y| + \frac{v^2}{2b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v\right) < l_w \tag{100}$$

$$\mathit{plant} \equiv c := 0; \tag{101}$$
$$p' = vo, v' = a, o' = \omega o^\perp, \omega' = \frac{a}{r}, c' = 1 \tag{102}$$
$$\&\ v \geq 0 \land c \leq \varepsilon \tag{103}$$
$$\texttt{req} \equiv |p_y - l_y| < l_w \tag{104}$$

The proof of Ex. 11 uses differential cuts, differential weakening and differential induction.

6 Advanced Modeling Concepts and Pitfalls

6.1 PID controllers

Smoother control is possible with analog proportional-integral-derivative (PID) controllers. In addition to the current error, these use the accumulated error, i. e., the integral over the error, and the slope of the error function, i. e., its derivate, to steer the system. These more complex controllers can be presented as differential equations and used for the verification in KeYmaera as well. We refer the reader to [4, 46] for details on this encoding.

6.2 Hybrid Time

A common assumption in hybrid systems, as already mentioned, is that discrete actions do not consume time. Because discrete actions are assumed not to consume time, multiple discrete actions can occur at the same real point in time. To reflect this, we augment the time domain by a natural number that counts the number of discrete actions that have happened already. That is, a hybrid point in time is a pair $(r, n) \in \mathbb{R}_{\geq 0} \times \mathbb{N}$. For each real-valued point in time r there is a discrete time axis that reflects the order of discrete actions. Hence the time model for hybrid systems, called *hybrid time*, is given by $\mathbb{R}_{\geq 0} \times \mathbb{N}$.

6.3 Disjoint Tests and Evolution Domains

When combining choices and test it is important to make sure that the model does not get blocked in an unnatural way. For example, the program

$$(?v < 3; v' = A) \cup (?v > 5; v' = -B)$$

cannot evolve if v is between 3 and 5. Therefore, it is good modeling practice to have at least one branch for each case. Evolution domain constraints also need to be designed with care. For example, the HP

$$((v' = -B \ \& \ v \geq 0) \cup (v' = -b \ \& \ v < 0))^*$$

has disjoint evolution domain constraints. When $v = 0$, the system cannot switch to the second choice, because its evolution constraint $v < 0$ is not satisfied for the initial state.

6.4 Non-Existence of Systems

Tests outside nondeterministic choices must be used carefully, since they potentially entail non-existence of the modeled system. For example, the $\mathsf{d}\mathcal{L}$ statement

$$v > 0 \rightarrow [(v' = A)\,;\ ?v = 0]\,(v = 0)$$

results in an empty set of executions, since none of the values of v will satisfy the test $?v = 0$. Thus, the property $v = 0$ can be proven trivially. Such issues can be detected by liveness proofs using the diamond modality $\langle\alpha\rangle$: the $\mathsf{d}\mathcal{L}$ statement $v > 0 \rightarrow \langle(v' = A)\,;\ ?v = 0\rangle\,(v = 0)$ is only true, if at least one run satisfies the requirement.

6.5 Safety Throughout vs. Safety Finally

The evolution of a differential equation system is allowed to nondeterministally stop at any time (even zero) before the evolution domain becomes false. Thus, $\mathsf{d}\mathcal{L}$ properties of the form $[(x' = \theta \ \& \ H)]\,\phi$ usually verify safety *throughout* system execution. A subsequent test, as in $[(x' = \theta \ \& \ c \leq \varepsilon)\,;\ ?c = \varepsilon]\,\phi$, however, means that all traces that end before the clock reached its maximum time will not be considered during the proof. Thus, we only verify that the requirement ϕ will be true at the end of the evolution, which is weaker than safety throughout.

7 Outlook

In this tutorial, we presented the basic proof techniques provided by KeYmaera to verify parametric hybrid systems. In this tutorial we focused on safety properties. However, KeYmaera is able to show liveness properties as well. These can be expressed using the diamond modality $\langle\cdot\rangle$. Where in the proofs of safety properties invariants were necessary to prove properties of loops now *variants* are required [38]. A variant can be seen as a formula encoding progress (cf. a termination function in discrete program verification). KeYmaera can also be used to prove general formulas of $\mathsf{d}\mathcal{L}$ with arbitrary nestings of quantifiers and modalities. Due to space constraints, we refer to previous work [38, 40, 46] for such examples. Extending these ideas even further, KeYmaera can be used to reason about hybrid games by means of *differential dynamic game logic* [48]. Here, the number of interactions between box and diamond modalities is not fixed a priori but instead statements about arbitrary alternations of these can be made. Therefore, this extension can be used to reason about the existence of a controller for examples rather than the correct functioning of a specific one.

In addition KeYmaera can be used to reason about distributed hybrid systems with an a priori unknown number of interacting agents. For this hybrid programs can be extended to quantified hybrid programs and quantifiers over additional countable domains may be added. The resutling logic is called *quantified differential dynamic logic* [41]. This allows, among other things, to make reasoning about the number of cars to consider during a lane change maneuver explicit during the proof instead of having to apply some argument why it is sufficient to consider only a certain limited number of cars.

Recently, there has been significant work on the automatic generation of promising differential invariant candidates [22].

Furthermore, we have been investigating proof-aware refactorings [34]. There, the idea is to perform transformations on the hybrid programs in a structured way in order to minimize the effort when reproving properties about these programs. This specifically supports iterative development of hybrid systems.

References

1. Alur, R.: Formal verification of hybrid systems. In: Chakraborty, S., Jerraya, A., Baruah, S.K., Fischmeister, S. (eds.) EMSOFT. pp. 273–278. ACM (2011)
2. 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)
3. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. Theor. Comput. Sci. 318(3), 297–322 (2004)
4. Aréchiga, N., Loos, S.M., Platzer, A., Krogh, B.H.: Using theorem provers to guarantee closed-loop system properties. In: Tilbury, D. (ed.) ACC (2012)
5. Asarin, E., Dang, T., Maler, O.: The d/dt tool for verification of hybrid systems. In: Brinksma, E., Larsen, K.G. (eds.) CAV. Lecture Notes in

Computer Science, vol. 2404, pp. 365–370. Springer (2002)

6. Beckert, B., Hähnle, R., Schmitt, P.H. (eds.): Verification of Object-Oriented Software: The KeY Approach, LNCS, vol. 4334. Springer (2007)
7. Behrmann, G., Fehnker, A.: Efficient guiding towards cost-optimality in uppaal. In: Margaria, T., Yi, W. (eds.) TACAS. Lecture Notes in Computer Science, vol. 2031, pp. 174–188. Springer (2001)
8. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.W.: Minimum-cost reachability for priced timed automata. In: Benedetto, M.D.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC. Lecture Notes in Computer Science, vol. 2034, pp. 147–161. Springer (2001)
9. Bräunl, T.: Driving robots. In: Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems, pp. 97–111. Springer (2006)
10. Chen, C.T.: Linear System Theory and Design. Oxford University Press, third edition edn. (1999)
11. Chen, X., brahm, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification, Lecture Notes in Computer Science, vol. 8044, pp. 258–263. Springer Berlin Heidelberg (2013)
12. Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, MA, USA (1999)
13. Damm, W., Dierks, H., Disch, S., Hagemann, W., Pigorsch, F., Scholl, C., Waldmann, U., Wirtz, B.: Exact and fully symbolic verification of linear hybrid automata with large discrete state spaces. Sci. Comput. Program. 77(10-11), 1122–1150 (2012)
14. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. Journal of Symbolic Computation 5(1/2), 29–35 (1988)
15. Davoren, J.M., Nerode, A.: Logics for hybrid systems. IEEE 88(7), 985–1010 (2000)
16. DeCarlo, R.A., Branicky, M.S., Petersson, S., Lennartson, B.: Perspectives and results on the stability and stabilizability of hybrid systems. In: Proceedings of the IEEE. vol. 88, pp. 1069—1082. IEEE (July 2000)
17. Deshpande, A., Göllü, A., Varaiya, P.: SHIFT: A formalism and a programming language for dynamic networks of hybrid automata. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S. (eds.) Hybrid Systems. LNCS, vol. 1273, pp. 113–133. Springer (1996)
18. Eggers, A., Ramdani, N., Nedialkov, N., Fränzle, M.: Improving sat modulo ode for hybrid systems analysis by combining different enclosure methods. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM. Lecture Notes in Computer Science, vol. 7041, pp. 172–187. Springer (2011)
19. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. JSAT 1(3-4), 209–236 (2007)
20. Frehse, G.: Phaver: algorithmic verification of hybrid systems past hytech. STTT 10(3), 263–279 (2008)
21. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV. Lecture Notes in Computer Science, vol. 6806, pp. 379–395. Springer (2011)
22. Ghorbal, K., Platzer, A.: Characterizing algebraic invariants by differential radical invariants. In: Ábrahám, E., Havelund, K. (eds.) TACAS. Lecture Notes in Computer Science, vol. 8413, pp. 279–294. Springer (2014)
23. Haddad, W.M., Chellaboina, V.: Nonlinear DyDynamic Systems and Control: A Lyapunov-Based Approach. Princeton University Press (2008)
24. Henzinger, T.A.: The theory of hybrid automata. In: LICS. pp. 278–292. IEEE Computer Society, Los Alamitos (1996)
25. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HYTECH: A model checker for hybrid systems. STTT 1(1-2), 110–122 (1997)
26. Ioannu, P., Xu, Z., Eckert, S., Clemons, D., Sieja, T.: Intelligent cruise control: Theory and experiment. In: CDC. pp. 1885–1890 (1993)
27. Khalil, H.K.: Nonlinear Systems. Prentice Hall, third edition edn. (2001)
28. Kouskoulas, Y., Renshaw, D.W., Platzer, A., Kazanzides, P.: Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In: Belta, C., Ivancic, F. (eds.) Hybrid Systems: Computation and Control (part of CPS Week 2013), HSCC'13, Philadelphia, PA, USA, April 8-13, 2013. pp. 263–272. ACM (2013)
29. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. In Springer International Journal of Software Tools for Technology Transfer, 1(1+2), 134–152 (May 1997)
30. Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: Hybrid, distributed, and now formally verified. In: Butler, M., Schulte, W. (eds.) FM. LNCS, vol. 6664, pp. 42–56. Springer (2011)
31. Loup, U., Scheibler, K., Corzilius, F., Ábrahám, E., Becker, B.: A symbiosis of interval constraint propagation and cylindrical algebraic decomposition. In: Bonacina, M.P. (ed.) CADE. Lecture Notes in Computer Science, vol. 7898, pp. 193–207. Springer (2013)
32. Mitsch, S., Ghorbal, K., Platzer, A.: On provably safe obstacle avoidance for autonomous robotic ground vehicles. In: Robotics: Science and Systems (2013)
33. Mitsch, S., Loos, S.M., Platzer, A.: Towards formal verification of freeway traffic control. In: Lu, C. (ed.) ICCPS (2012)

34. Mitsch, S., Quesel, J.D., Platzer, A.: Refactoring, refinement, and reasoning - a logical characterization for hybrid systems. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM. Lecture Notes in Computer Science, vol. 8442, pp. 481–496. Springer (2014)
35. Olderog, E.R., Dierks, H.: Real-time systems - formal specification and automatic verification. Cambridge University Press (2008)
36. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Hybrid systems: from verification to falsification by combining motion planning and discrete search. Form. Methods Syst. Des. 34(2), 157–182 (2009)
37. Platzer, A.: Differential dynamic logic for verifying parametric hybrid systems. In: Olivetti, N. (ed.) TABLEAUX. LNCS, vol. 4548, pp. 216–232. Springer (2007)
38. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. 41(2), 143–189 (2008)
39. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. 20(1), 309–352 (2010)
40. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010)
41. Platzer, A.: Quantified differential dynamic logic for distributed hybrid systems. In: Dawar, A., Veith, H. (eds.) Computer Science Logic 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010, Proceedings. LNCS, vol. 6247, pp. 469–483. Springer (2010)
42. Platzer, A.: The structure of differential invariants and differential cut elimination. Logical Methods in Computer Science 8(4), 1–38 (2012)
43. Platzer, A., Clarke, E.M.: Computing differential invariants of hybrid systems as fixedpoints. In: Gupta, A., Malik, S. (eds.) CAV. LNCS, vol. 5123, pp. 176–189. Springer (2008)
44. Platzer, A., Clarke, E.M.: Formal verification of curved flight collision avoidance maneuvers: A case study. In: Cavalcanti, A., Dams, D. (eds.) FM. LNCS, vol. 5850, pp. 547–562. Springer (2009)
45. Platzer, A., Quesel, J.D.: KeYmaera: A hybrid theorem prover for hybrid systems. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR. LNCS, vol. 5195, pp. 171–178. Springer (2008)
46. Platzer, A., Quesel, J.D.: European Train Control System: A case study in formal verification. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM. LNCS, vol. 5885, pp. 246–265. Springer (2009)
47. Pratt, V.R.: Semantical considerations on floyd-hoare logic. In: FOCS. pp. 109–121. IEEE Computer Society (1976)
48. Quesel, J.D., Platzer, A.: Playing hybrid games with KeYmaera. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning, Sixth International Joint Conference, IJCAR 2012, Manchester, UK, Proceedings. LNCS, vol. 7364, pp. 439–453. Springer (June 2012)
49. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation-based abstraction refinement. ACM Trans. Embed. Comput. Syst. 6(1) (Feb 2007)
50. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. University of California Press, Berkeley, 2nd edn. (1951)
51. Teige, T., Eggers, A., Fränzle, M.: Constraint-based analysis of concurrent probabilistic hybrid systems: An application to networked automation systems. Nonlinear Analysis: Hybrid Systems 5(2), 343 – 366 (2011), special Issue related to {IFAC} Conference on Analysis and Design of Hybrid Systems (ADHS09) {IFAC} ADHS09
52. Tomlin, C., Pappas, G.J., Sastry, S.: Conflict resolution for air traffic management: a study in multi-agent hybrid systems. IEEE T. Automat. Contr. 43(4) (1998)
53. Umeno, S., Lynch, N.A.: Safety verification of an aircraft landing protocol: A refinement approach. In: Bemporad, A., Bicchi, A., Buttazzo, G.C. (eds.) HSCC. LNCS, vol. 4416, pp. 557–572. Springer (2007)
54. Vidyasagar, M.: Nonlinear Systems Analysis. Classics in Applied Mathematics, SIAM: Society for Industrial and Applied Mathematics, second edition edn. (2002)