KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description)*

André Platzer and Jan-David Quesel

University of Oldenburg, Department of Computing Science, Germany
{platzer,quesel}@informatik.uni-oldenburg.de

Abstract. KeYmaera is a hybrid verification tool for hybrid systems that combines deductive, real algebraic, and computer algebraic prover technologies. It is an automated and interactive theorem prover for a natural specification and verification logic for hybrid systems. KeYmaera supports *differential dynamic logic*, which is a real-valued first-order dynamic logic for hybrid programs, a program notation for hybrid automata. For automating the verification process, KeYmaera implements a generalized free-variable sequent calculus and automatic proof strategies that decompose the hybrid system specification symbolically. To overcome the complexity of real arithmetic, we integrate real quantifier elimination following an iterative background closure strategy. Our tool is particularly suitable for verifying parametric hybrid systems and has been used successfully for verifying collision avoidance in case studies from train control and air traffic management.

Keywords: dynamic logic, automated theorem proving, decision procedures, computer algebra, verification of hybrid systems.

1 Introduction

Formal verification becomes more and more important as computerized control systems in safety-critical systems grow significantly in complexity. In many applications, system states, like positions of vehicles, change continuously according to differential equations and are affected by discrete controller decisions. Hybrid systems [7] are a mathematical model for such systems with interacting discrete and continuous dynamics. Model checkers [7,5] verify correctness properties by exploring the state space exhaustively, which provides a good mechanism to find bugs or concrete counterexamples for specifications. Unfortunately, the state space of hybrid systems is uncountably infinite and cannot be partitioned into finitely many relevant regions for deciding reachability [7].

As deductive methods [4,2,8,1] are known for being capable of dealing with infinite domains, we choose a proof-based approach. We present the verification tool KeYmaera that uses a combination of automated theorem proving (for

* This research was partially supported by the German Research Council (DFG) in the Transregional Collaborative Research Center SFB/TR 14 AVACS.

A. Armando, P. Baumgartner, and G. Dowek (Eds.): IJCAR 2008, LNCS 5195, pp. 171–178, 2008.
© Springer-Verlag Berlin Heidelberg 2008

symbolically decomposing and executing system models), real quantifier elimination [3] (for handling the arithmetic of hybrid systems), and symbolic computations in computer algebra systems (for handling differential equations of continuous evolutions). As the central concept, our tool implements an axiomatization of the transition behavior of hybrid systems in the form of the sequent calculus for the differential dynamic logic $\mathsf{d}\mathcal{L}$ [12,13]. KeYmaera provides proof strategies that automate the verification process to a large extent. In several realistic applications, the proof construction is even completely automatic, e.g., for proving collision avoidance of trains or aircraft.

In addition, theorem proving in combination with the equivalence-transformations of quantifier elimination enables us to verify highly parametrized hybrid systems and even to discover safety-critical parameter constraints. The traceability gained by the deductive symbolic system decomposition enables the user to use his system knowledge for projecting the obtained constraints on the free parameters of the system to the relevant cases. Since the underlying logic $\mathsf{d}\mathcal{L}$ and its compositional proof calculus are natural and intuitive, even computationally intractable problems can be verified with selective user guidance in KeYmaera.

In this paper, we describe the theorem prover KeYmaera and the various techniques that it combines for verifying hybrid systems. KeYmaera consists of ca. 186,000 lines of Java code and 141 optimized proof rules, including rules for symbolic decomposition, propositional logic, first-order logic, and simplification.

2 KeYmaera Verification Tool for Hybrid Systems

KeYmaera is a deductive verification tool for hybrid systems. We have implemented KeYmaera as a combination of the deductive theorem prover KeY [2] with the computer algebra system Mathematica, see Fig. 1. KeY is an interactive theorem prover with a user-friendly graphical interface for proving correctness properties of Java programs. We generalize KeY from discrete systems to hybrid systems by adding support for the differential dynamic logic $\mathsf{d}\mathcal{L}$ [12,13], which is a dynamic logic [6] that provides a natural way to formalize properties of the states reachable by following the dynamics of hybrid systems. With this, KeYmaera can prove correctness, safety, controllability, reactivity, and liveness properties of hybrid systems.

In discrete KeY, rule applications are comparably fast, but in KeYmaera, proof rules that use decision procedures for real arithmetic can require a substantial amount of time to produce a result. To overcome this, we have implemented

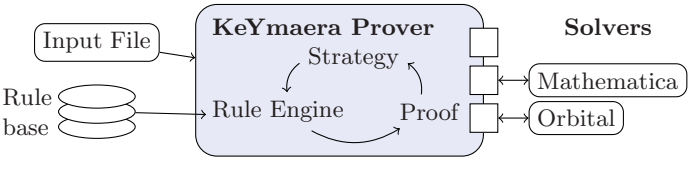


Fig. 1. Architecture and plug-in structure of the KeYmaera Prover

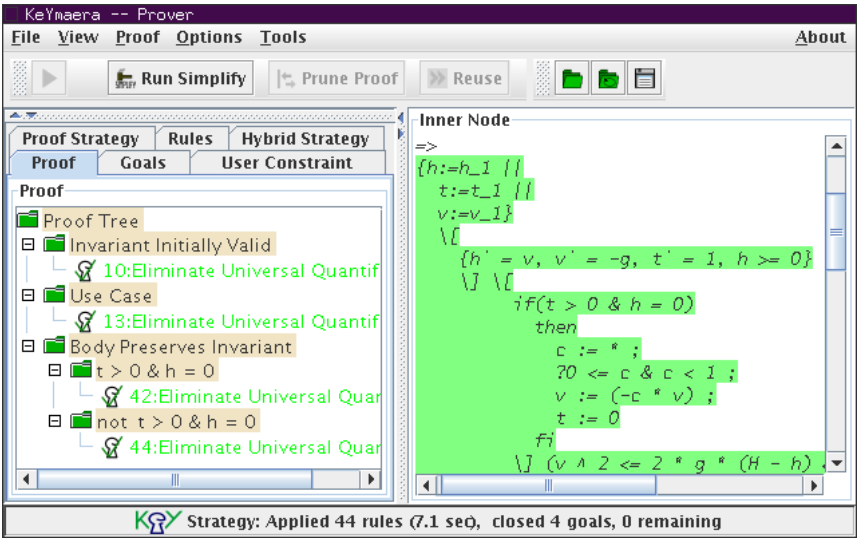


Fig. 2. Screenshot of the KeYmaera user interface

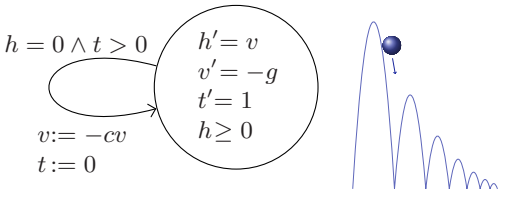
new automatic proof strategies for the hybrid case that navigate among computationally expensive rule applications.

We have implemented a plug-in architecture for integrating multiple implementations of decision procedures for the different fields of arithmetic handling, cf. Fig. 1. We integrate arithmetical simplification and real quantifier elimination support by interfacing Mathematica. Symbolic solutions of differential equations, which can be used for handling continuous dynamics, are obtained either from Mathematica or Orbital, a math library for Java developed by the first author.

3 Hybrid Systems, Hybrid Automata, and Hybrid Programs

Hybrid systems [7] are mathematical models for systems with interacting continuous and discrete state transitions. The standard description language for specifying the operational behavior of hybrid systems is that of *hybrid automata* [7]. A hybrid automaton is a finite automaton with real variables that evolve continuously in the automaton locations as specified by differential equations. Additionally, state variables can occur in transition guards and transitions can change the values of the variables. The graph notation of hybrid automata is not compositional, e.g., it is not sufficient to prove properties separately for each location to infer a global system property, as the transition effects have to be taken into account.

Instead, we use a program notation for hybrid automata which is designed to have a compositional semantics that we exploit for verifying systems by symbolic decomposition. *Hybrid programs* [12,13] are an extension of discrete regular programs [6] by continuous evolutions. An overview of the syntax and



$$\texttt{Ball} \equiv \Big((h' = v, v' = -g, t' = 1, h \geq 0); \text{ if } (h = 0 \wedge t > 0) \text{ then } c := *; ?0 \leq c < 1; \quad // \textit{ add-on } v := -cv; t := 0 \text{ fi} \Big)^*$$

Fig. 3. Hybrid automaton of a bouncing ball and corresponding hybrid program

informal semantics of hybrid programs is given in Tab. 1 (where F is a formula of first-order real arithmetic). Hybrid automata can be embedded into hybrid programs [13].

Example 1. Consider the well-known bouncing ball example. A ball falls from height h and bounces back from the ground ($h = 0$) after an elastic deformation. The current speed of the ball is denoted by v, and t is a clock measuring the falling time. We assume an arbitrary positive gravity force g and that the ball loses energy according to a damping factor $0 \leq c < 1$. Fig. 3 depicts the hybrid automaton, an illustration of the system dynamics, and the representation of the system as a hybrid program. However, the automaton still enforces infinite bouncing. In reality, the ball can come to a standstill when its remaining kinetic energy is insufficient. To model this phenomenon without the need to have a precise physical model for all involved forces, we allow for the damping factor to change at each bounce. Line 4 of the hybrid program in Fig. 3 represents a corresponding uncountably infinite nondeterministic choice for c, which is beyond the modelling capabilities of hybrid automata.

4 Syntax and Semantics of Differential Dynamic Logic

For characterizing states of hybrid systems, the foundation of the specification and verification logic of KeYmaera is first-order logic over real arithmetic. For

Table 1. Statements of hybrid programs

Statement	Effect
$\alpha;\ \beta$	sequential composition, first performs α and then β afterwards
$\alpha \cup \beta$	nondeterministic choice, following either α or β
α^*	nondeterministic repetition, repeating α $n \geq 0$ times
$x := \theta$	discrete assignment of the value of term θ to variable x (jump)
$x := *$	nondeterministic assignment of an arbitrary real number to x
$\big(x_1' = \theta_1, \ldots, x_n' = \theta_n, F\big)$	continuous evolution of x_i along differential equation system $x_i' = \theta_i$, restricted to maximum domain or invariant region F
$?F$	check if formula F holds at current state, abort otherwise
if(F) then α	perform α if F holds, do nothing otherwise
if(F) then α else β	perform α if F holds, perform β otherwise

expressing correctness statements about hybrid systems, this foundation is extended with parametrized modal operators $[\alpha]$ and $\langle\alpha\rangle$, for each hybrid program α. The resulting specification and verification logic is called *differential dynamic logic* $\mathsf{d}\mathcal{L}$ [12,13].

As is typical in dynamic logic, $\mathsf{d}\mathcal{L}$ integrates operational system models and formulas within a single language. The $\mathsf{d}\mathcal{L}$ formulas are generated by the following EBNF grammar (where $\sim \in \{<, \leq, =, \geq, >\}$ and θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals):

$$\phi ::= \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \forall x \phi \mid \exists x \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

The modal operators refer to states reachable by the hybrid program α and can be placed in front of any formula. Formula $[\alpha]\phi$ expresses that all states reachable by the hybrid program α satisfy formula ϕ (safety). Formula $\langle\alpha\rangle\phi$ expresses that there is a state reachable by the hybrid program α that satisfies formula ϕ.

Example 2. The ball looses energy at every bounce, thus the ball never bounces higher than the initial height. This can be expressed by the safety property $0 \leq h \leq H$, where H denotes the initial energy level, i.e., the initial height if $v = 0$. As a simple specification, which we generalize to the general parametric case later on, we can verify the following property using KeYmaera:

$$(h = H \wedge v = 0 \wedge 0 \leq H \leq 4 \wedge 0 < g \leq 2) \rightarrow [\texttt{Ball}]\,(0 \leq h \leq H) \qquad (1)$$

This specification follows the pattern of Hoare-triples. It expresses that the bouncing ball, when started in initial state $h = H$ etc. always respects $0 \leq h \leq H$.

The semantics of $\mathsf{d}\mathcal{L}$ is a Kripke semantics [12,13] where states correspond to states of the hybrid system, i.e., assignments of real values to all system variables.

5 Verification by Symbolic Decomposition

Exploiting the compositional semantics of $\mathsf{d}\mathcal{L}$, KeYmaera verifies properties of hybrid programs by proving corresponding properties of their parts in a sequent calculus [12,13]. For instance, the proof for specification (1) splits into a case where the if-statement takes effect and one where its condition gives false so that its body is skipped:

$$\frac{h = 0 \vdash [v := -cv]h \leq H \quad h \neq 0 \vdash h \leq H}{\vdash [\text{if } h = 0 \text{ then } v := -cv \text{ fi}]h \leq H} \text{ by } \mathsf{d}\mathcal{L} \text{ rule } \frac{F \vdash [\alpha]\phi \quad \neg F \vdash \phi}{\vdash [\text{if } F \text{ then } \alpha \text{ fi}]\phi}$$

6 Real Arithmetic and Computer Algebra

One challenge when verifying hybrid systems is the handling of intricate arithmetic resulting from continuous evolution along differential equations. If the

computer algebra system does not find a polynomial solution, we handle differential equations by their local dynamics using differential induction [11]. Otherwise, we substitute all occurrences of the evolution variables by their solutions at position τ. There, the fresh variable τ represents the evolution duration and is universally quantified for proving $[\alpha]\phi$ and existentially quantified for $\langle\alpha\rangle\phi$. Additionally, for all times in between 0 and τ the invariant must hold. Using the solution of the differential equation, the property $h \leq H$ in the proof of Example 1 yields:

$$\ldots \vdash \forall \tau{\geq}0 \left((\forall \tilde{\tau}\ (0{\leq}\tilde{\tau}{\leq}\tau \rightarrow \frac{-g}{2}\tilde{\tau}^2 + \tilde{\tau}v + h \geq 0)) \rightarrow \frac{-g}{2}\tau^2 + \tau v + h \leq H\right) \quad (2)$$

The inner quantifier checks if the invariant region $h \geq 0$ is respected at all times during the evolution. The symbolic decomposition rules of $\mathsf{d}\mathcal{L}$ result in quantified arithmetical formulas like (2). KeYmaera handles them using real quantifier elimination [3] as a decision procedure for real arithmetic, which is provided, e.g., using a seamless integration of KeYmaera with Mathematica.

7 Automation and Iterative Background Closure

KeYmaera implements a verification algorithm [10] that decomposes $\mathsf{d}\mathcal{L}$ formulas recursively in the $\mathsf{d}\mathcal{L}$ calculus. For the resulting arithmetical formulas, real quantifier elimination can be intractable in theory and practice. Experiments show that neither eager nor lazy calls of background solvers are feasible [10]. Due to the doubly exponential complexity of real quantifier elimination, eager calls seldom terminate in practice. For lazy calls, instead, the proof splits into multiple sub-problems, which can be equally computationally expensive.

To overcome the complexity pitfalls of quantifier elimination and to scale to real-world application scenarios, we implement an *iterative background closure* strategy [10] that interleaves background solver calls with deductive $\mathsf{d}\mathcal{L}$ rules. The basic idea is to interrupt background solvers after a timeout and only restart them after a $\mathsf{d}\mathcal{L}$ rule has split the proof. In addition, we increase timeouts for sub-goals according to a simple exponential scheme, see Fig. 4. The effect is that KeYmaera avoids splitting goals in the average case but is still able to split cases with prohibitive computational cost along their first-order and propositional structure.

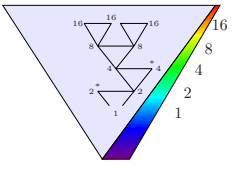


Fig. 4. Incremental timeouts in proof search space

8 Parameter Discovery

Required parameter constraints can be discovered in KeYmaera by selecting the appropriate parts obtained by symbolic decomposition and transformation by quantifier elimination. Essentially, equivalence-transformations of quantifier elimination yield equivalent parameter constraints when a proof does not succeed, which can be exploited for parameter discovery. See [12,13] for details.

Example 3. To obtain a fully parametric invariant for Example 2, properties for isolated modes like $[h'' = -g]h \leq H$ can be analyzed in KeYmaera. By selecting the relevant constraints from the resulting formula we obtain the invariant

$$v^2 \leq 2g(H - h) \wedge h \geq 0$$

which will be used for verifying the nondeterministic repetition of the system in Fig. 3. Assuming the invariant to hold in the initial state, we obtain a general parametric specification for the bouncing ball which is provable using KeYmaera:

$$(v^2 \leq 2g(H - h) \wedge h \geq 0 \wedge g > 0 \wedge H \geq 0) \rightarrow [\texttt{Ball}]\,(0 \leq h \leq H)$$

9 Applications

KeYmaera can be used for verifying hybrid systems even in the presence of parameters in the system dynamics. Its flexible specification logic $\mathsf{d}\mathcal{L}$ can be used for discovering the required parameter constraints. We have verified several properties of the European Train Control System (ETCS) successfully in KeYmaera, including safety, liveness, controllability, and reactivity, thereby entailing collision freedom [14]. In addition, collision avoidance has been verified for roundabout maneuvers in air traffic management [16], which involve challenging continuous dynamics with trigonometric functions.

10 Related Work

Davoren and Nerode [4] outline other uses of logic in hybrid systems. Theorem provers have been used for verifying hybrid systems in STeP [8] or PVS [1]. However, they do not use a genuine logic for hybrid systems but compile prespecified invariants of hybrid automata into an overall verification condition. Further, by using background solvers and iterative background closure strategies, we obtain a larger degree of automation than interactive proving in STeP [8] or higher-order logic [1]. VSE-II [9] uses discrete approximations of hybrid automata for verification. In contrast, KeYmaera respects the full continuous-time semantics of hybrid systems. PHAVer [5] is a model checker primarily for linear hybrid automata. CheckMate [15] supports more complex continuous dynamics, but still requires initial states and switching surfaces to be linear. KeYmaera supports nonlinear constraints on the system parameters as required for train applications or even the parametric bouncing ball.

References

1. Ábrahám-Mumm, E., Steffen, M., Hannemann, U.: Verification of hybrid systems: Formalization and proof rules in PVS. In: ICECCS, pp. 48–57. IEEE Computer, Los Alamitos (2001)
2. Beckert, B., Hähnle, R., Schmitt, P.H. (eds.): Verification of Object-Oriented Software. LNCS (LNAI), vol. 4334. Springer, Heidelberg (2007)

3. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. 12(3), 299–328 (1991)
4. Davoren, J.M., Nerode, A.: Logics for hybrid systems. IEEE 88(7) (July 2000)
5. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 258–273. Springer, Heidelberg (2005)
6. Harel, D., Kozen, D., Tiuryn, J.: Dynamic logic. MIT Press, Cambridge (2000)
7. Henzinger, T.A.: The theory of hybrid automata. In: LICS, pp. 278–292. IEEE Computer Society, Los Alamitos (1996)
8. Manna, Z., Sipma, H.: Deductive verification of hybrid systems using STeP. In: Henzinger, T.A., Sastry, S.S. (eds.) HSCC 1998. LNCS, vol. 1386. Springer, Heidelberg (1998)
9. Nonnengart, A., Rock, G., Stephan, W.: Using hybrid automata to express realtime properties in VSE-II. In: Russell, I., Kolen, J.F. (eds.) FLAIRS. AAAI Press, Menlo Park (2001)
10. Platzer, A.: Combining deduction and algebraic constraints for hybrid system analysis. In: Beckert, B. (ed.) VERIFY 2007 at CADE 2007, CEUR-WS.org (2007)
11. Platzer, A.: Differential algebraic dynamic logic for differential algebraic programs. (submitted, 2007)
12. Platzer, A.: Differential dynamic logic for verifying parametric hybrid systems. In: Olivetti, N. (ed.) TABLEAUX 2007. LNCS (LNAI), vol. 4548, pp. 216–232. Springer, Heidelberg (2007)
13. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reasoning (to appear, 2008)
14. Platzer, A., Quesel, J.D.: Logical verification and systematic parametric analysis in train control. In: Egerstedt, M., Mishra, B. (eds.) HSCC. LNCS, Springer, Heidelberg (2008)
15. Silva, B.I., Richeson, K., Krogh, B.H., Chutinan, A.: Modeling and verification of hybrid dynamical system using CheckMate. In: ADPM 2000: 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems (2000)
16. 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)