



ARCH-COMP19 Category Report: Hybrid Systems Theorem Proving

Stefan Mitsch¹, Andrew Sogokon¹, and Yong Kiam Tan¹,
Xiangyu Jin², Bohua Zhan², Shuling Wang², and Naijun Zhan²

¹ Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
{smitsch, asogokon, yongkiat}@cs.cmu.edu

² State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences
{jinxxy, bzhan, wangsl, znj}@ios.ac.cn

Abstract

This paper reports on the Hybrid Systems Theorem Proving (**HSTP**) category in the ARCH-COMP Friendly Competition 2019. The most important characteristic features of the HSTP category remain as in the previous edition [MST⁺18]: *i*) The flexibility of programming languages as structuring principles for hybrid systems, *ii*) The unambiguity and precision of program semantics, and *iii*) The mathematical rigor of logical reasoning principles. The HSTP category especially features many nonlinear and parametric continuous and hybrid systems. Owing to the nature of theorem proving, HSTP again accommodates three modes: **A) Automatic** in which the entire verification is performed fully automatically without any additional input beyond the original hybrid system and its safety specification. **H) Hints** in which select proof hints are provided as part of the input problem specification, allowing users to communicate specific advice about the system such as loop invariants. **S) Scripted** in which a significant part of the verification is done with dedicated proof scripts or problem-specific proof tactics. This threefold split makes it possible to better identify the sources of scalability and efficiency bottlenecks in hybrid systems theorem proving. The existence of all three categories also makes it easier for new tools with a different focus to participate in the competition, wherever they focus on in the spectrum from fast proof checking all the way to full automation. The types of benchmarks considered and experimental findings are described in this paper as well.

1 Introduction

This report summarizes the experimental results of the Hybrid Systems Theorem Proving (HSTP) category in the ARCH-COMP19 friendly competition. It is largely based on the previous edition of the HSTP category [MST⁺18]. The benchmark examples in the HSTP competition strive for a large variety in hybrid systems modeling patterns of basic extent to provide a low entry barrier for tools as well as examples at scale to identify opportunities for improving on proof automation, scalability and efficiency. The almost 160 examples in the benchmark competition are grouped into the following categories:

- Hybrid systems design shapes: small-scale examples over a large variety of model shapes to test for prover flexibility.
- Nonlinear continuous models: test for prover flexibility in terms of generating and proving properties about continuous dynamics.
- Hybrid systems case studies: hybrid systems models and specifications at scale to test for application scalability and efficiency.

In each of these categories, tools can select the degree of automation as follows, depending on their focus in the spectrum from fast proof checking to full proof automation:

- (A) Automated: hybrid systems models and specifications are the only input, proofs and counterexamples are produced fully automatically.
- (H) Hints: select proof hints (e.g., loop invariants) are provided as part of the specifications.
- (S) Scripted: significant parts of the verification is done with dedicated problem-specific scripts or tactics.

All benchmark examples are available at <https://github.com/LS-Lab/KeYmaeraX-projects/tree/master/benchmarks> and specified in differential dynamic logic (dL) [Pla08, Pla17], whose format and ASCII syntax are presented in Section 2. The participating tools are presented in Section 3. An overview of the examples together with the findings from the competition is given in Section 4.

2 Problem Format

All benchmarks in the Hybrid Systems Theorem Proving (HSTP) category are written in differential dynamic logic (dL) [Pla08, Pla17] which has axioms and an unambiguous semantics available [BRV+17] in KeYmaera 3, KeYmaera X, Isabelle/HOL, and Coq. To make it easier for tools to participate in the HSTP category, almost all benchmarks in the HSTP category are differential dynamic logic formulas of the particular safety form

$$\phi \rightarrow [\alpha]\psi \quad (1)$$

where ϕ is a real arithmetic formula describing the initial conditions,
 ψ is a real arithmetic formula describing the postcondition / set of safe states, and
 α is the hybrid system described using hybrid programs as a program notation.

The dL formula (1) means that if the system starts in a state satisfying the initial condition ϕ , then all final states of all possible runs of the hybrid system α satisfy postcondition ψ . The operators / statements of hybrid programs are summarized in Table 1. Those of logical formulas in dL are summarized in Table 2. In particular, the hybrid program α contains both the discrete and continuous dynamics of the hybrid system.

An example with a purely continuous system is:

$$-\frac{4}{5} < x < -\frac{1}{3} \wedge -1 \leq y < 0 \rightarrow [x' = 2x - 2xy, y' = 2y - x^2 + y^2](x + y \leq 1 \wedge (x \neq 0 \vee y \neq 0)) \quad (2)$$

An example with a trivial hybrid system is:

$$v \geq 0 \wedge A > 0 \wedge b > 0 \rightarrow [(?v \leq 5; a := A \cup a := -b); \{x' = v, v' = a \& v \geq 0\}^*] v \geq 0 \quad (3)$$

This particular example is completely trivial, because the postcondition $v \geq 0$ directly follows from the evolution domain constraint $v \geq 0$ in the differential equation. But safety properties become more exciting and more challenging when the postcondition is a different one. For

Table 1: Statements of hybrid programs (Q 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
$(x'_1 = \theta_1, \dots, x'_n = \theta_n \& Q)$	continuous evolution of x_i along the differential equation system $x'_i = \theta_i$ of any non-negative duration restricted to remain in evolution domain Q throughout
$?Q$	test if formula Q holds at current state, abort program otherwise
$\text{if}(Q) \alpha$	run α if Q is true at current state, do nothing otherwise
$\text{if}(Q) \alpha \text{ else } \beta$	run α if Q is true at current state, run β otherwise

Table 2: Operators of differential dynamic logic ($\text{d}\mathcal{L}$) formulas

$\text{d}\mathcal{L}$	Operator	Meaning
$\theta_1 \sim \theta_2$	comparison	true iff $\theta_1 \sim \theta_2$ with operator $\sim \in \{>, \geq, =, \neq, \leq, <\}$
$\neg\phi$	negation / not	true if ϕ is false
$\phi \wedge \psi$	conjunction / and	true if both ϕ and ψ are true
$\phi \vee \psi$	disjunction / or	true if ϕ is true or if ψ is true
$\phi \rightarrow \psi$	implication / 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 in \mathbb{R}
$\exists x \phi$	existential quantifier	true if ϕ is true for some values of variable x in \mathbb{R}
$[\alpha]\phi$	$[\cdot]$ modality / box	true if ϕ is true after all runs of hybrid program α
$\langle\alpha\rangle\phi$	$\langle\cdot\rangle$ modality / diamond	true if ϕ is true after at least one run of α

example, $x \geq 10$ to say that the position is at least 10 always is much more complicated (and not even true for the above example).

Note that the operator precedence is such that unary operators bind stronger than binary operators and, just like in regular expressions, $;$ binds stronger than \cup . In particular, the controller in (3) is $(?(v \leq 5); a := A) \cup a := -b$.

ASCII syntax. The benchmark examples are specified in the $\text{d}\mathcal{L}$ ASCII syntax and grouped into `.kx` files, each containing several named archive entries. The ASCII syntax is a straightforward ASCII rendition of Tables 1 and 2, e.g., using `A->B` for $A \rightarrow B$ and using `A&B` for $A \wedge B$. The ASCII notation `alpha++beta` is used for $\alpha \cup \beta$. For improved readability in longer examples, braces `{...}` are used for grouping differential equation systems and other program operators. Like in C programs, assignments etc. end with explicit semicolons.

Archive entries follow the general shape below, listing optional definitions, system variables, a (safety) specification in $\text{d}\mathcal{L}$, and optional tactic scripts. The example (3), specialized, just for the sake of illustration, to the case where $A = 5$, is written in ASCII KeYmaera X input as follows. Unlike the `ProgramVariables` and `Problem` block, the `Definitions` and `Tactic` blocks are optional. The symbols defined in the `Definitions` can be used in the `Problem` block or in other definitions. All examples are additionally provided in the format of the previous edition [MST⁺18] of the HSTP category.

ArchiveEntry "Benchmark Example 1"

```

Definitions                                /* definitions cannot change their value */
  Real A() = 5;                               /* real-valued maximum acceleration defined to be 5 */
  Real b();                                   /* real-valued braking, undefined so unknown value */
  Bool geq(Real x, Real y) <-> x>=y; /* predicate geq defined to be formula x>=y */
  HP drive ::= {                               /* program drive defined to choose either */
    ?v<=5; a:=A();                             /* maximum acceleration if slow enough */
    ++ a:=-b();                                /* or braking, nondeterministically */
  };
End.

```

```

ProgramVariables /* program variables may change their value over time */
  Real x;                                     /* real-valued position */
  Real v;                                     /* real-valued velocity */
  Real a;                                     /* current acceleration chosen by controller */
End.

```

```

Problem                                     /* conjecture in differential dynamic logic */
  v>=0 & A()>0 & b()>0                       /* initial condition */
  ->                                           /* implies */
  [                                           /* all runs of this hybrid program */
    {                                         /* braces {} group programs */
      drive;                                 /* expand program drive here as defined above */
      { x'=v, v'=a & v>=0 }                /* differential equation system */
    } * @invariant(v>=0)                    /* loop repeats, with @invariant contract */
  ] v>=0                                     /* safety/postcondition after hybrid program */
End.

```

```

Tactic "Automated proof in KeYmaera X"
  master
End.

```

```

Tactic "Scripted proof in Bellerophon tactic language"
  implyR(1) ; loop({v>=0}, 1) ; <( /* < splits separate branches */
    closeId,                               /* initial case: shown with close by identity */
    QE,                                     /* postcondition: prove by real arithmetic QE */
    /* induction step: decomposes hybrid program semi-explicitly */
    composeb(1) ; solve(1.1) ; choiceb(1) ; andR(1) ; <( /* controller branches */
      composeb(1) ; testb(1) ; master,      /* decompose some steps then ask master */
      assignb(1) ; QE                       /* assignment, then real arithmetic */
    )
  )
End.

```

```

End. /* end of ArchiveEntry */

```

Background. A short survey on differential dynamic logic and hybrid programs can be found in a LICS’12 tutorial [Pla12a], a tutorial on its modeling principles in STTT [QML+16], a research monograph [Pla10b], and a comprehensive introduction in a textbook [Pla18]. The precise mathematical semantics of differential dynamic logic and its hybrid programs can be found in the literature as well, for example the most recent details in [Pla17], and a brief version in the LICS’12 tutorial [Pla12a].

3 Participating Tools

KeYmaera X. KeYmaera X [FMQ+15] is a theorem prover for the hybrid systems logic differential dynamic logic ($d\mathcal{L}$). It implements the uniform substitution calculus of $d\mathcal{L}$ [Pla17].¹ KeYmaera X supports systems with nondeterministic discrete jumps, nonlinear differential equations, nondeterministic input, and it provides invariant construction and proving techniques for differential equations [SGJP16, PT18]. Unlike numerical hybrid systems reachability analysis tools, KeYmaera X also supports unbounded initial sets and unbounded time analysis. KeYmaera X participates in v4.4.3 (2018) and the latest v4.6.3 (2019).

KeYmaera X comes with automated proof search procedures that can be steered in the following ways: annotations in the input models provide additional design insight and, if available, are used to steer the invariant generation techniques in KeYmaera X; fine-grained control over proofs is available with proof scripts [FMBP17].

Extension with and experimentation in proof search without reducing trust in the prover is made possible on top of a small trusted kernel that checks all reasoning steps for soundness. The prover kernel contains a list of sound $d\mathcal{L}$ axioms that are instantiated using a uniform substitution proof rule [Pla17]. This approach isolates all soundness-critical reasoning in the prover kernel and obviates the intractable task of ensuring that each new proof search algorithm is implemented correctly. New proof search algorithms are always sound and can either be programmed directly in Scala (or Java) or can simply be added as a tactic in the hybrid systems tactic language Bellerophon [FMBP17].

The proof automation for differential equations makes use of insights on how to prove all invariants of differential equations [PT18]. Tactical implementations allow KeYmaera X to soundly reduce ODE invariance questions to a small number of core ODE axioms and real arithmetic. The proof tactic is optimized for fast proofs of commonly used invariants, e.g., barrier certificates [PJP07]. All real arithmetic questions that arise in the proofs are rigorously checked, *including* the ones that arise from the use of barrier certificates. This guarantees that any barrier certificate that proves with KeYmaera X is a *true* barrier certificate, rather than the result of numerical or floating-point errors.

To prove properties of differential equations, KeYmaera X combines an axiomatic differential equation solver [Pla17] and local fixedpoint computation for differential invariants [PC09a] with tactics based on differential equation axiomatization [PT18], and Pegasus: a toolbox for automatically generating continuous invariants for systems of ordinary differential equations. Given a system of ODEs subject to an evolution domain constraint, a set of initial states, and a set of unsafe states, Pegasus will attempt to automatically generate a continuous invariant that is sufficient to prove that the ODE cannot continuously evolve into an unsafe state from any of its initial states while respecting the evolution constraint. Pegasus is implemented in Mathematica and is able to connect to Matlab; at present it implements an array of techniques from qualitative analysis and discrete abstraction [SGJP16] for constructing continuous invariants

¹This $d\mathcal{L}$ uniform substitution calculus is also formally verified in Isabelle/HOL and Coq [BRV+17].

and is additionally capable of searching for *barrier certificates* (using both sum-of-squares optimization [PJ04] and linear programming [SC⁺13]), as well as *Darboux polynomials* (by making use of algorithms developed for the *Prelle-Singer* procedure [Man93]). The methods for continuous invariant generation within Pegasus are deployed in a targeted fashion, and depend on the nature of the verification problem: the important features of the problem are extracted by the tool in a pre-processing step which suggests promising strategies for invariant generation base on those features.

KeYmaera 3. KeYmaera 3 [PQ08] is the previous generation theorem prover for differential dynamic logic $d\mathcal{L}$. Unlike its successor KeYmaera X, the older KeYmaera 3 directly implements a sequent calculus for differential dynamic logic [Pla08], instead of a uniform substitution calculus. What KeYmaera X implements from a few simple modular axioms, KeYmaera 3 uses several dedicated proof rules for [Pla08, Pla10a, Pla12b]. This leads to a more directly usable but substantially bigger soundness-critical prover kernel of about 66000 lines of code written in a mix of Java and Scala. In some cases, one single proof rule use, e.g., for solving differential equations in KeYmaera 3 corresponds to thousands of axiom uses in KeYmaera X. The impact on soundness, however, is that the ODE solver of KeYmaera 3 is trusted while that of KeYmaera X is not trusted, because each of its outputs is verified with a proof.

For proof automation, KeYmaera 3 implements a simple but fast fixpoint loop [PC09a] for generating loop invariants of hybrid systems and differential invariants of differential equations. It provides an array of different SMT strategies for splitting real arithmetic subquestions [Pla10b]. Changing proof search procedures in KeYmaera 3 (beyond choosing from the list of predefined ones) is significantly more complicated and, notably, soundness-critical.

HHL Prover. HHL prover [WZZ15] is an interactive theorem prover for verifying hybrid systems modelled by Hybrid CSP (HCSP) [He94, ZWR96]. It implements the Hybrid Hoare Logic (HHL) [LLQ⁺10], a Hoare style logic for reasoning about HCSP, in the proof assistant Isabelle/HOL [NWP02].

HCSP extends CSP by introducing continuous variables, differential equations, and interruptions by domain boundary and communication. Given an HCSP process P , an HHL specification takes the form $\{Pre\} P \{Post; HF\}$, where Pre and $Post$ are pre-/post-conditions in first-order logic, and HF is a history formula in duration calculus [ZH04] for specifying time-related properties held throughout the whole execution interval. HHL defines a set of proof rules for deducing such specifications for HCSP. HHL prover formalizes HCSP and HHL for proving partial correctness of hybrid systems.

For reasoning about differential equations, HHL includes a proof rule that reduces the specification of the continuous evolution to be proved to the synthesis problem of differential invariants for the corresponding differential equations. HHL prover resorts to an external invariant generator based on quantifier elimination or sum-of-squares (SOS) relaxation, to automatically solve the unproven constraints containing unknown differential invariants. The invariant generator relies on the solvers for quantifier elimination and semi-definite programming for constructing differential invariants. In the newest version of HHL prover, we removed the dependency on Mathematica for the SOS-based invariant generator.

For the newest version of HHL prover, we also borrow some idea from [Pla18] and add the differential cut and ghost rules to the HHL framework. Differential cut rule strengthens the domain of differential equations by an invariant property proved to hold throughout the continuous evolution, while the differential ghost rule adds new continuous variables with new differential equations without affecting the original differential equations. These rules make

reasoning about differential equations more flexible. For differential invariants, the current version of HHL prover resorts to external tools for both invariant verification and generation. Stating proof rules for reasoning about differential invariants within Isabelle is left open for future work.

For the newest version of the HHL Prover, we added an invariant checker using quantifier elimination from the external tool Redlog. Invoking Redlog from Isabelle follows the same pattern as for invariant generation. First, the goal to be proved, which may contain both entailments (of the form $p \implies q$) and preservation of invariant by a differential equation (of the form $\text{exeFlow}(\mathbf{x}' = f(\mathbf{x}), I) \implies I$), is translated from Isabelle's abstract syntax tree to JSON format. Next, the JSON file is translated to the input for Redlog using a Python script. In this step, preservation of invariant is converted to appropriate entailments involving the domain of evolution and the Lie derivative of the invariant. Finally, Redlog is invoked on the output of the Python script and checks each of the resulting entailments. This sequence is implemented as a bash script and invoked by an oracle in Isabelle, which checks the final output produced by Redlog. The entire process is automatic, after the user supplies the invariant in usual mathematical notation, and a list of constants that are relevant for quantifier elimination.

We translated a subset of the examples in the basic design shapes and the nonlinear continuous models of the benchmarks to HCSP. There are some essential differences between the semantics of $d\mathcal{L}$ and HCSP. In particular, in HCSP, evolution by a differential equation cannot stop before reaching the boundary (or interrupted by a communication). Also, there is no assignment to an arbitrary value ($x := *$) in HCSP. As a result, some of the examples cannot be translated naturally. For those examples that can be translated naturally, we proved some of them in Isabelle, with the help of invariant checking using Redlog.

4 Benchmarks

One of the strengths of hybrid systems theorem proving as a verification technique is its support for combined automated and interactive verification steps as well as its applicability to proof search and proof checking. The benchmark examples were analyzed in three modes:

Automated The specification is the only input to the theorem prover. Proofs and counterexamples are obtained fully automated to highlight the capabilities of theorem provers in terms of invariant generation, proof search, and proof checking.

Hints Known design properties of the system, such as loop invariants and invariants of differential equations, are annotated in the model and allowed to be exploited during an otherwise fully automated proof to highlight the capabilities of theorem provers in terms of proof search and proof checking.

Scripted User guidance with proof scripts is allowed to highlight the capabilities of theorem provers in terms of proof checking.

The benchmark examples are structured into 3 categories: hybrid systems design shape examples to test for system design variations at a small scale, nonlinear continuous models to test for continuous invariant construction and proving capabilities, and hybrid systems case studies to test for prover scalability.

Experimental setup. The machines used to run the benchmark examples are listed in [Appendix A: KeYmaera X 4.4.3](#) and [KeYmaera X 4.6.3 \(in automated \(A\), hints \(H\), and scripted](#)

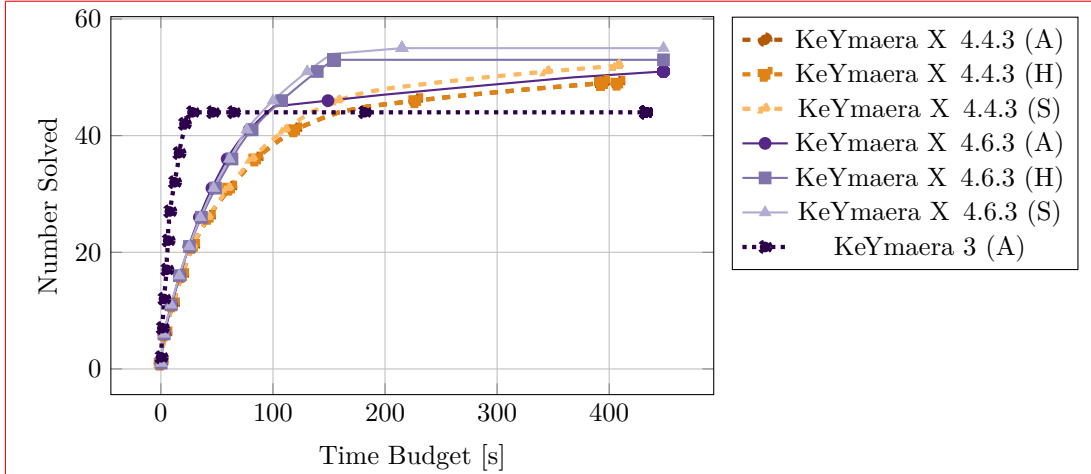


Figure 1: Computation times: Basic benchmark examples. Ranked accumulated time budgets [s], which are the number of examples solved within a total accumulated time budget

(S) mode) and KeYmaera 3 (in automated (A) mode) participated on all benchmark sets and were executed on the same machine M_k , and therefore their computation times are directly comparable. HHL Prover participated with the Chinese train control system, lunar lander descent guidance, and roller-coaster safety case studies, as well as on a subset of the hybrid systems design shapes and the nonlinear continuous models on its own machine M_{hhl} . The execution time measurements were taken separately on a fresh prover instance for each example in the benchmark set. Proof attempts were aborted after a category-specific timeout, well above the longest successful solution in the category. The competition results are presented with *accumulated execution times* after examples are ranked according to their execution time.

4.1 Hybrid Systems Design Shapes

Category overview. In this category (unmodified from 2018 [MST⁺18]), basic examples² test for proof automation techniques for a large variety of system designs: event-triggered systems, time-triggered systems, systems with nested loops and differential equations, and systems with model-predictive control. Instead of focusing on particularly complex systems, this set of examples strives at a certain degree of coverage of qualitatively different kinds of systems and their different typical shapes. The benchmark examples are grouped as follows:

Static semantics correctness 9 examples with various sequential orders and nested structures of assignments, differential equations, and loops.

Dynamics 30 examples with differential equations ranging from solvable to nonlinear.

LICS Tutorial 9 $d\mathcal{L}$ tutorial examples [Pla12a] ranging from basic time-triggered motion control to model-predictive control.

STTT Tutorial 12 $d\mathcal{L}$ modeling tutorial examples [QML⁺16] ranging from basic discrete event-triggered and time-triggered control for straight-line motion to speed control with a trajectory generator and lane-keeping with two-dimensional curved motion.

²<https://github.com/LS-Lab/KeYmaeraX-projects/blob/master/benchmarks/basic.kyx>

Competition results. Proof attempts were aborted after a timeout of 300s in the basic category, with the longest successful solution after about 80s. The results for the basic category in terms of accumulated execution times are shown in Fig. 1. For example, the fastest 30 fully automated examples can all be solved in cumulative time 10.5s in KeYmaera 3 vs. 57.5s in KeYmaera X 4.4.3 vs. 43.1s in KeYmaera X 4.6.3 (note that the sets of fastest examples are not necessarily the same). The main insight from Fig. 1 is that automated proof search with a fixpoint loop in KeYmaera 3 is faster than the proof search in KeYmaera X for the solved examples, but KeYmaera X solves a larger portion of the benchmark set with the additional time it takes. Hints and proof scripts in KeYmaera X help speed up a little bit and solve additional examples. This indicates that the primary impact of further proof automation *for the basic category of benchmarks* will not be the resulting speed but the number of examples that can be proved fully automatically.

4.2 Nonlinear Continuous Models

Category overview. This set of 99 nonlinear continuous safety verification problems³ is based on the problems proposed in [SGJ16] and significantly extended from [MST+18]. The problems in this benchmark set were gathered from published papers in the area of continuous safety verification and invariant generation for nonlinear systems ([DGXZ17, LZZ11, DCKB17, SGS14, SGJP16]). The bulk of the problems in the benchmark set feature planar (i.e., 2-dimensional) polynomial systems of ODEs in which the safety property is known to hold for unbounded time. The ODEs are furthermore autonomous (i.e., do not explicitly depend on the independent time variable t); this fact presents no real restriction since non-autonomous ODEs can be brought into autonomous form by augmenting the dynamics with $t' = 1$. Certain non-polynomial systems of ODEs can likewise be brought into polynomial form by introducing fresh variables in a process called *re-casting* [SV87]. While we stress that the existing set of nonlinear polynomial ODE safety benchmarks can in no way be said to be representative (owing to its small size), the general class of problems which fits into this category is highly important.

Example 4.1. The nonlinear system from [DLA06, Ex. 5.2. ii] that was shown in (2) has the following dynamics:

$$\begin{aligned}x' &= 2x - 2xy, \\y' &= 2y - x^2 + y^2.\end{aligned}$$

Taking the initial states to be $-\frac{4}{5} < x < -\frac{1}{3} \wedge -1 \leq y < 0$ and $(x = 0 \wedge y = 0) \vee x + y > 1$ to be the forbidden states, the verification problem is illustrated in Fig. 2.

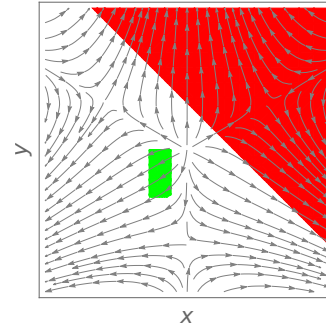


Figure 2: Nonlinear continuous safety verification problem. No initial state (green rectangle) can evolve into unsafe states (red half-plane) along the trajectories.

Competition results. Proof attempts in the nonlinear category were aborted after a timeout of 300s, above the longest successful solution of about 126s in automated mode and 243s in scripted mode. Fig. 3 plots the accumulated execution times for the nonlinear category after examples are ranked according to their execution time. The main insight from Fig. 3 is that the invariant construction [SGJP16] and proving techniques [PT18]

³<https://github.com/LS-Lab/KeYmaeraX-projects/blob/master/benchmarks/nonlinear.kyx>

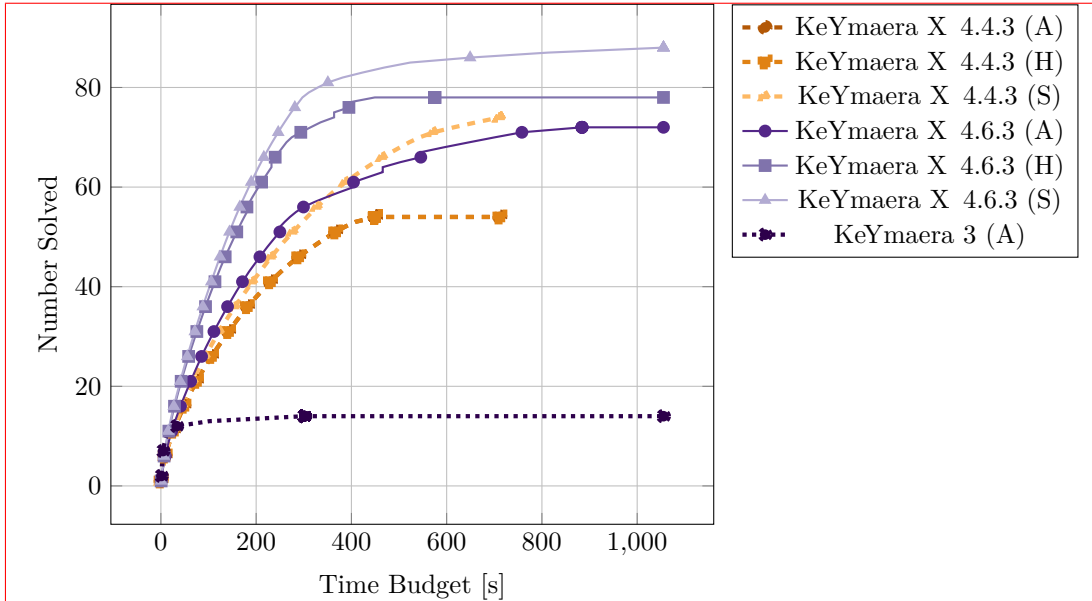


Figure 3: Computation times: Nonlinear benchmark examples. Ranked accumulated time budgets [s], which are the number of examples solved within a total accumulated time budget. Automation in KeYmaera X 4.6.3 outperforms hints and automation and is on par with scripting in KeYmaera X 4.4.3. KeYmaera X 3 solves only a small number of nonlinear examples.

of KeYmaera X significantly outperform and improve upon KeYmaera 3 the extent to which continuous dynamics can be analyzed fully automatically. Even proof hints have a negligible impact compared to full automation.⁴ The results in scripted mode (S) emphasize the generality of the implemented proving techniques for differential equations: proof scripts with barrier certificates that were generated outside KeYmaera X solve almost all the remaining examples. This highlights a potential to improve automated invariant construction with methods to construct barrier certificates, which are plagued by numerical robustness issues.

4.3 Hybrid Systems Case Study Benchmarks

Category overview. The benchmark examples in this category are selected to test theorem provers for scalability and efficiency on examples of a significant size and interest in applications and extended over [MST⁺18]. The benchmark examples⁵ are inspired from prior case studies on train control [PQ09, ZLW⁺14], flight collision avoidance [PC09b], robot collision avoidance [MGVP17], a lunar lander descent guidance protocol [ZYZ⁺14a], and rollercoaster safety [BLCP18a].

European train control system (ETCS). This benchmark on automated train control bases on the safety analysis [PQ09] of the cooperation protocol in the European Train Control System [ERT02, DHO03], which specifies the interaction between an automated train protection system and a radio-block controller. The radio-block controller (purely discrete dynamics) may

⁴But this observation could be sensitive to the chosen benchmarks.

⁵<https://github.com/LS-Lab/KeYmaeraX-projects/blob/master/benchmarks/advanced.kyx>

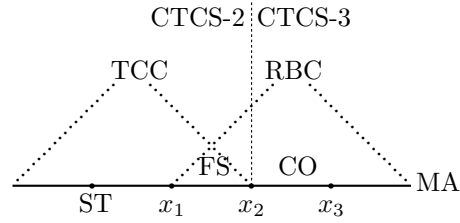


Figure 4: A combined scenario of CTCS-3.

at any time issue speed limits that take effect at certain positions; the train must respect these speed limits (hybrid dynamics of train controller and train motion).

E-1 (ETCS: Essentials) Describes the core safety theorem: a time-triggered train controller never violates the posted speed limit.

E-2 (ETCS: Proposition 1 (Controllability)) Describes the motion of a train on brakes and translates it into a stopping distance. Tests a prover’s ability to show equivalence between a hybrid systems specification in $d\mathcal{L}$ and its core information in terms of stopping distance in real arithmetic.

E-3 (ETCS: Proposition 4 (Reactivity)) Describes the motion of a train when accelerating for a bounded amount of time and the necessary distance to a full stop. Tests a prover’s ability to work with universally quantified assumptions and/or analyze programs in the context of universally quantified input.

The benchmark tests a prover’s ability to handle $d\mathcal{L}$ safety properties (modal formulas) in various places of a specification, for example, as proof obligations and as assumptions.

Chinese train control systems (CTCS). This case study is about modeling and verification of a combined operational scenario of Chinese Train Control System Level-3 (CTCS-3). It originates from an under-specification error of the System Requirements Specification (SRS) of CTCS-3, revealed during a spot testing of the system, which caused a train to stop unexpectedly. It has been studied in [ZLW⁺14, ZZW⁺13, ZZWF15] and the failure was reproduced by simulation and also formally verified.

The combined scenario integrates the movement authority (MA) scenario, the level transition (from CTCS-2 to CTCS-3) scenario, as well as the mode transition (from Full Supervision mode to Calling On mode, FS to CO for short) scenario of CTCS-3. The combined scenario is shown in Fig. 4, which occurs under the following situation:

- The train has got enough MA to complete the combined scenario, and
- There are two adjacent segments in the MA, divided by location x_2 . At x_2 , the level transition from CTCS-2 to CTCS-3, and the mode transition from FS to CO, will occur simultaneously, and
- The train starts to move at location ST , and has an agreement from RBC (Radio Block Center) to start level transition at x_1 and complete the level transition at x_2 .

According to the SRS, the combined scenario is required to satisfy a *liveness property*: the train can eventually move beyond the location x_2 with a positive speed, with both the level transition and mode transition completed successfully.

However, the under-specified SRS fails to guarantee the liveness property. Basically, for safety reasons, to switch from FS mode to CO mode under CTCS-3, the driver’s confirmation is required before the switching point x_2 to upgrade the speed limit of the CO mode, which is originally set to 0. However, in the old version of the SRS, such a confirmation request is not

explicitly specified to be issued to the driver during a region where the train is co-supervised by both CTCS-2 and CTCS-3 (x_1 to x_2 in Fig. 4). As a result, the speed limit of the CO segment cannot be upgraded and remains 0, which forces the train to stop at x_2 . Thus the verification objective for this case study is to prove on the underspecified model the *negation* of the liveness property, that is, the train must stop at x_2 .

Roundabout air traffic conflict resolution (ATC). Air traffic conflict resolution maneuvers with curved flight dynamics exhibit nontrivial interactions of discrete and continuous dynamics. The roundabout benchmark [PC09a] is based on [TPL⁺96, TPS98, HHMW00, MF01, DPR05, PC09b, PKV09] to analyze collision freedom of planar roundabout maneuvers in air traffic control that should guarantee safe spatial separation of aircraft throughout their flight. The scale of this benchmark can be adjusted easily with the number of aircraft involved in the conflict resolution maneuver: additional aircraft increase the number of variables in the benchmark and introduce additional invariants that must be found, but analysis is separable into pairwise collision freedom questions.

A-2 (ATC: 2 Aircraft Tangential Roundabout Maneuver) Describes the circular conflict resolution of two aircraft in a planar roundabout collision avoidance maneuver.

A-3 (ATC: 3 Aircraft Tangential Roundabout Maneuver) Circular conflict resolution of three aircraft in planar roundabout collision avoidance maneuvers. Safety of the entire system is collision-freedom between all three aircraft pairs.

A-4 (ATC: 4 Aircraft Tangential Roundabout Maneuver) Circular conflict resolution of four aircraft in planar roundabout collision avoidance maneuvers. Safety of the entire system is collision-freedom between all six aircraft pairs.

The benchmark tests a prover’s ability to analyze nested loops and multiple nonlinear differential equations. At larger numbers of aircraft it also tests the scale of reasoning about nonlinear dynamics by identifying and splitting analysis into isolated sub-questions.

Robot collision avoidance (RX). This benchmark bases on [MGVP17] and analyzes obstacle avoidance in ground robot navigation. The benchmark uses models and safety properties to analyze collision avoidance safety in the presence of stationary obstacles and moving obstacles.

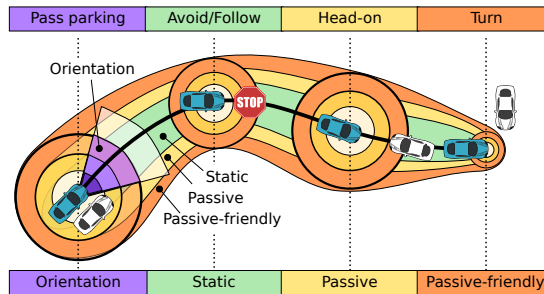


Figure 5: Robot collision avoidance properties: benchmark tests static safety and passive safety.

The resulting real arithmetic formulas describing the Euclidian distance between robot and obstacle after symbolic execution are challenging for current solvers and may require overapproximation and simplification in the theorem prover steering the backend decision procedures.

R-1 (Robot collision avoidance: static safety) ensures that no collisions can happen with stationary obstacles. Tests a prover’s ability to handle mixed solvable (longitudinal robot

acceleration) and nonlinear (rotational robot motion) continuous dynamics, and its ability to overapproximate norms (Euclidian distance overapproximated to infinity norm).

R-2 (Robot collision avoidance: passive safety) ensures that no collisions can happen with stationary or moving obstacles while the robot moves. The size of the resulting real arithmetic formulas are challenging for current solvers even after overapproximation of Euclidian distances. Tests a prover’s ability to steer backend decision procedures by selecting relevant assumptions, using monotonicity arguments to eliminate variables, and simplify arithmetic.

This benchmark tests a prover’s ability to analyze mixed solvable and nonlinear differential equations, overapproximation of norms, and arithmetic simplifications.

Lunar Lander Descent Guidance (LLDG). The lunar lander control program is a closed loop system, which is composed of the lander’s dynamics and the guidance program for the slow descent phase. The guidance program is executed periodically with a fixed sampling period. At each sampling point, the current state of the lander is measured by inertial measurement unit or various sensors. Processed measurements are then input into the guidance program, which outputs control commands, e.g. the magnitude and direction of thrust, to be imposed on the lander’s dynamics in the following sampling cycle. The mathematical description of the lander’s dynamics as well as the guidance program of the slow descent phase can be found in [ZYZ⁺14b, ZWZ16].

Rollercoaster Safety (RCS). The roller-coaster safety case study [BLCP18b] is a benchmark in component-based verification combining smaller-scale components with non-trivial continuous dynamics to a full large-scale hybrid system. The components represent motion of a coaster car along geometrical primitives (straights, arcs) that can be connected to form complicated track shapes of varying scale.

Competition results. Proof attempts in the hybrid systems case study category were aborted after a timeout of 1500s, with the longest successful proof after about 1400s. KeYmaera 3 and KeYmaera X participated on the full benchmark set (CTCS and LLDG attempted only in automated (A) mode, RCS only in scripted mode; future verification of CTCS and LLDG with hints and scripts is planned). HHL Prover participated on the CTCS, LLDG, and RCS case study.

Again, the fixpoint loop invariant generation technique in KeYmaera 3 solves examples fast, while hints and proof scripts in KeYmaera X help scale. The results point out a potential to improve tactic implementation efficiency, since on the hybrid systems case studies that KeYmaera 3 can solve automatically it outperforms proof checking from hints in KeYmaera X. The ETCS benchmark examples feature solvable continuous dynamics, which unsurprisingly leads to a significant computation time difference between the ODE solution sequent rule in KeYmaera 3 and the proof-producing tactic in KeYmaera X. The ATC benchmark examples highlight a particularly useful proof scalability technique in KeYmaera 3, which splits conjunctive safety properties into separate proof obligations. This technique is mimicked in the KeYmaera X 4.6.3 scripted mode to reduce computation time, for example, in ATC A-4 from 1306s to 148s, but requires further tactic improvements to get to the computational efficiency of KeYmaera 3. The robot collision avoidance benchmark examples illustrate where current automation fails to find invariants and identify the necessary arithmetic simplifications for backend procedures to

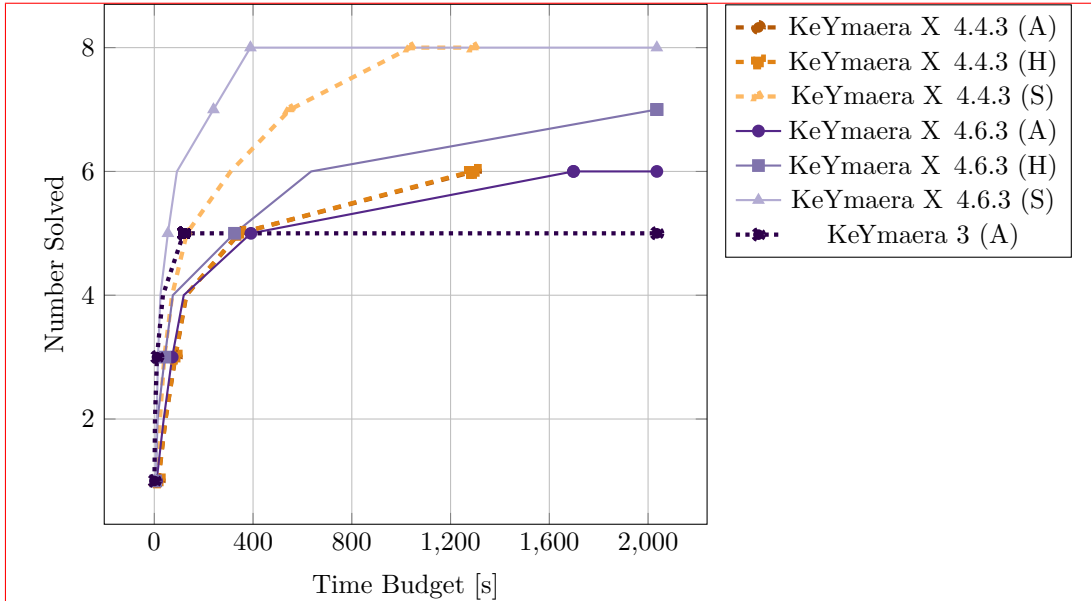


Figure 6: Computation times: Case study benchmark examples. Ranked accumulated time budgets [s], which are the number of examples solved within a total accumulated time budget

complete in reasonable time; proof checking from hints and scripts for arithmetic simplifications illustrate potential ways forward to improve proof search automation.

Note HHL Prover. The CTCS case study is identical to the previous edition [MST⁺18]: a Simulink/Stateflow model has been built for the combined scenario in the CTCS-3 case study. Applying the tool Sim2HCSP to the Simulink/Stateflow model, seven files were generated which describe the HCSP model as well as the goal to be verified. Then using HHL Prover, the goal was proved successfully as a theorem, taking 59 seconds to finish on the M_{hhl} platform with Intel Core i7-4790 CPU 3.60GHZ and 16GB memory. In particular, during the interactive proof process, certain differential invariants were manually fed into the HHL specification.

In the LLDG case study, the entire Isabelle theory including the model, specification, and proof for the entire example is 327 lines long. By applying HHL prover, the unproven subgoals related to differential invariants are transformed to a set of SOS constraints with respect to the user-defined invariant template, and then the SOS-based invariant generator is invoked on these constraints to synthesize a satisfying invariant.

In the RCS case study, the roller coaster example [BLCP18b] was converted to HCSP. The conversion is natural, as the differences between $d\mathcal{L}$ and HCSP does not produce any problems. The proof makes use of invariant checking using Redlog, as well as the newly added differential ghost rule. The entire Isabelle theory (including the model, specification, and proof for all ten parts of the example) is 1141 lines long.

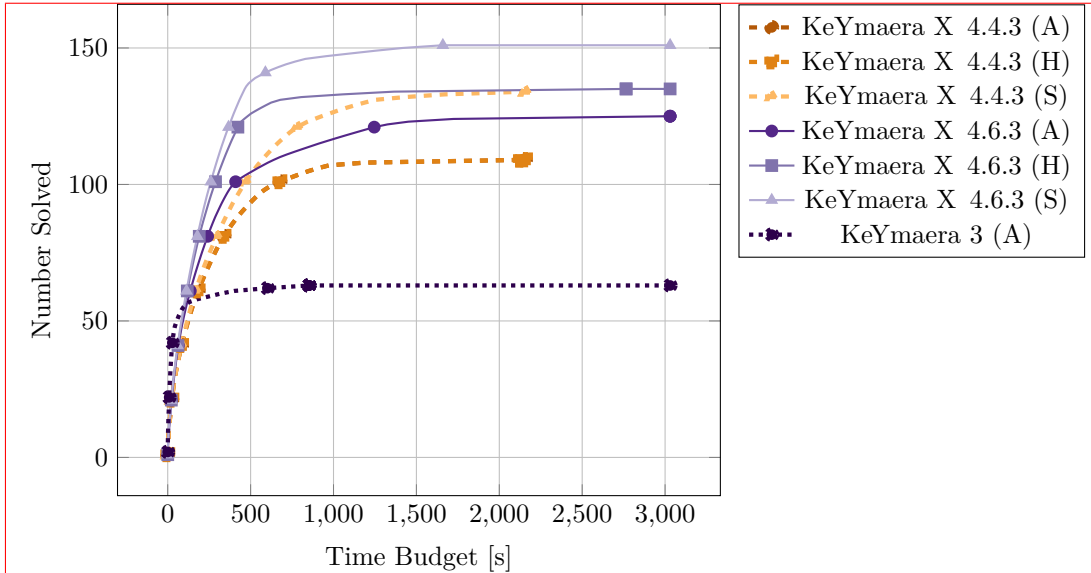


Figure 7: Ranked accumulated time budgets [s]: number of examples solved in total accumulated time budget (steeper is better). Result summary: KeYmaera 3 sets out faster, but KeYmaera X 4.6.3 closes the performance gap of KeYmaera X 4.4.3 (curves are steeper longer). KeYmaera 3 solves less examples, especially among those with nonlinear dynamics. KeYmaera X scales better; hints and scripts increase the number of solved examples and reduce computation time.

5 Conclusion and Outlook

The hybrid systems theorem proving friendly competition focuses on the characteristic features of hybrid systems theorem proving: flexibility of programming language principles for hybrid systems, unambiguous program semantics, and mathematically rigorous logical reasoning principles.

The (almost 160) benchmark examples are chosen to reflect a large variety of hybrid systems model shapes and scales to test hybrid systems theorem provers both for their flexibility to analyze typical modeling styles and for their scalability. More potential benchmark examples are always welcome in future years of the competition! The hybrid systems theorem proving category allows tools to choose their operating mode on the spectrum from fast proof checking of scripted proofs, hint-supported proof search and checking, to full automation.

The results, summarized in Figures 7–9, show significant improvements over theorem prover generations (KeYmaera X compared to its predecessor KeYmaera 3) in handling continuous dynamics fully automatically, but also highlight that the tactics in KeYmaera X, even though they are closing the gap, can still learn in terms of performance from the proof search and checking procedures of KeYmaera 3. The comparison between KeYmaera X 4.4.3 and KeYmaera X 4.6.3 illustrates the benefit of separating tactics and core in the KeYmaera X architecture: proof automation in KeYmaera X 4.6.3 is now at the level of scripted proving in KeYmaera X 4.4.3, while additional scripted functionality became available that is expected to trickle down to automation in future years.

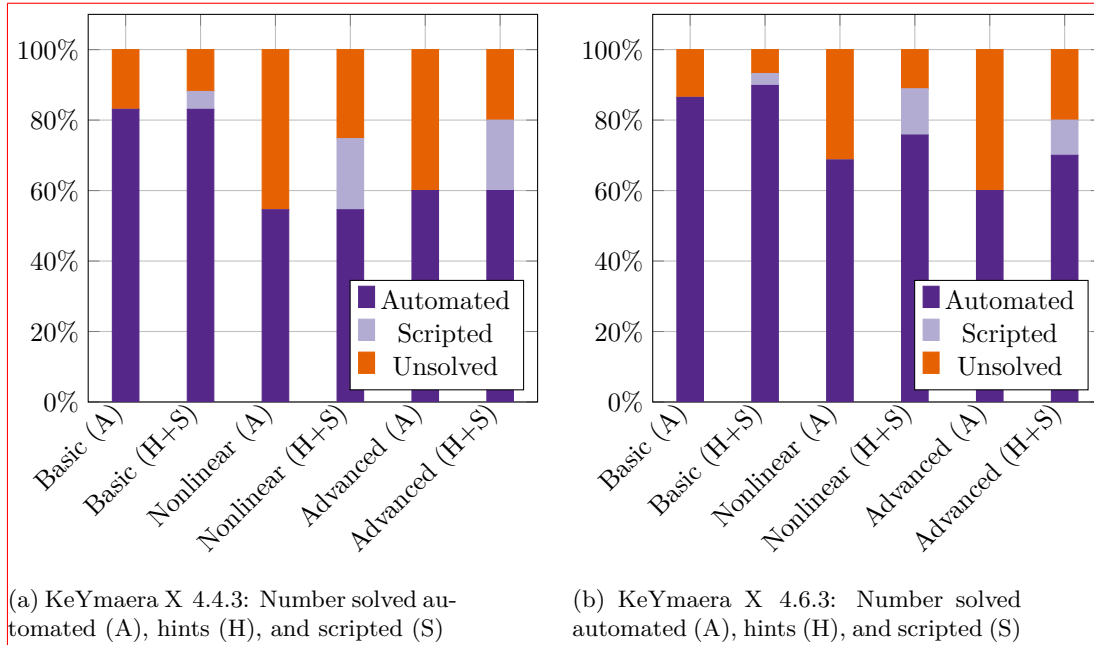


Figure 8: Result summary: KeYmaera X 4.6.3 automation is now at the level of KeYmaera X 4.4.3 scripting; additional scripting in KeYmaera X 4.6.3 increases the number of solvable examples.

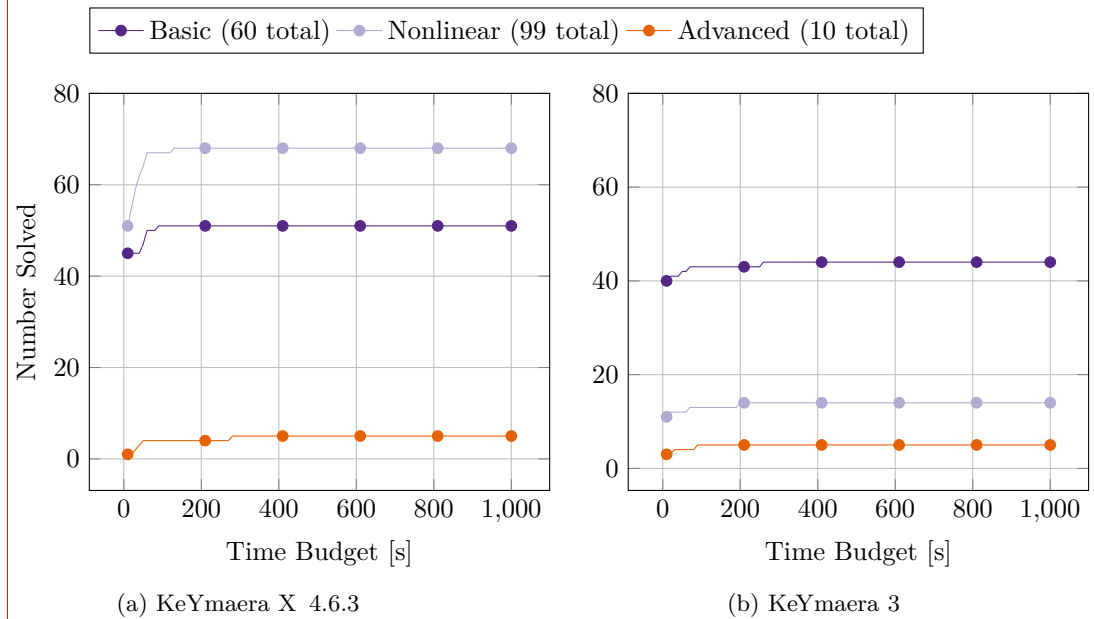


Figure 9: Result summary: Number of examples solvable fully automatically (A) with individual time budgets. KeYmaera X solves more examples, especially among those with nonlinear dynamics.

Acknowledgments. This material is based upon work supported by the National Science Foundation under CNS-1739629, by the AFOSR under grant number FA9550-16-1-0288, and by the United States Air Force and DARPA under Contract No. FA8750-18-C-0092. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity. Yong Kiam Tan was also supported by A*STAR, Singapore. Bohua Zhan is supported by CAS Pioneer Hundred Talents Program under grant No. Y9RC585036. Xiangyu Jin, Shuling Wang, and Naijun Zhan are funded partly by NSFC under grant No. 61625206 and 61732001, by “973 Program” grant No. 2014CB340701 and by the CAS/SAFEA International Partnership Program for Creative Research Teams.

References

- [BLCP18a] Brandon Bohrer, Adriel Luo, Xue An Chuang, and André Platzer. CoasterX: A case study in component-driven hybrid systems proof automation. *IFAC-PapersOnLine*, 2018. Analysis and Design of Hybrid Systems ADHS.
- [BLCP18b] Brandon Bohrer, Adriel Luo, Xue An Chuang, and André Platzer. Coasterx: A case study in component-driven hybrid systems proof automation. In *6th IFAC Conference on Analysis and Design of Hybrid Systems, ADHS 2018, Oxford, UK, July 11-13, 2018*, pages 55–60, 2018.
- [BRV⁺17] Brandon Bohrer, Vincent Rahli, Ivana Vukotic, Marcus Völp, and André Platzer. Formally verified differential dynamic logic. In Yves Bertot and Viktor Vafeiadis, editors, *Certified Programs and Proofs - 6th ACM SIGPLAN Conference, CPP 2017, Paris, France, January 16-17, 2017*, pages 208–221, New York, 2017. ACM.
- [DCKB17] Adel Djaballah, Alexandre Chapoutot, Michel Kieffer, and Olivier Bouissou. Construction of parametric barrier functions for dynamical systems using interval analysis. *Automatica*, 78:287–296, 2017.
- [DGXZ17] Liyun Dai, Ting Gan, Bican Xia, and Naijun Zhan. Barrier certificates revisited. *J. Symb. Comput.*, 80:62–86, May 2017.
- [DHO03] Werner Damm, Hardi Hungar, and Ernst-Rüdiger Olderog. On the verification of cooperating traffic agents. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *FMCO*, volume 3188 of *LNCS*, pages 77–110. Springer, 2003.
- [DLA06] Freddy Dumortier, Jaume Llibre, and Joan C Artés. *Qualitative Theory of Planar Differential Systems*. Springer, 2006.
- [DPR05] Werner Damm, Guilherme Pinto, and Stefan Ratschan. Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. In Doron A. Peled and Yih-Kuen Tsay, editors, *ATVA*, volume 3707 of *LNCS*, pages 99–113. Springer, 2005.
- [ERT02] ERTMS User Group. UNISIG: ERTMS/ETCS system requirements specification. <http://www.era.europa.eu>, 2002. Version 2.2.2.
- [FMBP17] Nathan Fulton, Stefan Mitsch, Brandon Bohrer, and André Platzer. Bellerophon: Tactical theorem proving for hybrid systems. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *ITP*, volume 10499 of *LNCS*, pages 207–224. Springer, 2017.
- [FMQ⁺15] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In Amy Felty and Aart Middeldorp, editors, *CADE*, volume 9195 of *LNCS*, pages 527–538, Berlin, 2015. Springer.
- [He94] J. He. From CSP to hybrid systems. In *A Classical Mind, Essays in Honour of C.A.R. Hoare*, pages 171–189. Prentice Hall International (UK) Ltd., 1994.

- [HHMW00] Thomas A. Henzinger, Benjamin Horowitz, Rupak Majumdar, and Howard Wong-Toi. Beyond HYTECH: hybrid systems analysis using interval numerical methods. In Nancy A. Lynch and Bruce H. Krogh, editors, *HSCC*, volume 1790 of *LNCS*, pages 130–144. Springer, 2000.
- [LLQ⁺10] J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, and L. Zou. A calculus for hybrid CSP. In *APLAS 2010*, volume 6461 of *LNCS*, pages 1–15. Springer, 2010.
- [LZZ11] Jiang Liu, Naijun Zhan, and Hengjun Zhao. Computing semi-algebraic invariants for polynomial dynamical systems. In Samarjit Chakraborty, Ahmed Jerraya, Sanjoy K. Baruah, and Sebastian Fischmeister, editors, *EMSOFT*, pages 97–106. ACM, 2011.
- [Man93] Yiu-Kwong Man. Computing closed form solutions of first order odes using the prelle-singer procedure. *Journal of Symbolic Computation*, 16(5):423–443, 1993.
- [MF01] Mieke Massink and Nicoletta De Francesco. Modelling free flight with collision avoidance. In *ICECCS*, pages 270–280. IEEE Computer Society, 2001.
- [MGVP17] Stefan Mitsch, Khalil Ghorbal, David Vogelbacher, and André Platzer. Formal verification of obstacle avoidance and navigation of ground robots. *I. J. Robotics Res.*, 36(12):1312–1340, 2017.
- [MST⁺18] Stefan Mitsch, Andrew Sogokon, Yong Kiam Tan, André Platzer, Hengjun Zhao, Xiangyu Jin, Shuling Wang, and Naijun Zhan. ARCH-COMP18 category report: Hybrid systems theorem proving. In Goran Frehse, Matthias Althoff, Sergiy Bogomolov, and Taylor T. Johnson, editors, *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems, ARCH@ADHS 2018, Oxford, UK, July 13, 2018*, volume 54 of *EPiC Series in Computing*, pages 110–127. EasyChair, 2018.
- [NWP02] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer, Berlin, Heidelberg, 2002.
- [PC09a] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des.*, 35(1):98–120, 2009. Special issue for selected papers from CAV’08.
- [PC09b] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In Ana Cavalcanti and Dennis Dams, editors, *FM*, volume 5850 of *LNCS*, pages 547–562, Berlin, 2009. Springer.
- [PJ04] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *International Workshop on Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.
- [PJP07] Stephen Prajna, Ali Jadbabaie, and George J. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE T. Automat. Contr.*, 52(8):1415–1429, 2007.
- [PKV09] Erion Plaku, Lydia E. Kavradi, and Moshe Y. Vardi. Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Form. Methods Syst. Des.*, 34(2):157–182, 2009.
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008.
- [Pla10a] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010.
- [Pla10b] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010.
- [Pla12a] André Platzer. Logics of dynamical systems. In *LICS*, pages 13–24, Los Alamitos, 2012. IEEE.
- [Pla12b] André Platzer. The structure of differential invariants and differential cut elimination. *Log. Meth. Comput. Sci.*, 8(4:16):1–38, 2012.

- [Pla17] André Platzer. A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.*, 59(2):219–265, 2017.
- [Pla18] André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, Switzerland, 2018.
- [PQ08] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 171–178, Berlin, 2008. Springer.
- [PQ09] André Platzer and Jan-David Quesel. European Train Control System: A case study in formal verification. In Karin Breitman and Ana Cavalcanti, editors, *ICFEM*, volume 5885 of *LNCS*, pages 246–265, Berlin, 2009. Springer.
- [PT18] André Platzer and Yong Kiam Tan. Differential equation axiomatization: The impressive power of differential ghosts. In Anuj Dawar and Erich Grädel, editors, *LICS*, New York, 2018. ACM.
- [QML⁺16] Jan-David Quesel, Stefan Mitsch, Sarah Loos, Nikos Aréchiga, and André Platzer. How to model and prove hybrid systems with KeYmaera: A tutorial on safety. *STTT*, 18(1):67–91, 2016.
- [SC⁺13] Sriram Sankaranarayanan, Xin Chen, et al. Lyapunov function synthesis using handelmann representations. *IFAC Proceedings Volumes*, 46(23):576–581, 2013.
- [SGJ16] Andrew Sogokon, Khalil Ghorbal, and Taylor T Johnson. Non-linear continuous systems for safety verification (benchmark proposal). In *ARCH@CPSWeek*, volume 43, pages 42–51. EasyChair, 2016.
- [SGJP16] Andrew Sogokon, Khalil Ghorbal, Paul B. Jackson, and André Platzer. A method for invariant generation for polynomial continuous systems. In Barbara Jobstmann and K. Rustan M. Leino, editors, *VMCAI*, volume 9583 of *LNCS*, pages 268–288. Springer, 2016.
- [SGS14] Mohamed Amin Ben Sassi, Antoine Girard, and Sriram Sankaranarayanan. Iterative computation of polyhedral invariants sets for polynomial dynamical systems. In *CDC*, pages 6348–6353. IEEE, 2014.
- [SV87] Michael A. Savageau and Eberhard O. Voit. Recasting nonlinear differential equations as S-systems: a canonical nonlinear form. *Mathematical Biosciences*, 87(1):83 – 115, 1987.
- [TPL⁺96] Claire Tomlin, George J. Pappas, John Lygeros, Datta N. Godbole, and Shankar Sastry. Hybrid control models of next generation air traffic management. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems IV*, volume 1273 of *LNCS*, pages 378–404. Springer, 1996.
- [TPS98] Claire Tomlin, George J. Pappas, and Shankar Sastry. Conflict resolution for air traffic management: a study in multiagent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, Apr 1998.
- [WZZ15] S. Wang, N. Zhan, and L. Zou. An improved HHL prover: an interactive theorem prover for hybrid systems. In *ICFEM 2015*, volume 9407 of *LNCS*, pages 382–399. Springer, 2015.
- [ZH04] C. Zhou and M.R. Hansen. *Duration Calculus — A Formal Approach to Real-Time Systems*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag Berlin Heidelberg, 2004.
- [ZLW⁺14] Liang Zou, Jidong Lv, Shuling Wang, Naijun Zhan, Tao Tang, Lei Yuan, and Yu Liu. Verifying Chinese train control system under a combined scenario by theorem proving. In Ernie Cohen and Andrey Rybalchenko, editors, *VSTTE 2013*, volume 8164 of *LNCS*, pages 262–280. Springer, 2014.
- [ZWR96] Chaochen Zhou, Ji Wang, and Anders P. Ravn. A formal description of hybrid systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 511–530. Springer Berlin Heidelberg, 1996.
- [ZWZ16] Naijun Zhan, Shuling Wang, and Hengjun Zhao. *Formal Verification of Simulink/Stateflow*

- Diagrams - A Deductive Approach*. Springer, 2016.
- [YZZ⁺14a] Hengjun Zhao, Mengfei Yang, Naijun Zhan, Bin Gu, Liang Zou, and Yao Chen. Formal verification of a descent guidance control program of a lunar lander. In Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun, editors, *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, volume 8442 of *Lecture Notes in Computer Science*, pages 733–748. Springer, 2014.
- [YZZ⁺14b] Hengjun Zhao, Mengfei Yang, Naijun Zhan, Bin Gu, Liang Zou, and Yao Chen. Formal verification of a descent guidance control program of a lunar lander. In Cliff Jones, Pekka Pihlajasaari, and Jun Sun, editors, *FM 2014: Formal Methods*, pages 733–748, Cham, 2014. Springer International Publishing.
- [ZZW⁺13] Liang Zou, Naijun Zhan, Shuling Wang, Martin Fränzle, and Shengchao Qin. Verifying Simulink diagrams via a Hybrid Hoare Logic prover. In *EMSOFT*, pages 1–10. IEEE Press, 2013.
- [ZZWF15] Liang Zou, Naijun Zhan, Shuling Wang, and Martin Fränzle. Formal verification of Simulink/Stateflow diagrams. In *ATVA*, volume 9346 of *LNCS*, pages 464–481. Springer, 2015.

A Specification of Machines

A.1 M_k

- Processor: Intel Xeon E5-1650 v2 @ 3.5GHz x 6
- Memory: 32GB
- Average CPU Mark on www.cpubenchmark.net: 12695 (full), 1990 (single thread)

A.2 M_{hhl}

- Processor: Intel Core i7-4790 CPU @ 3.6GHz
- Memory: 16GB
- Average CPU Mark on www.cpubenchmark.net: 9995 (full), 2284 (single thread)