EPiC Series in Computing
Volume 80, 2021, Pages 120–131
8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)

ARCH-COMP21 Category Report: Hybrid Systems Theorem Proving

Stefan Mitsch[1]
Xiangyu Jin[2], Bohua Zhan[2], Shuling Wang[2], and Naijun Zhan[2]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
`smitsch@cs.cmu.edu`
[2] State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences
`{jinxy,bzhan,wangsl,znj}@ios.ac.cn`

Abstract

This paper reports on the Hybrid Systems Theorem Proving (**HSTP**) category in the ARCH-COMP Friendly Competition 2021. The characteristic features of the HSTP category remain as in the previous editions [MST+18, MST+19, MMJ+20], it focuses on flexibility of programming languages as structuring principles for hybrid systems, unambiguity and precision of program semantics, and mathematical rigor of logical reasoning principles. The benchmark set includes nonlinear and parametric continuous and hybrid systems and hybrid games, each in three modes: fully automatic verification, semi-automatic verification from proof hints, proof checking from scripted tactics. This instance of the competition introduces extensions to the scripting language, a comparison of the influence of arithmetic backend versions on verification performance in KeYmaera X, as well as improvements in the HHL Prover.

1 Introduction

This report summarizes the experimental results of the Hybrid Systems Theorem Proving (HSTP) category in the ARCH-COMP21 friendly competition, focusing on extensions to the scripting language used in the proof checking evaluation mode, a comparison of the influence of arithmetic backend versions on verification performance in KeYmaera X, as well as improvements in the HHL Prover. Details on the benchmark sets and the evaluation modes can be found in previous editions of the HSTP category [MST+18, MST+19, MMJ+20]. The 214 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, based on [SMT+19, SMT+20].
- Hybrid games: small-scale examples with adversary dynamics in differential dynamic game logic.

- Hybrid systems case studies: hybrid systems models and specifications at scale to test for application scalability and efficiency, based on [PQ09, PC09, ZLW+13, ZYZ+14, MGVP17, BLCP18].

In each of these categories, tools can select the degree of automation 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 (d$\mathcal{L}$) [Pla08, Pla17]. 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 (d$\mathcal{L}$) [Pla08, Pla17] which has axioms and an unambiguous semantics available [BRV+17] in KeYmaera 3, KeYmaera X, Isabelle/HOL, and Coq. A tutorial on the modeling principles in d$\mathcal{L}$ can be found in [QML+16], details on the ASCII syntax are in [MMJ+20]. Here, we list the extensions over [FMBP17] to the scripting language that are introduced in this instance of the competition.

Scripting Language ASCII syntax. The KeYmaera X ASCII syntax is illustrated in the example below, with tactics using position identifiers to refer to formulas and terms in a sequent. The order of branches is important, e.g., after `loop`, the first branch operates on the initial case of the induction proof, the second branch on the postcondition, and the third branch on the induction step.

```
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 & 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"
  auto
End.

Tactic "Scripted proof in Bellerophon tactic language"
  implyR(1) ; loop("v>=0", 1) ; <(                    /* < splits into separate branches */
    id,                                                /* 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) ; auto,                   /* decompose some steps then ask auto */
      assignb(1) ; QE                                  /* assignment, then real arithmetic */
    )
  )
End.

End. /* end of ArchiveEntry */
```

The extended proof script language introduces robustness to changes in the order of formulas in sequents and to changes in the order of branches: For example, in the tactic script below, the search locator `implyR('R==...)` in line 2 refers to a formula in the alternatives to prove (right-hand side of the sequent turnstile), as opposed to `implyR(1)` in the tactic above, which refers to a fixed position in the sequent and is vulnerable to changes in formula ordering in case automated tactics progress in proofs differently across prover versions. Tactics can use marker `#` to refer to sub-formulas or terms: e.g., the locator `simplify('R=="x>=0 & #y+0#>=x")` applies tactic `simplify` to term `y+0`. Branch labels (e.g., `"Init"` in line 4) now unambiguously identify on which of the branches to apply some tactic. The branch labels themselves are created by tactics, e.g., tactic `loop` generate labels `Init`, `Post`, and `Step` in lines 4, 6, and 8, respectively, while tactic `andR` generates labels according to the formula it was applied to.

```
 1 Tactic "Scripted proof in extended Bellerophon tactic language"
 2 implyR('R=="v>=0&b()>0->[{{?v<=5;a:=5;++a:=-b();}{x'=v,v'=a&v>=0}}*]v>=0");
 3 loop("v>=0", 'R=="[{{?v<=5;a:=5;++a:=-b();}{x'=v,v'=a&v>=0}}*]v>=0"); <(
 4   "Init":
 5     id,
 6   "Post":
 7     QE,
 8   "Step":
 9     composeb('R=="[{?v<=5;a:=5;++a:=-b();}{x'=v,v'=a&v>=0}]v>=0");
10     solve('R=="[?v<=5;a:=5;++a:=-b();]#[{x'=v,v'=a&v>=0}]v>=0#");
11     choiceb('R=="[?v<=5;a:=5;++a:=-b();]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=
         ↪ t_->a*s_+v>=0)->a*t_+v>=0)");
12     andR('R=="[?v<=5;a:=5;]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v>=0)
         ↪ ->a*t_+v>=0)&[a:=-b();]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*
         ↪ s_+v>=0)->a*t_+v>=0)"); <(
13       "[?v<=5;a:=5;]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v>=0)->a*t_+
           ↪ v>=0)":
14         composeb('R=="[?v<=5;a:=5;]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*
             ↪ s_+v>=0)->a*t_+v>=0)");
15         testb('R=="[?v<=5;][a:=5;]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*
             ↪ s_+v>=0)->a*t_+v>=0)");
16         auto,
17       "[a:=-b();]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v>=0)->a*t_+v
           ↪ >=0)":
18         assignb('R=="[a:=-b();]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v
             ↪ >=0)->a*t_+v>=0)");
19         QE
20     )
21 )
22 End.
```

Proof scripts expressed in the locator style `'R==...` and `'L==...` are more explicit about how they operate on specific examples, but such scripts do not transfer well between different examples. In order to generalize a script for applicability to other examples, or to increase robustness to changes in the input model, locators `'R~=...` and `'L~=...` use unification to decide where to apply tactics. For example, the tactic below uses unification to become applicable to a wider variety of examples, but further scripting language extensions are needed to be truly generalizable to any example of the shape $init(v) \rightarrow [((acc \cup brake); plant)^*]p(v)$.

```
Tactic "Scripted proof with unification in search locators"
implyR('R~="p() -> q()");
loop("v>=0", 'R~="[{drive;plant;}*]p(v)"); <(
  "Init":
    id,
  "Post":
    QE,
  "Step":
    composeb('R~="[drive;plant;]p(v)");
    solve('R~="[drive;]#[plant;]p(v)#");
    choiceb('R~="[acc;++brake;]solution(a,v)");
    andR('R~="[acc;]solution(a,v) & [brake;]solution(a,v)"); <(
      /* future extension: unification in branch labels */
      "[?v<=5;a:=5;]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v>=0)->a*t_+
          ↪ v>=0)":
        composeb('R~="[?v<=5;a:=5;]solution(a,v)");
        testb('R~="[?v<=5;][a:=5;]solution(a,v)");
        auto,
      "[a:=-b();]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v>=0)->a*t_+v
          ↪ >=0)":
        assignb('R~="[a:=-b();]solution(a,v)");
        QE
    )
)
End.
```

3 Participating Tools

KeYmaera X. KeYmaera X [FMQ+15] is a theorem prover for the hybrid systems logic differential dynamic logic ($\mathsf{d}\mathcal{L}$). It implements the uniform substitution calculus of $\mathsf{d}\mathcal{L}$ [Pla17].[1] A comparison of the internal reasoning principles in the KeYmaera family of provers and a discussion of their relative benefits and drawbacks is in [MP20]. KeYmaera X supports systems with nondeterministic discrete jumps, nonlinear differential equations, nondeterministic input, and it provides invariant construction [SMT+19, SMT+20] and proving techniques for differential equations [SGJP16, PT18, PT20]. Unlike numerical hybrid systems reachability analysis tools, KeYmaera X also supports unbounded initial sets and unbounded time analysis. KeYmaera X participates in v4.8.0 (2020) and v4.9.4 (2021), the specific commits of the Github repository `https://github.com/LS-Lab/KeYmaeraX-release` used in the competition are tagged `arch2020` for KeYmaera X v4.8.0 and `arch2021` for KeYmaera X v4.9.4.

Major improvements of this competition version over the 2020 edition include a simplified automation implementation, support for the scripting language extensions described above, proof management on top of uniform substitution [Mit21], a parallel quantifier elimination tactic that attempts to utilize all available Mathematica Kernels concurrently to find the fastest succeeding among several variations of a formula (e.g., the tactic propositionally splits a large single quantifier elimination attempt into several smaller ones and tries the large formula concurrently with the several smaller ones), and numerous smaller stability and performance improvements. Due to an incompatibility of JLink in the latest Wolfram Engine versions 12.2 and 12.3 with the libraries in the repeatability operating system, the repeatability evaluation focuses purely on performance with the Z3 arithmetic backend and compares verification performance over

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

several versions of Z3; results are reported in the Appendix. As a consequence, Pegasus invariant generation is not available in the repeatability package and proof automation is severely limited, especially in the nonlinear sub-category. The performance results reported here are obtained on MacOS, for KeYmaera X v4.8.0 with Mathematica 12.1 and Pegasus invariant generator [SMT+19] on Matlab 2019b with SOSTools 3.03, and for KeYmaera X v4.9.4 with Mathematica 12.3 and extended Pegasus invariant generator [SMT+20] on Matlab 2021a with SOSTools 3.04.

HHL Prover. HHL Prover [WZZ15] is an interactive theorem prover for verifying hybrid systems modelled by Hybrid CSP (HCSP) [He94, ZWR96]. HCSP is an extension of CSP by introducing differential equations for modeling continuous evolutions and interrupts for modeling interaction between continuous and discrete dynamics.

This year, we implemented a new trace-based hybrid Hoare logic for reasoning about HCSP processes. This is an improvement and simplification over the trace-based logic presented last year. Traces for both sequential and parallel HCSP processes are represented as lists of *trace blocks*. There are two types of trace blocks: ODE blocks and communication blocks. ODE blocks specify evolution of the process over an interval of time, consisting of duration of the interval, the state of the process as a function of time, and a set of communications that are ready during the interval. Communication blocks are of three types: input, output, and IO. Input and output blocks specify an unmatched communication event, while IO blocks specify a matched communication event. All three types of events also specify the value that is communicated.

We defined big-step and small-step semantics of HCSP processes, and proved a notion of equivalence between these two semantics. The big-step semantics defines relation of the form $(c, s) \Rightarrow (s', tr)$, which means executing process c starting from state s results in final state s', and where tr is the list of trace blocks produced. The small-step semantics defines relation of the form $(c, s) \xrightarrow{e} (c', s')$, which means executing process c for one step starting from state s results in state s, with c' being the program that remains to be executed, and e is the event produced (either the empty event τ or a trace block).

An assertion is a predicate over pairs of state and trace. The Hoare triple (for partial correctness) is defined as follows:

$$\{P\}\, c\, \{Q\} \Longleftrightarrow \forall s\ tr\ s'\ tr'.\, P(s, tr) \longrightarrow (c, s) \Rightarrow (s', tr') \longrightarrow Q(s', tr^\frown tr'),$$

where $\cdot^\frown\cdot$ denotes the concatenation of two traces. We prove Hoare triples for basic commands, as well as rules for reasoning about differential equations. Moreover, there are rules about synchronization of two traces, enabling compositional reasoning about parallel HCSP processes.

The new system is tested on the basic benchmarks, as well as two control-plant models.

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 4 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, hybrid game examples to test adversarial dynamics, and hybrid systems case studies to test for prover scalability.

Experimental setup. KeYmaera X (in automated (A), hints (H), and scripted (S) mode) participated on all benchmark sets and was executed on a 2013 Mac Pro with 6-core Intel Xeon E5 3.5GHz and 28GB memory. 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. 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, which was kept consistent with [MMJ+20] to make results comparable. In the repeatability package, proof duration was shortened considerably over previous years in order to allow comparing several backend tool versions. The competition results are presented with *accumulated execution times* after examples are ranked according to their execution time.

4.1 Hybrid Systems Design Shapes

This category is unmodified from previous years, designed to test for basic verification features on simple examples.

KeYmaera X In KeYmaera X, proof attempts were aborted after a timeout of 300s, the performance results are summarized in Fig. 1. Owing to the simplistic nature of the examples in the hybrid systems design shapes sub-category, parallel quantifier elimination does not result in an overall performance gain compared to the previous results [MMJ+20]. Out of the 60 design shapes examples, KeYmaera X solves 54 fully automatic, 55 from proof hints, and 58 from proof scripts.

HHL Prover. The HHL Prover is tested on the 60 examples in the *basic* benchmark. We have successfully proved 49 of the 60 examples in Isabelle/HOL using our proof system. Since the benchmarks are originally formulated in terms of dynamic logic, some modifications are made to adapt it to a Hoare-logic style system. In particular, we added specific time constraints for evolution according to ODEs. Appearances of tests $?Q$ are replaced by equivalent conditional statements. Non-deterministic assignments $x := *$ are changed to assignments of arbitrary elements satisfying some property. We made these changes to stay as closed as possible to the original intent of the example. In particular, the method and difficulty of the proof are largely unchanged.

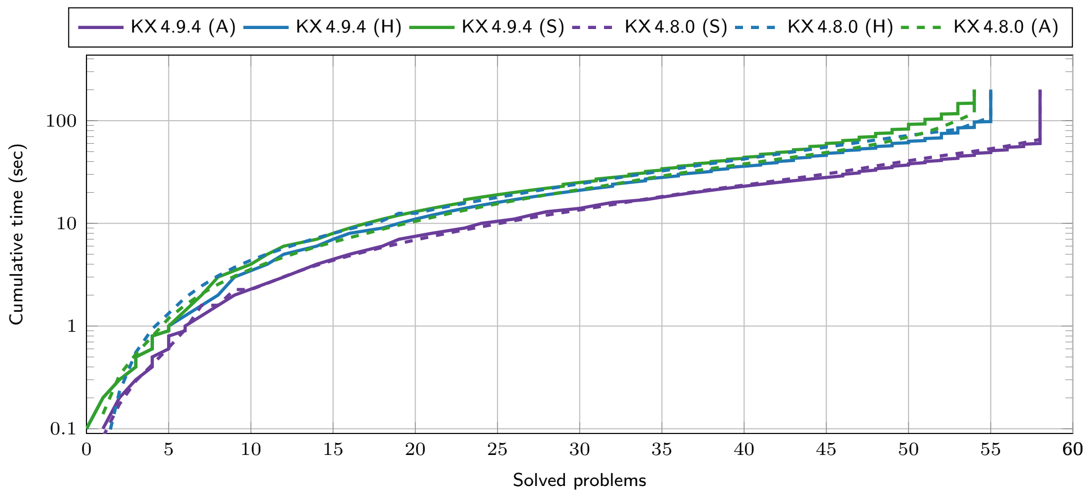


Figure 1: Cumulative time to solve fastest n basic problems in KeYmaera X (flatter and more is better); dedicated proof scripts gain some efficiency and solve more problems, full automation is almost on par with proof hints.

4.2 Nonlinear Continuous Models

The examples in this category remained unchanged from [MMJ$^+$20] for direct comparison of the verification performance with previous results; the examples test for pure continuous verification performance. Future competitions may additionally utilize the extended benchmark set of [SMT$^+$20].

Competition results. The participants in the Hybrid Systems Case Study category include KeYmaera X. Proof attempts in the nonlinear category were aborted after a timeout of 300 s.

KeYmaera X In KeYmaera X, proof attempts were aborted after a timeout of 300s, the performance results are summarized in Fig. 2. Out of the 141 nonlinear examples, KeYmaera X solves 92 fully automatic, 95 from proof hints, and 108 from proof scripts. Improved proof automation for nonlinear systems [SMT$^+$20] further closed the gap to scripted proofs: proving from hints in KeYmaera X 4.9.4 is now on par with scripting in KeYmaera X 4.8.0, while improvements in differential equation automation and utilizing quantifier elimination backends increased the number of examples solved from scripts within the 300 s timeout period.

4.3 Hybrid Games

Competition results. The participants in the Hybrid Games category include KeYmaera X. Owing to the simple nature of the games examples, KeYmaera X 4.9.4 solved all three examples in less than 4 s in scripted mode. The performance remained unchanged compared to the previous results [MMJ$^+$20]. The hybrid systems proof automation of KeYmaera X 4.9.4 is capable of solving two of the examples, but more games automation is needed to truly search for winning strategies in games.

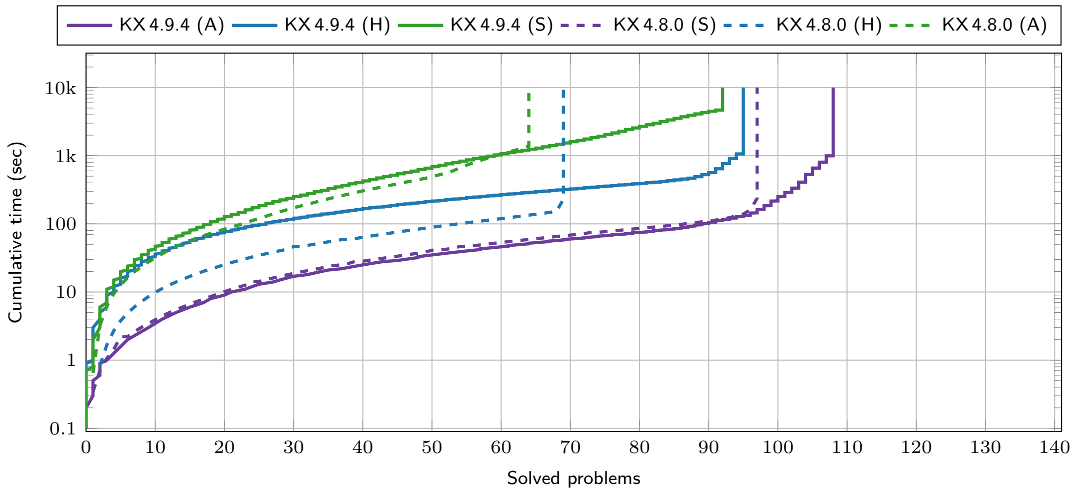


Figure 2: Cumulative time to solve fastest n nonlinear problems in KeYmaera X (flatter and more problems solved is better).

4.4 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 remained unchanged from [MST+19]. The benchmark examples[2] are inspired from prior case studies on train control [PQ09, ZLW+13], flight collision avoidance [PC09], robot collision avoidance [MGVP17], a lunar lander descent guidance protocol [ZYZ+14], and rollercoaster safety [BLCP18].

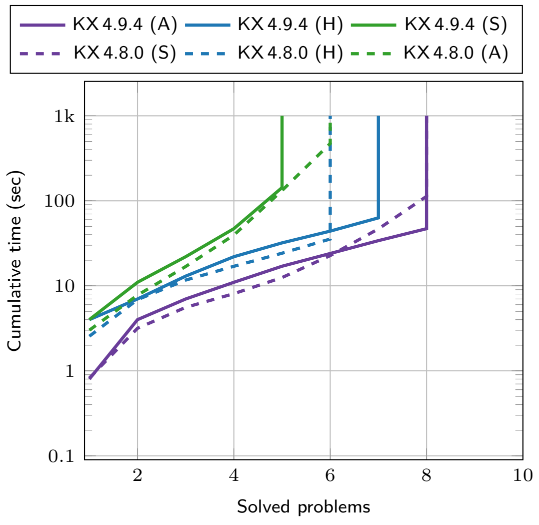


Figure 3: Cumulative time to solve fastest n case study problems in KeYmaera X (flatter and more problems solved is better).

KeYmaera X. Proof attempts in the case study benchmarks sub-category were aborted after 1500 s. Out of 10 examples, KeYmaera X solves 5 fully automatically, 7 from hints, and 8 from scripts (2 not attempted); the results are in Fig. 3. The overall performance remained comparable, with parallel quantifier elimination providing some performance gains on complex arithmetic goals, which results in an additional example solved from proof hints compared to the previous results [MMJ+20]. Simplifying full automation tactics for maintenance reasons, however, resulted in less examples being provable fully automatically.

[2]https://github.com/LS-Lab/KeYmaeraX-projects/blob/master/benchmarks/advanced.kyx

HHL Prover. We present two examples demonstrating verification of control-plant models using hybrid Hoare logic. The first example is a simple velocity control. The HCSP process is given as follows:

$$\begin{aligned} plant &::= (t := 0; (\dot{v} = a, \dot{t} = 1 \& t < 1); p2c!v; c2p?a)^* \\ control &::= (p2c?v; \text{if } v < 10 \text{ then } c2p!1 \text{ else } c2p!(-1))^* \\ system &::= plant \,\|\, control \end{aligned}$$

The second example is adapted from the Chinese train control system (CTCS) example [ZLW+13]. We mainly verify that the control law is able to keep the train from going beyond the stop point.

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 automation tactic simplifications, nonlinear invariant generator improvements, and concurrent arithmetic backend utilization make a difference on some examples and especially in pure continuous systems verification performance, but their potential is not yet truly realized in case study verification performance. Future competitions are planned to extend the case study sub-category to provide better assessment of verification performance on realistic examples, and to gain insight into potential proof automation to generalize the current specialized tactics and proof scripts from single example applicability to general-purpose proof automation. A related challenge for proof repeatability and transferability are timeouts used in proof automation to decide how long to explore specific proof alternatives, and overall proof timeouts as used in this competition.

Specific to KeYmaera X, the parallel quantifier elimination introduced in this year's competition opens up the possibility to explore alternatives in proof automation concurrently, as opposed to the current sequential proof exploration that executes alternative attempts in a heuristic order[3].

Acknowledgments. This material is based upon work supported 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. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force and DARPA. Bohua Zhan is supported by CAS Pioneer Hundred Talents Program under grant No. Y9RC585036. Xiangyu Jin, Bohua Zhan, Shuling Wang, and Naijun Zhan are funded partly by NSFC under grant No. 61625206, 61732001, and 62032024. We thank the entire Logical Systems Lab at Carnegie Mellon University for their many contributions and suggestions to KeYmaera X and its associated tools.

[3]Heuristic ordering of proof alternatives in automated tactics is inspired by usual proof attempts, but may not work well for all examples; quite frequently, automation may get stuck at an early alternative, while a later alternative would succeed. Similar observations can be made for generated loop and ODE invariants.

References

[BLCP18] 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.

[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.

[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.

[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.

[Mit21] Stefan Mitsch. Implicit and explicit proof management in keymaera x. In *6th Workshop on Formal Integrated Development Environment, Proceedings*, 2021.

[MMJ$^+$20] Stefan Mitsch, Jonathan Juli\'an Huerta Y Munive, Xiangyu Jin, Bohua Zhan, Shuling Wang, and Naijun Zhan. Arch-comp20 category report:hybrid systems theorem proving. In Goran Frehse and Matthias Althoff, editors, *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, volume 74 of *EPiC Series in Computing*, pages 153–174. EasyChair, 2020.

[MP20] Stefan Mitsch and André Platzer. A retrospective on developing hybrid system provers in the keymaera family - A tale of three provers. In Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle, and Mattias Ulbrich, editors, *Deductive Software Verification: Future Perspectives - Reflections on the Occasion of 20 Years of KeY*, volume 12345 of *Lecture Notes in Computer Science*, pages 21–64. Springer, 2020.

[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.

[MST$^+$19] Stefan Mitsch, Andrew Sogokon, Yong Kiam Tan, Xiangyu Jin, Bohua Zhan, Shuling Wang, and Naijun Zhan. ARCH-COMP19 category report: Hybrid systems theorem proving. In Goran Frehse and Matthias Althoff, editors, *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systemsi, part of CPS-IoT Week 2019, Montreal, QC, Canada, April 15, 2019*, volume 61 of *EPiC Series in Computing*, pages 141–161. EasyChair, 2019.

[PC09] 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.

[Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008.

[Pla17] André Platzer. A complete uniform substitution calculus for differential dynamic logic. *J.*

Autom. Reas., 59(2):219–265, 2017.

[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.

[PT20] André Platzer and Yong Kiam Tan. Differential equation invariance axiomatization. *J. ACM*, 67(1):6:1–6:66, 2020.

[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.

[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.

[SMT$^+$19] Andrew Sogokon, Stefan Mitsch, Yong Kiam Tan, Katherine Cordwell, and André Platzer. Pegasus: A framework for sound continuous invariant generation. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods - The Next 30 Years - Third World Congress, FM 2019, Porto, Portugal, October 7-11, 2019, Proceedings*, volume 11800 of *LNCS*, pages 138–157. Springer, 2019.

[SMT$^+$20] Andrew Sogokon, Stefan Mitsch, Yong Kiam Tan, Katherine Cordwell, and André Platzer. Pegasus: Sound continuous invariant generation. *Form. Methods Syst. Des.*, 2020. Special issue for selected papers from FM'19.

[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.

[ZLW$^+$13] 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, *Verified Software: Theories, Tools, Experiments - 5th International Conf., VSTTE 2013, Menlo Park, CA, USA, May 17-19, 2013, Revised Selected Papers*, volume 8164 of *LNCS*, pages 262–280. Springer, 2013.

[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 *LNCS*, pages 511–530. Springer, 1996.

[ZYZ$^+$14] 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 *LNCS*, pages 733–748. Springer, 2014.

A Repeatability Package: KeYmaera X 4.9.4 with Z3

Here, we compare the verification performance using different versions of Z3 as a backend for real arithmetic solving:

- Z3 4.6.0 (late 2017)
- Z3 4.8.4 (late 2018)
- Z3 4.8.7 (late 2019)
- Z3 4.8.10 (latest, early 2021)

The performance on the hybrid systems design shapes is summarized in Fig. 4, on nonlinear continuous examples in Fig. 5, and on case study benchmarks in Fig. 6.

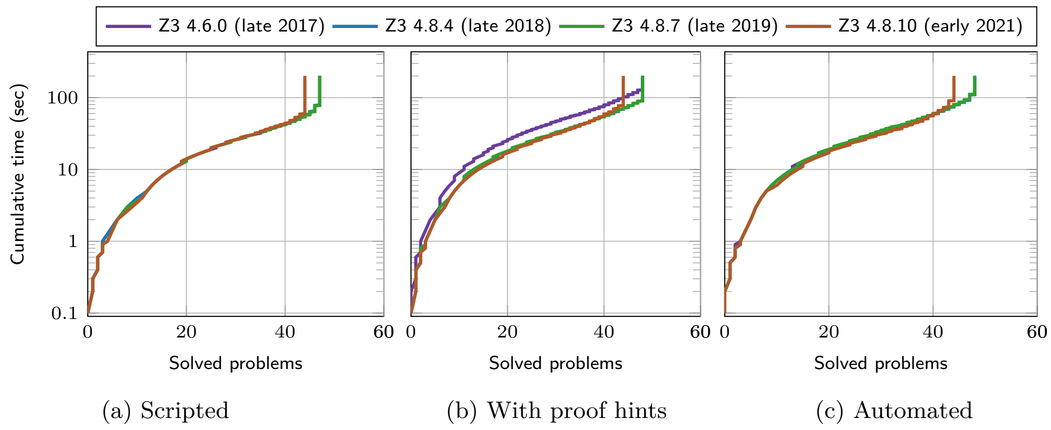


Figure 4: Hybrid systems design shapes: cumulative time to solve fastest n problems

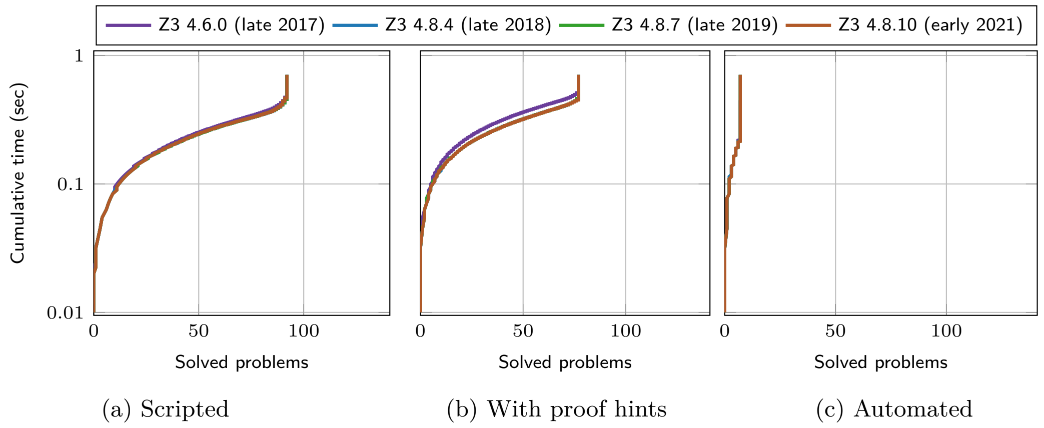
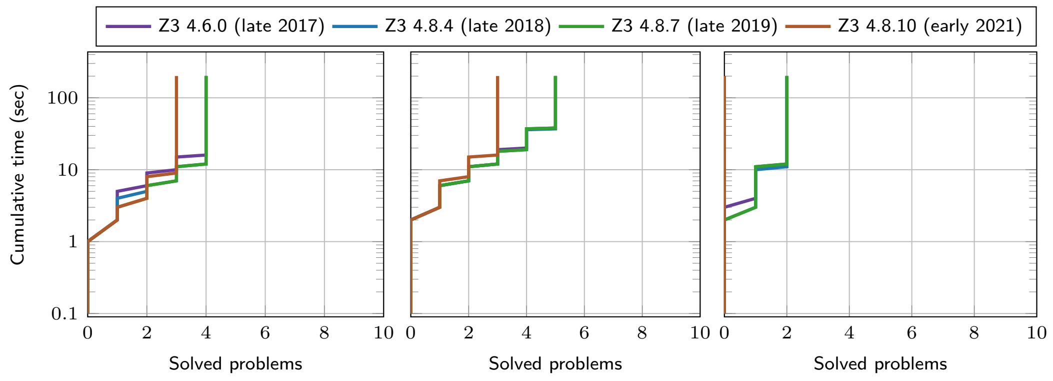


Figure 5: Nonlinear continuous systems: cumulative time to solve fastest n problems



(a) Scripted (b) With proof hints (c) Automated

Figure 6: Case study benchmarks: cumulative time to solve fastest n problems