EPiC Series in Computing
Volume 96, 2023, Pages 170–188
Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23)
EPiC Computing

ARCH-COMP23 Category Report: Hybrid Systems Theorem Proving

Stefan Mitsch[1], Huanhuan Sheng[2], Bohua Zhan[2], Shuling Wang[2], Simon Foster[3], and Jonathan Julián Huerta y Munive[4]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
smitsch@cs.cmu.edu
[2] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
{shenghh,bzhan,wangsl}@ios.ac.cn
[3] Department of Computer Science, University of York, UK
simon.foster@york.ac.uk
[4] Department of Computer Science, University of Copenhagen, Denmark
jonathan.munive@di.ku.dk

Abstract

This paper reports on the Hybrid Systems Theorem Proving (*HSTP*) category in the ARCH-COMP Friendly Competition 2023. The characteristic features of the HSTP category remain as in the previous edition [MZS+22]: HSTP 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 focuses on presenting the differences between the provers on a subset of the benchmark examples.

1 Introduction

This report summarizes the experimental results of the Hybrid Systems Theorem Proving (HSTP) category in the ARCH-COMP23 friendly competition, focusing on a feature comparison between the participating theorem provers. Details on the benchmark sets and the evaluation modes can be found in previous editions of the HSTP category [MMJ+20, MJZ+21, MZS+22]. The 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 [MGVP17].
- Hybrid systems from Simulink/Stateflow models: examples translated from Simulink/Stateflow models to verify.

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.

Benchmark examples in the hybrid systems design shapes, nonlinear continuous models, hybrid games and hybrid systems case study benchmarks are available at `https://github.com/LS-Lab/KeYmaeraX-projects/tree/master/benchmarks` and specified in differential dynamic logic (d$\mathcal{L}$) [Pla08, Pla17]. Benchmark examples for HHLPy, including the Simulink/Stateflow models and their translations to Hybrid CSP [ZWR95], are available at `https://gitee.com/bhzhan/mars/tree/master/hhlpy/examples/simulink`. An introduction to the problem format syntax is 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

Benchmarks in the hybrid systems design shapes, nonlinear continuous models, hybrid games and hybrid systems case study categories 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]. In this edition, we introduce libraries of pre-defined functions (e.g., `import kyx.math.abs`) [GTMP22]. Benchmarks in the hybrid systems design shapes and nonlinear continuous models are also translated to the HHLPy input language, along with the Simulink/Stateflow benchmarks. In the second subsection, we describe the input language for HHLPy in the competition.

Problem Format Example. 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.

```
ArchiveEntry "Benchmark Example 1"

Definitions                              /* definitions cannot change their value */
  import kyx.math.abs;                   /* import absolute value function */
  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 extended Bellerophon tactic language"
implyR('R=="v>=0&b()>0->[{{?v<=5;a:=5;++a:=-b();}{x'=v,v'=a&v>=0}}*]v>=0");
loop("v>=0", 'R=="[{{?v<=5;a:=5;++a:=-b();}{x'=v,v'=a&v>=0}}*]v>=0"); <( /* < splits branches */
  "Init":
    id,       /* initial case: shown with close by identity */
  "Post":
    QE,       /* postcondition: prove by real arithmetic QE */
  "Step":
    composeb('R=="[{?v<=5;a:=5;++a:=-b();}{x'=v,v'=a&v>=0}]v>=0");
    solve('R=="[?v<=5;a:=5;++a:=-b();]#[{x'=v,v'=a&v>=0}]v>=0#");
    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)");
    /* separate controller branches */
    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)"); <(
      "[?v<=5;a:=5;]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v>=0)->a*t_+v>=0)":
        /* decompose some steps then ask auto */
        composeb('R=="[?v<=5;a:=5;]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v>=0)->
            ↪ a*t_+v>=0)");
        testb('R=="[?v<=5;][a:=5;]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v>=0)->a*
            ↪ t_+v>=0)");
        auto,
      "[a:=-b();]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v>=0)->a*t_+v>=0)":
        /* assignment, then real arithmetic */
        assignb('R=="[a:=-b();]\forall t_ (t_>=0->\forall s_ (0<=s_&s_<=t_->a*s_+v>=0)->a*t_
            ↪ +v>=0)");
        QE
    )
)
End.

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

Input Language for HHLPy Benchmarks in the Hybrid Systems from Simulink/Stateflow category are modeled using Hybrid CSP, with properties specified as Hoare triples in Hybrid Hoare Logic. Both Hybrid CSP programs and properties as Hoare triples can be written using an ASCII syntax, as illustrated below. They are composed of pre-conditions, programs and post-conditions. The program is annotated with invariants and rules for proof.

```
# ArchiveEntry Benchmark Example 2

pre;                                   # pre-conditon
t := 0;                                # Assignment command
x := 0;
{                                      # braces {} group programs
  Chart_A_done := 0;
  if (t >= 1) {                        # If command
    t := 0;
    x := 0;
    Chart_A_done := 1;
  }
  Chart_ret := Chart_A_done;
  {x_dot = 1, t_dot = 1 & t < 1}       # differential equation systems
```

```
    invariant [x == t]{di};                # differential  equation invariant, with proof rule
}*                                         # loop repeats
  invariant [x == t] [0 <= x] [x <= 1];    # loop invariant
post [0 <= x] [x <= 1];                    # post−condition
```

The pre-condition is true and the post-condition is $0 \leq x \wedge x \leq 1$ in the above example. In the program, t and x are assigned as 0, followed by a loop command. The invariants of the loop are $x == t$, $0 \leq x$ and $x \leq 1$. A differential equation is in the loop with invariant $x == t$ proved by the rule dI (differential invariant).

3 Participating Tools

KeYmaera X. KeYmaera X [FMQ+15] is a theorem prover for the hybrid systems logic differential dynamic logic (dL). It implements the uniform substitution calculus of dL [Pla17]. A comparison of the internal reasoning principles in the KeYmaera family of provers with a discussion of their relative benefits and drawbacks is in [MP20], and model structuring and proof management on top of uniform substitution is discussed in [Mit21]. KeYmaera X supports systems with nondeterministic discrete jumps, nonlinear differential equations, nondeterministic inputs, and allows defining functions implicitly through their characterizing differential equations [GTMP22]. It provides invariant construction and proving techniques for differential equations [SMT+21, PT20], and stability verification techniques for switched systems [TMP22]. To discharge proof obligations in real arithmetic, KeYmaera X interacts with trusted backend procedures for quantifier elimination (Z3, Wolfram Mathematica, Wolfram Engine); a verified backend procedure based on virtual term substitution is under development [SCMP21]. Proofs in KeYmaera X can be conducted interactively [MP16], steered with tactics [FMBP17], or attempted fully automatic. Compared to previous editions of this benchmark, KeYmaera X development focused on introducing a definitions mechanism for functions [GTMP22].

HHL Prover/HHLPy. HHL Prover is a verification tool for hybrid systems modeled by Hybrid CSP (HCSP) [He94, ZWR96], implemented in Isabelle/HOL. HCSP is an extension of CSP by introducing differential equations for modeling continuous evolution and interrupts for modeling interaction between continuous and discrete dynamics. The proof system of HHL Prover is Hybrid Hoare Logic (HHL) [LLQ+10].

HHLPy is a new verification tool for HCSP, that provides a friendlier web-based user interface. Currently it handles only the sequential fragment of HCSP, with reasoning rules similar to those in $\mathsf{d}\mathcal{L}$. We briefly introduce each of the two tools in the following paragraphs.

HHL Prover HHL Prover [WZZ15] is an interactive theorem prover for verifying hybrid systems modeled by Hybrid CSP (HCSP). We use the trace-based hybrid Hoare logic for reasoning about HCSP processes as in 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.

HHLPy HHLPy is a theorem prover with a friendlier user interface, currently for verifying sequential HCSP programs only. It is implemented using Python and JavaScript. The sequential fragment of HCSP contains ODEs with domain boundary, but not communication, interrupts, and parallel processes. Extending HHLPy to handle the full HCSP language is left for future work. Given a sequential HCSP process P, a specification takes the form of Hoare triple, $\{Pre\}P\{Post\}$, where Pre and $Post$ are pre-/post-conditions in first-order logic.

To reason about differential equations, HHLPy makes use of a set of proof rules that are inspired by d$\mathcal{L}$ [Pla10, Pla11, PT18], but adapted to the semantics of sequential HCSP. The differential weakening (dW) rule reduces a Hoare triple concerning ODEs to an invariant triple of the ODE and some verification conditions. Invariant triple is in the form of $[\![P]\!]\langle\dot{x} = e\rangle[\![Q]\!]$, whose semantics is roughly stated as follows: for any solution to the differential equation $\dot{x} = e$, if Q is satisfied at beginning and P is satisfied throughout, then Q is satisfied throughout. Rules such as differential invariant (dI), differential cut (dC), Darboux's rule (dbx) and barrier certificates (barrier), many of which borrowed from differential dynamic logic, are then used to prove invariant triples.

HHLPy stores proof information after the corresponding assertion (post-condition, invariant), so that the user can still reuse proofs when they modify the program or the assertions slightly. Specifically, proof rules are stored after the corresponding invariants, and proof methods for proving verification conditions, for example, Z3 or Wolfram Engine, are stored after the assertion that generates the corresponding verification condition. Sometimes one assertion corresponds to several verification conditions. HHLPy also proposed a labeling system to distinguish these verification conditions.

IsaVODEs. **Isa**belle **V**erification with **O**rdinary **D**ifferential **E**quation**s** [FHGS21] is an extension of a hybrid systems verification component [HS22, Hue19] with lenses from the Unifying Theories of Programming (UTP) framework [HH98, FBC+20]. IsaVODEs shallowly represents programs as state transformers and propositions as predicates, while lenses algebraically characterise access and mutation of state. Proving soundness in Isabelle/HOL of Hoare logic and weakest liberal precondition (wlp) laws makes them available for verification with IsaVODEs. In particular, IsaVODEs provides its own wlp-rules for reasoning about hybrid programs. It also offers d$\mathcal{L}$-like syntax and all the major d$\mathcal{L}$-rules. Specifically, IsaVODEs now includes rules for reasoning with forward diamonds and it can manipulate differential invariants with cuts, ghosts, and inductions in the style of d$\mathcal{L}$. Since IsaVODEs is only restricted to Isabelle's higher order logic (HOL), it also offers support for direct reasoning with transcendental functions like exponentials, sines and cosines. Similarly, it can encode linear systems of ODEs as operations between matrices and vectors [Hue20a, Hue20b]. IsaVODEs also provides more automation than its predecessor verification components [HS22, FHS20]. It has Eisbach-methods for automatically discharging common differential induction and Hoare logic arguments, and it adds support for manipulating real arithmetic expressions. Through the Wolfram engine, IsaVODEs can suggest solutions to systems of ODEs to aid in the verification process. We use Isabelle's pdf-generation tool to produce a proof-document with our solutions to the competition's problems in our reproducibility package. Recent IsaVODEs developments are available online[1,2].

[1] https://github.com/isabelle-utp/Hybrid-Verification
[2] https://isabelle-utp.york.ac.uk/paradigms/cyber-physical

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 5 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, hybrid systems case studies to test for prover scalability, and a new category for hybrid systems from Simulink/Stateflow models.

Experimental Setup. IsaVODEs participated in scripted mode in the hybrid systems design shapes problems and in the European train control System case study benchmark. The IsaVODEs setup was an 8 core Apple M1 machine with a 16 GB memory.

HHLPy participated in three benchmark sets, which are hybrid systems design shapes, nonlinear continuous models and hybrid systems from Simulink/Stateflow models. The performance results were obtained on Windows 11, with Z3-solver 4.8.12.0 and Wolfram Engine 13.0, on the machine with 8-core Intel(R) Core(TM) i5-1035G4 CPU @ 1.10GHz and 16 GB memory.

HHL Prover participated in hybrid systems design shapes. The results were obtained on Windows 10, with Isabelle2020 and afp-2020-12-22 on the machine with Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz.

In this edition, KeYmaera X participated in scripted mode in the design shapes, games, nonlinear continuous models, and case study benchmarks categories. Performance in hints and automated mode are in previous editions [MZS+22]. The performance results reported here are obtained on the repeatability evaluation system `arch.repeatability.cps.cit.tum.de`. By moving to the repeatability system, the setup this year includes only Z3 as a backend. As a consequence, KeYmaera X focused on proof search and proof checking, but did not benchmark its invariant generation capabilities.

4.1 Hybrid Systems Design Shapes

This category is designed to test for basic verification features on simple examples. 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 $\mathsf{d}\mathcal{L}$ tutorial examples [Pla12] ranging from basic time-triggered motion control to model-predictive control.

STTT Tutorial 12 $\mathsf{d}\mathcal{L}$ 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.

Modeling Comparison This year, we add a new modeling comparison problem from [GTMP22] using trigonometric functions, see Section 5.3.

KeYmaera X KeYmaera X participated in the scripted format of the competition (proof attempts were aborted after 60s, every proof attempt was made in a fresh prover instance with all caches cleared): In the scripted format, using Z3 as its only backend, KeYmaera X solved 52 of the 61 examples with an average duration of $600ms$ for checking the scripted proofs (minimum duration $9ms$, maximum $4s$). This drop from 60 proved examples when compared to [MZS+22] is due to using Z3 as a backend instead of Wolfram Engine.

HHL Prover/HHLPy. The HHL Prover successfully proved 49 of the 61 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.

The level of automation of HHLPy is between that of hints mode and scripted mode, requiring the users to annotate both the invariants and the rules of differential equations. Proof attempts were aborted after 300s. HHLPy verified 50 out of 61 examples in this category. For the 11 unverified examples, eight of them could not be translated into Hoare triples of HCSP programs, due to the semantics difference between $\mathsf{d}\mathcal{L}$ and Hoare triples of HCSP programs; one of them is non-polynomial; and we are still not clear about how to prove the last two. All of the verification conditions generated of the verified ones could be proved by Z3.

IsaVODEs. The participation of the IsaVODEs' team remains as in last year. We only did the scripted format of the competition. We solved 59 of the 61 Design Shapes problems with it. We fully proved 55 of those 59, while the remaining 4 required involvement of the WolframEngine. Computer algebra systems usually help interactive provers discharge proof obligations about real arithmetic. In our case, the WolframEngine reduced useful (in)equalities to the boolean value `true`. Thus, we could assert (but not prove) those (in)equalities within Isabelle. Yet, we used them as facts in our proofs of the aforementioned four Design Shapes. Isabelle-asserted lemmas appear in our reproducibility package's proof document with `sorry` commands below them.

Due to the competition time constraints, we could not find an argument for the truth of one of the remaining two problems: "LICS: Example 4b progress of time-triggered car". However, we were able to state the verification formula in IsaVODEs and formalise forward diamond operator rules to tackle it. We marked unsolved problems with `oops` in our proof document. The last unsolved problem, "Dynamics: Fractional Darboux equality", requires an unproved variant of Isabelle's Picard-Lindelöef theorem or a generalisation of our differential Ghost rule.

We measured Isabelle's speed to certify our proofs in two ways. Firstly, we measured the time it took Isabelle to certify our proofs for each problem. The average per Design Shape solved problem was 2.39 seconds in our 8 core Apple M1 machine. Alternatively, we measured the time it took the same machine to certify all problems. The computer averaged 34.21 seconds for the first 58 problems, making each of them solved in less than 0.59 seconds. We attribute the disparity between both methods to the inaccuracies generated by manual measurements in the first method and the extra effort the machine needs to display each problem to the monitor.

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].

KeYmaera X. Due to the benchmark setup on the repeatability system with only Z3 available as a backend, KeYmaera X participated in the scripted format, but not the hints and automated format (proof attempts were aborted after 60s, every proof attempt was made in a fresh prover instance with all caches cleared): In the scripted format, using Z3 as its only backend, KeYmaera X solved 96 of the 141 examples, with an average duration of about $1s$ per example (minimum duration $13ms$, maximum $25s$).

HHLPy. The level of automation of HHLPy is between that of hints mode and scripted mode, requiring the users to annotate both the invariants and the rules of differential equations. Proof attempts were aborted after 300s. HHLPy verified 103 out of 141 examples in this category. For most of the unverified ones, we have not found appropriate invariants. Most of the generated verification conditions could be proved both by Z3 and Wolfram Engine, while some could only be proved by Z3 or Wolfram Engine. We found that Z3 has advantages of handling complex boolean expressions, while Wolfram Engine is better at handling decimals and quantifiers.

4.3 Hybrid Games

The hybrid games benchmark tests basic games reasoning over 3 examples with adversarial dynamics. Future editions of the competition may utilize extended games case studies, such as [CMP23].

KeYmaera X. KeYmaera X solves all 3 examples in scripted mode (for hints and automation see last year's results [MZS+22]), with an average duration of about $0.4s$ per example.

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[3] 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].

KeYmaera X. KeYmaera X participated in the scripted, hints, and automated format (proof attempts were aborted after 300s, every proof attempt was made in a fresh prover instance with all caches cleared), and attempted 8 examples (3 ETCS train control, 3 flight collision avoidance, 2 robot collision avoidance). In the scripted format, using Z3 as its only backend, KeYmaera X solved 5 examples with a total time of $14s$ (average duration $2.9s$, minimum $700ms$, maximum $8s$). Again, the drop in solved examples is due to switching from Wolfram Engine to Z3.

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

IsaVODEs. Our participation remains as in last year's competition. We did the scripted format of the competition and only tackled the European train control system (ETCS) case study benchmark. ETCS is further divided in three: essentials (safety), controllability and reactivity. We fully verified the essentials part of the problem with IsaVODEs. With the help of the WolframEngine, we also solved the controllability problem. We were able to formalise ETCS reactivity in Isabelle, but we did not find an argument for the truth of the problem within the timespan of the competition.

4.5 Hybrid Systems from Simulink/Stateflow Models

Category overview. This category contains hybrid systems modeled using Simulink/Stateflow. These models are first translated into the modeling language used for verification, and then its properties are verified using the appropriate tools. For now, only 5 benchmark problems are included, which illustrates the basic semantics of Simulink and Stateflow[4]. They include Stateflow charts with one or two states, ODEs within each state, delay blocks in Simulink diagrams, and a cruise control system modeled by Simulink diagrams.

HHLPy. HHLPy successfully verified all 5 examples. All of the generated verification conditions were proved by Z3. The Simulink/Stateflow diagrams were translated into HCSP programs automatically using the methods in [XZW+23, GZX+22]. The resulting HCSP programs were annotated manually with pre- and post-conditions specifying the desired properties, invariants and necessary proof rules for differential equations. The annotated Hoare triples were then verified automatically by HHLPy. For example, the Simulink diagram of the cruise control system consisted of subsystems for the PI controller and the vehicle. The translation process combined the controller and vehicle dynamics into a single differential equation. The initial and desired values for speed and control signal were annotated as pre- and post-conditions. The invariants were derived following the standard theory for analyzing linear dynamical systems. HHLPy generated 7 verification conditions for this example and all of them were proved by Z3.

5 Modeling and Proof Comparison

In addition to the examples from [MZS+22] (Harmonic Oscillator, Train Collision Avoidance), in this edition we introduce an example of a pendulum using trigonometric functions [GTMP22].

5.1 Harmonic Oscillator

The second-order ODE $x''(t) = a \cdot x(t) + b \cdot x'(t)$ represents a harmonic oscillator. It can be encoded as a linear system of ODEs and embedded in a hybrid program [Hue20a, Example, 6.1]:

$$x = 0 \land b^2 + 4a > 0 \land a < 0 \land b \leq 0 \rightarrow [(x := *; ?x > 0; y := 0; \{x' = y, y' = ax + by\})^*]x \geq 0 \ .$$

The term $b^2 + 4a$ is the oscillator's damping factor [Att03]. The hybrid program then states that releasing the oscillator starting from rest ($y = 0$) and arbitrarily extended ($x > 0$) will keep the oscillator extended in an overdamped system ($b^2 + 4a4 > 0$, $a < 0$ and $b \leq 0$).

[4] https://gitee.com/bhzhan/mars/tree/master/hhlpy/examples/simulink

KeYmaera X The KeYmaera X model and proof remained unchanged from [MZS+22].

```
Theorem "Benchmarks/Basic/Affine: Overdamped Door Closing Mechanism"

Definitions
  Real a, b;
End.

ProgramVariables
  Real x, y;
End.

Problem
  x=0 & b^2+a*4 > 0 & a<0 & b<=0
  ->
  [{x:=*; ?x>0; y:=0;
    {x'=y, y'=a*x+b*y}
   }*@invariant(x>=0)
  ]x>=0
End.

Tactic "Scripted proof"
implyR(1);
loop("x>=0", 'R=="[{x:=*;?x>0;y:=0;{x'=y,y'=a*x+b*y}}*]x>=0"); <(
  "Init": QE using "x=0 :: x>=0 :: nil",
  "Post": id,
  "Step":
    unfold;
    cut("\exists w w=(-b+(b^2+4*a)^(1/2))/2"); <(
      "Use":
        existsL('L=="\exists w w=(-b+(b^2+4*a)^(1/2))/2");
        dC("-w*x<=y&y<=0", 'R=="[{x'=y,y'=a*x+b*y}]x>=0"); <(
          "Use":
            dW('R=="[{x'=y,y'=a*x+b*y&true&-w*x<=y&y<=0}]x>=0");
            QE,
          "Show":
            ODEinv('R=="[{x'=y,y'=a*x+b*y}](-w*x<=y&y<=0)")
        ),
      "Show":
        QE using "b^2+a*4>0 :: \exists w w=(-b+(b^2+4*a)^(1/2))/2 :: nil"
    )
)
End.
```

IsaVODEs The “Harmonic Oscillator” benchmark remains as in last year’s competition. We restate our approach to tackling it here. The benchmark has been verified before with the predecessor verification components [Hue20a]. We formalised the $\mathsf{d}\mathcal{L}$ variant for this competition and proved it with IsaVODEs. We showed the equivalence with our previous formalisation (in terms of matrices) and used our results about linear systems of ODEs to verify the specification. Thus, we represented the system of ODEs via $z' = A \cdot z$ where $z = (x, y)^\top$ and

$$A = \begin{pmatrix} 0 & 1 \\ a & b \end{pmatrix}.$$

(in Isabelle) **abbreviation** $A \equiv mtx$
  $([0,\ 1]\ \#$
   $[a,\ b]\ \#\ [])$

In the code above, mtx transforms Isabelle lists to square matrices. Then we showed that A is diagonalizable: $A = P \cdot D \cdot P^{-1}$ for a change of basis matrix P and diagonal matrix D.

lemma *mtx-hosc-diagonalizable*:
  assumes $b^2 + a * 4 > 0$ **and** $a \neq 0$
  shows $A = P\ (-\iota_2/a)\ (-\iota_1/a) * (\text{diag}\ i.\ \mathit{if}\ i = 1\ \mathit{then}\ \iota_1\ \mathit{else}\ \iota_2) * (P\ (-\iota_2/a)\ (-\iota_1/a))^{-1}$
<proof>

We omit the definition of P and abbreviate the 7-line proof for the above lemma with the indicator `<proof>` (see proof document for full proofs). Also, ι_1 and ι_2 are A's eigenvalues in the diagonal matrix D. Therefore, the solution Φ^* to the linear system of ODEs is $\Phi^*(t) = P \cdot \exp(-tD) \cdot P^{-1}$, where exp is the matrix exponential.

```
definition discr ≡ sqrt (b^2 + 4 ∗ a)

abbreviation Φ t ≡ mtx (
  [ι2∗exp(t∗ι1) − ι1∗exp(t∗ι2),     exp(t∗ι2)−exp(t∗ι1)]#
  [a∗exp(t∗ι2) − a∗exp(t∗ι1), ι2∗exp(t∗ι2)−ι1∗exp(t∗ι1)]#[])

lemma mtx-hosc-solution-eq:
  assumes b^2 + a ∗ 4 > 0 and a ≠ 0
  shows P (−ι2/a) (−ι1/a) ∗ (diag i. exp (t ∗ (if i=1 then ι1 else ι2))) ∗ (P (−ι2/a) (−ι1/a))^−1
  = (1/discr) ∗R (Φ t)
<proof>
```

Here, $\Phi^*(t) = (1/discr)\Phi(t)$. The fact that this function is the unique solution to the system of ODES corresponds to the Isabelle statement below.

```
lemma local-flow-mtx-hosc:
  assumes b^2 + a ∗ 4 > 0 and a ≠ 0
  shows local-flow ((∗V) A) UNIV UNIV (λt. (∗V) ((1/discr) ∗R Φ t))
<proof>
```

We used this solution to prove the corresponding solution for the d$\mathcal{L}$-style system of ODEs. Then, we apply usual wlp-reasoning to prove the benchmark's specification.

```
lemma local-flow-hosc: a ≠ 0 =⇒ b^2 + 4 ∗ a > 0
  =⇒ local-flow-on [x ⇝ y, y ⇝ a ∗ x + b ∗ y] (x +L y) UNIV UNIV
  (λt. [x ⇝ x-sol t x y, y ⇝ y-sol t x y])
<proof>

lemma a < 0 =⇒ b ≤ 0 =⇒ b^2 + 4 ∗ a > 0
  =⇒ {x=0}
  LOOP (
    (x ::= ?);(y ::= 0); ¿x>0?;
    {x‘ = y, y‘ = a ∗ x + b ∗ y}
  ) INV (x≥0)
  {x≥0}
  apply (rule hoare-loopI)        (* apply Hoare-rule for loops with invariants *)
    prefer 3 apply expr-simp    (* discharge invariant implies postcondition *)
   prefer 2 apply expr-simp     (* discharge precondition implies invariant *)
  apply (clarsimp simp add:
    wp fbox-g-dL-easiest[OF local-flow-hosc])  (* apply wlp-rules including ODEs solution*)
  apply expr-simp
  apply (clarsimp simp: iota1-def iota2-def discr-def)
  using overdamped-door-arith[of b a] by force  (* discharge remaining real-arithmetic obligations *)
```

5.2 Train Collision Avoidance

KeYmaera X The example remained unchanged from [MZS+22].

HHLPy The ETCS example could also be described in a Hoare triple of HCSP programs, which is listed below. The triple was annotated with invariants and proof rules for differential equations. And we used `function` to define real functions (*stopDist*(v), *accCompensation*(v) and *SB*(v)) and boolean functions (*safeInv*(m, z, v)). The specification expressed that starting from the initial condition (pre-condition), after repeatedly running the controller (`if`, `else if`, `else` commands) and then driving according to the differential equation any number of times, the system was safe ($z \leq m$ in post-condition).

To prove safety, the loop was annotated with invariants ($v \geq 0 \wedge \mathit{safeInv}(m, z, v)$), and the differential equation was annotated with `solution`, representing using the solution of the differential equation for proof. HHLPy generated 19 verification conditions for proof and Z3/Wolfram Engine could verify all of the verification conditions.

```
function stopDist(v) = v^2/(2*b);
function accCompensation(v) = ((A/b) + 1)*((A/2)*ep^2 + ep*v);
function SB(v) = stopDist(v) + accCompensation(v);

function safeInv(m, z, v) = m-z >= stopDist(v);


pre [v >= 0] [m-z >= stopDist(v)] [b>0] [A>=0] [ep>=0];
{
    # The controller
    if (m - z < SB(v)){
        a := -b;
    }
    else if (m - z > SB(v)) {
        a := A;
    }
    else {
        a := -b; ++ a := A;
    }

    # The plant
    t := 0;
    {z_dot=v, v_dot=a, t_dot=1 & v > 0 && t < ep}     # Drive forwards for ep time (v > 0 && t == ep)
        ↪ , or drive forwards until v == 0, for at most ep time (v == 0 && t <= ep)
        solution;                   # Solve the differential equation for proof
}* invariant [v >= 0] [safeInv(m, z, v)];
post [z <= m];
```

IsaVODEs Our formalisation of the European Train Control System (ETCS) remains mostly as in last year. The difference resides in that our proof-automation tactics have reduced the size of our proofs. We start by describing the problem's set of variables and constants.

dataspace *ETCS* =
  constants
    ε :: *real* — control cycle duration upper bound
    b :: *real* — braking force
    A :: *real* — maximum acceleration
    m :: *real* — end of movement authority (train must not drive past m)
  variables
    t :: *real* — Actual control cycle duration t <= ep)
    z :: *real* — Train position)
    v :: *real* — Current velocity)

a :: *real* — Actual acceleration -b <= a <= A)

Next, we formalise the definitions with the command **abbreviation** as shown below.

abbreviation *stopDist w* ≡ $w^2/(2*b)$

Once all definitions are in place, we provide the solution to the problem's system of ODEs, and show in one-line that it is indeed the unique solution to it.

lemma *local-flow-LICS1*: *local-flow-on* $[t \rightsquigarrow 1,\ v \rightsquigarrow \$a,\ z \rightsquigarrow \$v]\ (z +_L v +_L t)$ *UNIV UNIV*
$(\lambda\tau.\ [t \rightsquigarrow \tau + t,\ z \rightsquigarrow \$a * \tau^2\ /\ 2 + \$v * \tau + \$z,\ v \rightsquigarrow \$a * \tau + \$v])$
by *local-flow-on-auto*

Finally, we use this result to prove the safety specification applying Hoare and wlp-laws.

lemma *initial* ≤ |*LOOP ctrl*;*drive INV* @*loopInv*] $(z \leq m)$
apply (*subst change-loopI*[**where** $I{=}(@loopInv \wedge b > 0 \wedge A \geq 0 \wedge \varepsilon \geq 0)^e$])
by (*rule hoare-loopI*)
using *ETCS-arith1*[*of b A* get_v - - ε *m* get_z -]
by (*auto simp*: *unrest-ssubst var-alpha-combine wp usubst usubst-eval fbox-g-dL-easiest*[*OF local-flow-LICS1*] *field-simps taut-def*)
(*smt* (*verit*, *best*) *mult-left-le-imp-le zero-le-square*)

5.3 Pendulum with Discrete Push

This year's competition introduces the hybrid program of a pendulum with discrete push [GTMP22] below. The pendulum has a rod length L, is slowed down by friction k and its swing is governed by gravity g. The position of the tip of the pendulum is characterized with angle θ relative to the vertical (downwards) orientation. The pendulum may receive a discrete push p that acts on the angular velocity w if the extra force from the push does not make it swing above horizontal.

$$g > 0 \wedge L > 0 \wedge k > 0 \wedge \overbrace{\theta = 0 \wedge w = 0}^{\text{pendulum starts at rest}} \rightarrow$$
$$\left[\left(p := *; \text{if } \left(\frac{1}{2}(w-p)^2 < \frac{g}{L}\cos(\theta)\right)\ w := w - p \text{ fi};\right.\right.$$
$$\left.\left.\{\theta' = w, w' = -\frac{g}{L}\sin(\theta) - kw\}\right)^*\right]\left(-\frac{\pi}{2} < \theta < \frac{\pi}{2}\right)$$

KeYmaera X The pendulum with discrete push is expressed in KeYmaera X ASCII syntax below.

```
Theorem "Pendulum with Discrete Push"

Definitions
  import kyx.math.{cos,sin,pi};
  Real g; /* Gravity */
  Real L; /* Length of rod */
  Real k; /* Coefficient of friction against angular velocity */
End.

```

```
ProgramVariables
  Real w;      /* Angular velocity */
  Real theta;  /* Displacement angle */
  Real push;   /* Extra push */
End.

Problem
  g > 0 & L > 0 & k > 0 &
  theta = 0 & w = 0 /* Pendulum starts at rest */
  ->
  [{
    /* Push if extra force will not make pendulum swing above horizontal */
    push :=*;
    if (1/2*(w-push)^2 < g/L *cos(theta)) { w := w-push; }
    /* Pendulum dynamics: angle and angular velocity */
    { theta' = w, w' = -g/L * sin(theta) - k*w }
  }*]
  (-pi/2 < theta & theta < pi/2) /* Pendulum never crosses horizontal */
End.
```

The proof in KeYmaera X uses Mathematica as its backend. The main insights are a loop invariant $\frac{g}{L}(1 - \cos(\theta)) + \frac{1}{2}w^2 < \frac{g}{L} \wedge \frac{\pi}{2} < \theta < \frac{\pi}{2}$ and a monotonicity step using the first conjunct of the loop invariant $\frac{g}{L}(1 - \cos(\theta)) + \frac{1}{2}w^2 < \frac{g}{L}$ as an intermediate formula (in order to focus differential equation automation on proving that the angular velocity does not exceed the threshold of swinging above horizontal). The proof tactic is listed below.

```
useSolver("Mathematica");
unfold;
loop("g/L*(1-cos(theta))+1/2*w^2 < g/L&-pi/2 < theta&theta < pi/2", 1"); <(
  "Init": QE,
  "Post": propClose,
  "Step":
    composeb(1);
    MR("g/L*(1-cos(theta))+1/2*w^2 < g/L", 1"); <(
      "Use Q->P":
        unfold;
        QE,
      "Show [a]Q":
        ODE(1)
    )
)
```

IsaVODEs For time constraints, we do not include this problem in our repeatability package. Nevertheless, we explain its formalisation and verification here, and direct readers to our repository[5]. As before, we start by defining the problem's variables and constants.

dataspace *sys* =
  constants *g* :: *real* *L* :: *real* *k* :: *real*
  assumes *L-gr-0*: *L* > *0* **and** *K-gr-0*: *k* > *0* **and** *g-gr-0*: *g* > *0*
  variables *ω* :: *real* *ϑ* :: *real* *push* :: *real*

In turn, these allow us to define the program structure and its invariant from [GTMP22].

abbreviation *ctrl* ≡ *IF* $(1/2 * (\omega - push)^2 < g / L * cos(\vartheta))$ *THEN* $\omega ::= \omega - push$ *ELSE skip*

abbreviation *ode* ≡ $\{\vartheta` = \omega, \omega` = -g/L * sin(\vartheta) - k * \omega \mid -pi < \vartheta \wedge \vartheta < pi\}$

abbreviation *program* ≡ *LOOP* (*ctrl* ; *ode*)

[5] https://github.com/isabelle-utp/Hybrid-Verification/blob/main/Hybrid_Programs/Verification_Examples/Pendulum_with_force.thy

$INV\ (-pi < \vartheta \wedge \vartheta < pi \wedge g\ /\ L * (1\ -\ cos(\vartheta)) + (1/2 * \omega^2) < g\ /\ L)$

abbreviation $invariant \equiv (-pi < \vartheta \wedge \vartheta < pi \wedge (g\ /\ L * (1\ -\ cos(\vartheta)) + (1/2 * \omega^2) < g\ /\ L))^e$

The proof is then standard. We show that both the dynamics of the hybrid program and its discrete counterpart preserve the invariant. We also show that the invariant implies the postcondition. Finally, we use these facts in the Hoare-style proof.

lemma *ode-correct*: **{**@(*invariant*)**}** *ode* **{**@(*invariant*)**}**
by (*dInduct-mega*, *meson K-gr-0 dual-order.order-iff-strict mult-nonneg-nonneg zero-le-square*)

lemma *ctrl-correct*: **{**@(*invariant*)**}** *ctrl* **{**@(*invariant*)**}**
apply *wlp-simp*
apply (*simp-all add*: *usubst-eval*)
apply (*expr-auto*)
using *L-gr-0* **apply** (*simp add*: *field-simps*)
by (*metis* (*no-types*, *opaque-lifting*) *distrib-left mult-less-cancel-left-pos*)

lemma *inv-impl-postcondition*: ‘@(*invariant*) $\longrightarrow -\ pi\ /\ 2 < \vartheta \wedge \vartheta < pi\ /\ 2$‘
by (*metis* (*no-types*, *lifting*) *L-gr-0 SEXP-def g-gr-0 tautI trig-prop*(*1*) *trig-prop*(*2*))

lemma *program-correct*: **{**$\vartheta = 0 \wedge \omega = 0$**}** *program* **{**$-\ pi\ /\ 2 < \vartheta \wedge \vartheta < pi\ /\ 2$**}**
apply *intro-loops*
apply (*rule hoare-kcomp-inv*)
using *ctrl-correct* **apply** *blast*
using *ode-correct* **apply** *blast*
apply *expr-simp*
using *L-gr-0 divide-pos-pos g-gr-0* **apply** *blast*
using *inv-impl-postcondition* **apply** *blast*
done

For the proof that the invariant implies the postcondition. Isabelle requires knowledge of some arithmetical facts about trigonometric functions. This is similar to reasoning about arithmetical facts of real numbers. In particular, we showed the following statement.

lemma *trig-prop*:
fixes *g L k* :: *real*
assumes $L > 0\ k > 0\ g > 0\ g\ /\ L * (1\ -\ cos(\vartheta)) + (1/2 * \omega^2) < g\ /\ L\ -pi < \vartheta\ \vartheta < pi$
shows $-\ pi\ /\ 2 < \vartheta\ \vartheta < pi\ /\ 2$
<proof>

6 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 with game examples [CMP23] 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.

Acknowledgments. This material is based upon work supported by the AFOSR under grant number FA9550-16-1-0288. 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. We thank the entire Logical Systems Lab for their many contributions and suggestions to KeYmaera X and its associated tools. Bohua Zhan, Huanhuan Sheng and Shuling Wang are funded partly by NSFC under grant No. 61625206, 61732001, 62032024 and 61972385.

References

[Att03] Mary Attenborough. 14 - differential equations and difference equations. In Mary Attenborough, editor, *Mathematics for Electrical Engineering and Computing*, pages 346–381. Newnes, Oxford, 2003.

[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.

[CMP23] Rachel Cleaveland, Stefan Mitsch, and André Platzer. Formally verified next-generation airborne collision avoidance games in ACAS X. *ACM Trans. Embed. Comput. Syst.*, 22(1):10:1–10:30, 2023.

[FBC+20] S. Foster, J. Baxter, A. Cavalcanti, J. Woodcock, and F. Zeyda. Unifying semantic foundations for automated verification tools in Isabelle/UTP. *Science of Computer Programming*, 197, October 2020.

[FHGS21] Simon Foster, Julián Huerta y Munive, Mario Gleirscher, and Georg Struth. Hybrid systems verification with isabelle/hol: Simpler syntax, better models, faster proofs. In *Proc. 24th Intl. Symp. on Formal Methods (FM 2021)*, volume LNCS 13047, pages 367–386, 2021.

[FHS20] Simon Foster, Jonathan Julián Huerta y Munive, and Georg Struth. Differential hoare logics and refinement calculi for hybrid systems with isabelle/hol. In *RAMiCS 2020[postponed]*, pages 169–186, 2020.

[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.

[GTMP22] James Gallicchio, Yong Kiam Tan, Stefan Mitsch, and André Platzer. Implicit definitions with differential equations for keymaera X - (system description). In *Automated Reason-*

ing - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings, pages 723–733, 2022.

[GZX+22] Panhua Guo, Bohua Zhan, Xiong Xu, Shuling Wang, and Wenhui Sun. Translating a large subset of stateflow to hybrid CSP with code optimization. *J. Syst. Archit.*, 130:102665, 2022.

[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.

[HH98] C. A. R. Hoare and J. He. *Unifying Theories of Programming*. Prentice-Hall, 1998.

[HS22] Jonathan Julián Huerta y Munive and Georg Struth. Predicate transformer semantics for hybrid systems. *JAR*, 66(1):93–139, 2022.

[Hue19] Jonathan Julián Huerta y Munive. Verification components for hybrid systems. *Archive of Formal Proofs*, 2019.

[Hue20a] Jonathan Julián Huerta y Munive. Affine systems of ODEs in Isabelle/HOL for hybrid-program verification. In *SEFM 2020*, volume 12310 of *LNCS*, pages 77–92. Springer, 2020.

[Hue20b] Jonathan Julián Huerta y Munive. Matrices for odes. *Archive of Formal Proofs*, 2020.

[LLQ+10] Jiang Liu, Jidong Lv, Zhao Quan, Naijun Zhan, Hengjun Zhao, Chaochen Zhou, and Liang Zou. A calculus for hybrid CSP. In Kazunori Ueda, editor, *Programming Languages and Systems - 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010. Proceedings*, volume 6461 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2010.

[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.

[MJZ+21] Stefan Mitsch, Xiangyu Jin, Bohua Zhan, Shuling Wang, and Naijun Zhan. ARCH-COMP21 category report: Hybrid systems theorem proving. In *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21), Brussels, Belgium, July 9, 2021*, pages 120–132, 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.

[MP16] Stefan Mitsch and André Platzer. The keymaera X proof IDE - concepts on usability in hybrid systems theorem proving. In *Proceedings of the Third Workshop on Formal Integrated Development Environment, F-IDE@FM 2016, Limassol, Cyprus, November 8, 2016*, pages 67–81, 2016.

[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+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.

[MZS+22] Stefan Mitsch, Bohua Zhan, Huanhuan Sheng, Alexander Bentkamp, Xiangyu Jin, Shuling Wang, Simon Foster, Christian Pardillo Laursen, and Jonathan Juli\'an Huerta Y Munive.

Arch-comp22 category report: Hybrid systems theorem proving. In Goran Frehse, Matthias Althoff, Erwin Schoitsch, and Jeremie Guiochet, editors, *Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*, volume 90 of *EPiC Series in Computing*, pages 185–203. EasyChair, 2022.

[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.

[Pla10] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010.

[Pla11] André Platzer. The structure of differential invariants and differential cut elimination. *Log. Methods Comput. Sci.*, 8(4), 2011.

[Pla12] André Platzer. Logics of dynamical systems. In *LICS*, pages 13–24. IEEE, 2012.

[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. *CoRR*, abs/1802.01226, 2018.

[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.

[SCMP21] Matias Scharager, Katherine Cordwell, Stefan Mitsch, and André Platzer. Verified quadratic virtual substitution for real arithmetic. In *Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings*, pages 200–217, 2021.

[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.

[SMT+21] Andrew Sogokon, Stefan Mitsch, Yong Kiam Tan, Katherine Cordwell, and André Platzer. Pegasus: sound continuous invariant generation. *Formal Methods Syst. Des.*, 58(1-2):5–41, 2021.

[TMP22] Yong Kiam Tan, Stefan Mitsch, and André Platzer. Verifying switched system stability with logic. In *HSCC '22: 25th ACM International Conference on Hybrid Systems: Computation and Control, Milan, Italy, May 4 - 6, 2022*, pages 2:1–2:11, 2022.

[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.

[XZW+23] Xiong Xu, Bohua Zhan, Shuling Wang, Jean-Pierre Talpin, and Naijun Zhan. A denotational semantics of simulink with higher-order UTP. *J. Log. Algebraic Methods Program.*, 130:100809, 2023.

[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.

[ZWR95] 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: Verification and Control, Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, volume 1066 of *Lecture Notes in Computer Science*, pages 511–530. Springer, 1995.

[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.