EPiC Series in Computing

Volume 108, 2025, Pages 152–168

Proceedings of 12th Int. Workshop on Applied Verification for Continuous and Hybrid Systems

ARCH-COMP25 Category Report: Hybrid Systems Theorem Proving

Stefan Mitsch[1], Ismail Patel[1], Hari Hara Sudhan Kannan[1], Xiangyu Jin[2], Bohua Zhan[3], and Shuling Wang[4]

[1] School of Computing, DePaul University, Chicago, IL, USA
smitsch@depaul.edu,ipatel31@depaul.edu,hkannan1@depaul.edu
[2] Key Laboratory of System Software and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
jinxy@ios.ac.cn
[3] Huawei Technologies Co., Ltd, Beijing, China
zhanbohua@huawei.com
[4] National Key Laboratory of Space Integrated Information System, Institute of Software, Chinese Academy of Sciences, Beijing, China
wangsl@ios.ac.cn

Abstract

This paper reports on the Hybrid Systems Theorem Proving (*HSTP*) category in the ARCH-COMP Friendly Competition 2025. 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-COMP25 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, MSZ+23]. 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+21].
- 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 (dℒ) [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 (dℒ) [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 dℒ can be found in [QML+16], details on the ASCII syntax are in [MMJ+20]. Libraries of pre-defined functions (e.g., `import kyx.math.abs`) [GTMP22] help expressing models more conveniently. 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. KeYmaera X provides a definitions mechanism for functions [GTMP22]. The newest addition to the KeYmaera X modeling tool suite is a VSCode extension for editing input files and checking proofs in KeYmaera X[1]. The extension supports VSCode language server features including syntax highlighting, hover information, syntax error diagnostics, and code action suggestions (quick fixes). The tool supports running proof checking for individual tactics returning output results and pass/fail symbols. Fig. 1 shows a screenshot of the Hybrid Games input file with the file content outline on the left. An option to check the proof for all the tactics is displayed above each entry and each individual tactic in the editor. The check proof feature communicates with the KeYmaera X backend via command line integration and displays the result in editor output along with visual symbols next to the selected tactic.

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 dL. 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 rea-

[1] https://github.com/ProVE-Lab/vscode-kyx

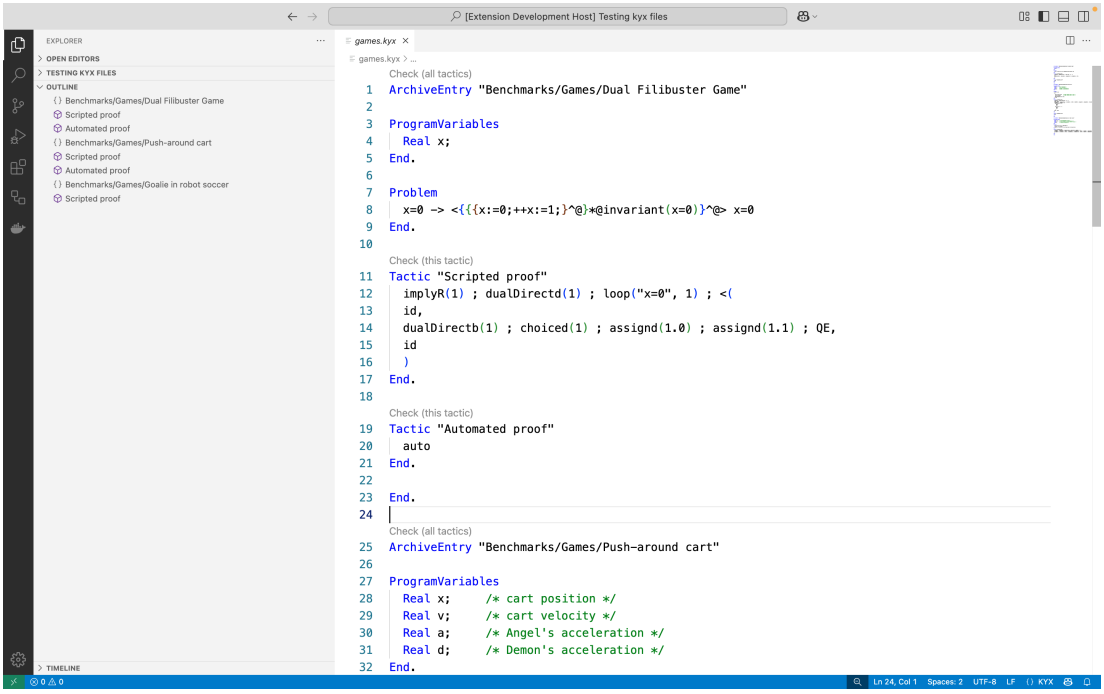

Figure 1: Screenshot of the VSCode extension

soning 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. The input and output blocks are synchronized during parallel composition of HCSP processes.

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 dℒ [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.

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 category for hybrid systems from Simulink/Stateflow models.

Experimental Setup. 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 a machine with Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz.

In this edition, KeYmaera X participated in all three modes in the design shapes, games, nonlinear continuous models, and case study benchmarks categories. The performance results reported here are obtained on with Wolfram Engine as a backend on the Chameleon testbed.

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 dℒ tutorial examples [Pla12] ranging from basic time-triggered motion control to model-predictive control.

STTT Tutorial 12 dℒ 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.

KeYmaera X KeYmaera X participated in scripted, hints, and automated mode (proof attempts were aborted after 60s, every proof attempt was made in a fresh prover instance with all caches cleared):

Script 60 of 61 examples solved (average 670ms, maximum 5.3s).
Hints 52 of 61 examples solved (average 540ms, maximum 1.8s).
Auto 50 of 61 examples solved (average 440ms, maximum 2.1s).

As in previous competitions ([MZS+22]) the remaining unsolved example is a progress proof.

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 dℒ 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.

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+21].

KeYmaera X. KeYmaera X participated in the scripted, hints, and automated format (proof attempts were aborted after 60s, every proof attempt was made in a fresh prover instance with all caches cleared):

Script 107 of 141 examples solved (average 1.2s, maximum 36.5s).
Hints 96 of 141 examples solved (average 1.2s, maximum 19.6s).
Auto 59 of 141 examples solved (average 1.7s, maximum 10.5s).

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 participated in the scripted, hints, and automated format (proof attempts were aborted after 60s, every proof attempt was made in a fresh prover instance with all caches cleared):

Script 3 of 3 examples solved (average 530ms, maximum 1.2s).
Hints 2 of 3 examples solved (average 300ms, maximum 500ms).
Auto 2 of 3 examples solved (average 260ms, maximum 460ms).

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].

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).

Script 8 of 8 examples solved (average 7.5s, maximum 30.6s).
Hints 6 of 8 examples solved (average 1.8s, maximum 2.9s).
Auto 5 of 8 examples solved (average 1.8s, maximum 3.4s).

The difference in average/maximum duration between hints and scripted format is due to the two additional solved examples, which are more complex arithmetically and result in longer checking times in the external arithmetic solver.

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[3]. 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.

[2] https://github.com/LS-Lab/KeYmaeraX-projects/blob/master/benchmarks/advanced.kyx
[3] https://gitee.com/bhzhan/mars/tree/master/hhlpy/examples/simulink

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

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 [Hue20, 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 $b \leq 0$).

KeYmaera X The KeYmaera X model and proof remained unchanged from [MZS+22, MSZ+23].

```
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.
```

HHLPy The "Harmonic Oscillator" benchmark now has been verified in HHLPy. In the modeling of this benchmark, we view a and b as constants and give their conditions and for the sake of brevity in writing, we introduce a new constant w to represent $((b^2 + 4 * a)^{1/2} - b)/2$. In the specification, the process starts with states satisfying the pre-condition $x == 0$, and then after the assignments, the ODE is executed. Due to the different semantics of ODE in HCSP, we use the time variable t with a non-deterministic non-negative initial value to determine the length of evolution. To prove the post-condition, we set the loop invariant to be $x \geq 0$ and the differential invariant to be $y + w * x \geq 0 \wedge y \leq 0$ using the barrier certificate method. Both of them are proved using the annotated proof rules.

```
# Harmonic Oscillator
constants                                          #const condition
    a < 0;
    b <= 0;
    b^2 + 4 * a > 0;
    w == ((b^2 + 4 * a)^(1/2) - b)/2;
end

pre[x==0];                                         # pre-conditon
{
  x := * (x > 0); y := 0;
  t := * (t >= 0);
  {x_dot = y, y_dot = a * x + b * y, t_dot = -1 & t>0}
        invariant [y+w*x>=0 && y<=0]{bc}{{maintain: wolfram}};

}*
        invariant [x >= 0]{{maintain exec: wolfram}};
post[x >= 0];                                      # post-condition
```

5.2 Pendulum with Discrete Push

We briefly recall a hybrid program of a pendulum with discrete push [GTMP22], introduced in HSTP 2023 [MSZ+23]. 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.

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

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 Wolfram Engine as its backend. The main insights are a loop invariant $\frac{g}{L}(1-\cos(\theta)) + \frac{1}{2}w^2 < \frac{g}{L} \land \frac{\pi}{2} < \theta < \frac{\pi}{2}$ 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)
    )
)
```

KeYmaera X does not delegate trigonometric functions and differentially-definable constants, such as π, to external arithmetic solvers; instead, it uses differential equation reasoning to prove properties about them [GTMP22].

HHLPy The "Pendulum with Discrete Push" benchmark is now verified in HHLPy. We add $th < pi \land th > -pi$ in constraints of the ODE, and prove that $(1 - cos(th)) * g/L + 1/2 * w^2 < g/L$ is both the differential and loop invariant. From that we can obtain the post-condition $th < pi/2 \land th > -pi/2$. Since the verification conditions generated contains trigonometric functions and π, we mainly adapt Wolfram Engine to this model.

```
# Pendulum with Discrete Push
constants                                   # const condition
    g > 0;
    L > 0;
    k > 0;
end

pre[th == 0][w == 0];                       #pre−condition
{
  p:=∗(true);
  if ((1/2)∗(w−p)^2 < cos(th)∗g/L)
    {w:=w−p;}
  else
    {skip;}
  t := ∗ (t >= 0);
  {th_dot=w, w_dot=−g/L∗sin(th)−k∗w, t_dot=−1 & t>0&&th<pi&&th>−pi}
        invariant [(1−cos(th))∗g/L+1/2∗w^2<g/L]{{init 1: wolfram, init 2: wolfram, maintain: wolfram}};
}∗
        invariant [(1−cos(th))∗g/L+1/2∗w^2<g/L]{{init: wolfram, maintain 1.skip: wolfram, maintain 2.skip:
            ↪ wolfram, maintain exec: wolfram}}[th<=pi && th>=−pi]{{init: wolfram, maintain 1.skip:
            ↪ wolfram, maintain 2.skip: wolfram, maintain exec: wolfram}};
post[th<pi/2 && th>−pi/2]{{wolfram}}; #post−condition
```

5.3 Skydiver

The skydiver example [FMBP17] illustrates differential ghost reasoning as a technique to certify arguments about conservation of energy.

KeYmaera X The skydiver model is expressed in KeYmaera X ASCII syntax below.

```
Theorem "Parachute Bounded Velocity"

Definitions
  Real g = 9;    /∗ gravity ∗/
  Real p = 1;    /∗ parachute drag coefficient ∗/
  Real a;        /∗ skydiver air drag coefficient ∗/
  Real m;        /∗ impact velocity ∗/
  Real T;        /∗ skydiver reaction time ∗/
End.

ProgramVariables
  Real x;        /∗ skydiver altitude ∗/
  Real v;        /∗ skydiver speed (< 0, so lower is faster) ∗/
  Real r;        /∗ skydiver current drag coefficient (depends on parachute open/closed) ∗/
  Real t;        /∗ time ∗/
End.

Problem
  g>0 & p>a & a>0 & T>0 & m < −(g/p)^(1/2) &
  x>=0 & v<0 & v > −(g/p)^(1/2) & r=a
−>
[ {
    {
      ?(v − g∗T > −(g/p)^(1/2) & r = a);
```

```
      ++
      r := p;
    }
    t := 0;
    {{x'=v, v'=-g+r*v^2, t'=1 & t<=T & x>=0 & v<0}@invariant(
      (v'=-g+a*v^2 -> v-g*(T-t)>-(g/p)^(1/2)),
      (v'=-g+p*v^2 -> v>=old(v)-g*t))
    }
  }* @invariant((x>=0 & v<0) & v>-(g/p)^(1/2))
](x=0 -> v>=m)
End.
```

The differential invariants are conditional on the state at the start of the differential equation: if $r = a$, so $v' = -g + a \cdot v^2$, then the differential equation preserves $v - g(T - t) > -\sqrt{g/p}$; otherwise ($r = p$), the differential equation preserves $v \geq old(v) - g \cdot t$. This proof hint uses $old(v)$ to introduce a ghost variable that holds the original value of v at the start of the differential equation. The model is proved using a differential ghost, an auxiliary differential equation that helps to express conservation of energy: the differential equation $y' = -\frac{1}{2} \cdot p \cdot (v - \sqrt{g/p}) \cdot y$ balances the change in v, so energy is constant per $y^2(v + \sqrt{g/p}) = 1$.

HHLPy In proof of the skydiver model in HHLPy, we use a constant c to represent $(g/p)^{1/2}$. To deal with the ODE, we introduce a ghost variable y with $y' = -(1/2) * r * (v - d) * y$, and then we can prove the following differential invariants: $r > 0$, $d == (g/r)^{1/2}$, $y^2 * (v + d) == 1$, $r == p \vee v - g * (T - t) > -c$ step by step. By the use of the loop invariant $x >= 0 \wedge v <= 0 \wedge r > 0 \wedge v > -c$, we can verify the post-condition.

```
constants                                          #const condition
    a > 0;
    g > 0;
    a < p;
    T >=0;
    c == (g/p)^(1/2);
    m < -(g/p)^(1/2);
end

pre[x >= 0][a == r][v > -(g/p)^(1/2)][v < 0];   #pre-condition
{ if (r==a && v-g*T>-(g/p)^(1/2))
    {skip; ++ r:=p;}
    else
    {r:=p;}
      t:=0; d:= (g/r)^(1/2);
      {x_dot = v, v_dot=r*v^2-g, t_dot=1 & x>0&&v<0&&t<T}
            invariant ghost y (y_dot= -(1/2) * r * (v-d)*y)
                  [r>0][d == (g/r)^(1/2)][y^2*(v+d)==1][r==p || v-g*(T-t)>-c]{{init_all 1(1): wolfram,
                        ↪ init_all 1(2): wolfram, init_all 2: wolfram}};
}*
        invariant [x>=0&&v<=0][r>0][v>-c]{{maintain exec: wolfram}};
post[!(x==0) || v^2<=m^2];                         #post-condition
```

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] and stability examples [TMP22] 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. We thank the entire Logical Systems Lab for their many contributions to KeYmaera X and its associated tools. KeYmaera X results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation, and we thank Alexandru Orhean for his support with the Chameleon testbed. Xiangyu Jin, Bohua Zhan and Shuling Wang are funded partly by NSFC under grant No. 62432005, 62032024, and the Major Project of ISCAS (ISCAS-ZD-202302).

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] Rose 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] Rose 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.

[FMBP17] Nathan Fulton, Stefan Mitsch, Rose 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 Reasoning - 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.

[Hue20] 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.

[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 *LNCS*, 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 Julián 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 *LNCS*, 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 Systems, part of CPS-IoT Week 2019, Montreal, QC, Canada, April 15, 2019*, volume 61 of *EPiC Series in Computing*, pages 141–161. EasyChair, 2019.

[MSZ+23] Stefan Mitsch, Huanhuan Sheng, Bohua Zhan, Shuling Wang, Simon Foster, and Jonathan Julian Huerta Y Munive. ARCH-COMP23 category report: Hybrid systems theorem proving. In Goran Frehse and Matthias Althoff, editors, *Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23)*, volume 96 of *EPiC Series in Computing*, pages 170–188. EasyChair, 2023.

[MZS+22] Stefan Mitsch, Bohua Zhan, Huanhuan Sheng, Alexander Bentkamp, Xiangyu Jin, Shuling Wang, Simon Foster, Christian Pardillo Laursen, and Jonathan Julián 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. 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.

[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+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. Special issue for selected papers from FM'19.

[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 *LNCS*, 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.