The ARCH-COMP Friendly Verification Competition for Continuous and Hybrid Systems

Alessandro Abate[1], Matthias Althoff[7], Lei Bu[12], Gidon Ernst[2], Goran Frehse[3], Luca Geretti[11], Taylor T Johnson[10], Claudio Menghi[8,9] Stefan Mitsch[6], Stefan Schupp[4], and Sadegh Soudjani[5]

[1] University of Oxford, Alessandro.Abate@cs.ox.ac.uk
[2] LMU Munich, Germany, gidon.ernst@lmu.de
[3] ENSTA Paris, Institut Polytechnique de Paris, France, goran.frehse@ensta-paris.fr
[4] TU Wien, Vienna, Austria, stefan.schupp@tuwien.ac.at
[5] Max Planck Institute for Software Systems, Germany, sadegh@mpi-sws.org
[6] DePaul University, Chicago, USA, smitsch@depaul.edu
[7] Technical University of Munich, Munich, Germany, althoff@tum.de
[8] University of Bergamo, Bergamo, Italy, claudio.menghi@unibg.it
[9] McMaster University, Hamilton, Canada, menghic@mcmaster.ca
[10] Vanderbilt University, Nashville, TN, USA taylor.johnson@vanderbilt.edu
[11] University of Verona, Verona, Italy luca.geretti@univr.it
[12] Nanjing University, Nanjing, Jiangsu, P.R. China, bulei@nju.edu.cn

Abstract. The workshop on Applied Verification for Continuous and Hybrid Systems (ARCH) is an annual venue for researchers and practitioners working on automated analysis and verification of hybrid systems. ARCH-COMP is a friendly competition held with the ARCH event. The competition was established in 2017 and aims to explore, document, and push forward the state of the art in the field. It evaluates and compares methods and tools for automated hybrid systems analysis and verification on predefined benchmark problems. It is supported by an active community around several categories of problems, including linear and nonlinear systems, simulation-based and analytic methods, and models from many application domains, such as automotive systems or neural networks. This paper describes the format of the competition and its organization. It documents the experiences and decisions from the current and past editions of the competition and presents reflections and lessons learned.

Keywords: hybrid systems · competition · tool evaluation
Website: https://cps-vo.org/group/ARCH/FriendlyCompetition

1 Introduction

The design of systems with continuous, real-valued quantities can be complicated when hard constraints on their behaviors must be satisfied. For example, designing a cruise controller to maintain the vehicle at target speed while avoiding collisions is complicated due to event-based phenomena in the plant or the controller. Hybrid

systems are a convenient modeling formalism to describe systems containing continuous variables subject to event-based dynamics changes. A hybrid system can model a cyber-physical system requiring a software controller to interact with a physical plant to ensure hard constraints. Examples of hard constraints on hybrid systems include assuring a minimum battery charge, maximum voltage levels in an electrical grid, or adherence to security features like a dead man's switch. Formal verification is an approach to assess whether a design satisfies hard constraints based on a mathematically rigorous analysis of a formal model of the system. It has been successfully applied to software and hardware systems since the 80s, and researchers have been developing verification techniques for continuous and hybrid systems since the 90s.

Verifying continuous and hybrid systems is quite challenging: even reachability is undecidable in many cases [76], and computing accurate solutions can be prohibitively expensive. The absence of sharp theoretical bounds on the computational time lends particular weight to experimental evaluations. Much of the research has therefore gone into finding abstractions and heuristics with a suitable trade-off between accuracy and computational cost such that problem instances of practical interest can be handled. However, evaluating, assessing, and comparing existing techniques raises several questions:

- Which problem instances are of practical interest, and how to find them?
- How much accuracy is really needed?
- How much computation is acceptable?

Answering these questions is challenging: The answers depend on the application domain and the particular system, the specification to be verified, and the available computation hardware. They also evolve as algorithms become more sophisticated, computation hardware gets faster, and practitioners develop more demanding problems. The goal of the ARCH workshop series and the ARCH-COMP friendly competition is to provide such problem instances to researchers and maintain a forum where the advantages and drawbacks of different approaches can be assessed. This requires defining a fair way to execute tools on problem instances and identifying instances suitable for different tools. The format and organization of ARCH-COMP are intended to facilitate progress toward these objectives and will be described in more detail in the next section.

This paper is structured as follows. Section 2 provides background information on the principles of formal verification and how they apply to continuous and hybrid systems. Section 3 describes the competition format and its organization. Section 4 presents the different tracks of the competition. Section 5 describes efforts towards repeatable experiments and results. Section 6 summarizes the achievements of ARCH-COMP, our reflections, lessons learned, and outlook.

2 Verification of Continuous, Hybrid, and Stochastic Systems

We summarize the principles of formal verification and how they apply to continuous and hybrid systems. As we will see, several difficulties arise when moving from the classic setting of finite state machines to a continuous time.

2.1 The Formal Verification Approach

A formal verification requires a model of the system and a formal description of the specification. If the formal verification procedure terminates, it either

- claims that the system satisfies the specification, possibly providing a *certificate* such as an invariant or an interpolant;
- claims that the system violates the specification, often providing a *witness* such as a counterexample;
- or returns "unknown", typically because the runtime exceeds a given upper bound.

In many verification competitions such as SV-COMP [29], participants submit their software tools, and the organizers run them on a set of benchmarks unknown a priori to the participants. The tools are evaluated by considering the correctness of their verdicts, the number of problems handled in a given time frame, and the overall runtime. To decide whether the result is correct, the problem instance either has a result known by construction, has a result considered trustworthy from a reference tool, or a referee tool checks whether the certificate or witness can be validated.

Several problems arise when replicating this procedure with continuous and hybrid systems as it is the goal in ARCH-COMP. Before discussing these problems, we define what we mean by continuous and hybrid systems and their verification.

2.2 Continuous, Hybrid, and Stochastic Systems

Continuous systems. We consider continuous systems to be systems whose state can be described by a vector $x \in \mathbb{R}^n$, i.e., with n real-valued variables. In a *discrete-time system*, the state evolves according to a function

$$x_{k+1} = f(x_k, u_k), \qquad x_0 \in \mathcal{X}_0, u_k \in \mathcal{U},$$

where k represents time, $\mathcal{X}_0 \subseteq \mathbb{R}^n$ the set of initial conditions, and $u_k \in \mathbb{R}^m$ is exogenous to the system. This is sometimes described as a nondeterministic input, but should not be confused with the notion of inputs in control systems. The input u_k is non-deterministically chosen from the compact set $\mathcal{U} \subseteq \mathbb{R}^m$. This allows one to, e.g., model bounded disturbances or to account for the difference between the dynamics function $f(\cdot)$ and the actual system. The specification is considered to be satisfied if it is valid for all admissible sequences of u_k. The input u effectively turns the equation into an inclusion.

We consider *continuous-time systems* described by ordinary differential equations (ODEs) of the form

$$\dot{x} = f(x, u), \qquad x(0) \in \mathcal{X}_0, u(t) \in \mathcal{U},$$

where $t \in \mathbb{R}^{\geq 0}$ represents time and whose solution is a *trajectory* $x : \mathbb{R}^{\geq 0} \mapsto \mathbb{R}^n$. In some cases, trajectories may only be defined for a finite time horizon. The *type* of system is named after the type of f. If f is linear in x and u, we speak of a *linear system*, etc. As above, the input u effectively turns the equation into a differential inclusion.

Hybrid systems. We consider hybrid systems modeled by extending finite state automata with a continuous system in each discrete state. We highlight two important aspects of this extension below, see [104] for the full definition:

- Trajectories within a discrete state must satisfy at all times a so-called *staying condition* or *invariant* $\mathcal{I}$ associated with the discrete state, i.e., $x(\tau) \in \mathcal{I}$ for all time points $\tau \in [0, t]$ in the interval from 0 to time horizon t.
- Transitions from one discrete state to another are only allowed when the continuous state x satisfies an associated *guard* condition $\mathcal{G}$, i.e., $x \in \mathcal{G}$. Note that the system may nonetheless remain in the discrete state as long as its staying condition is satisfied. A transition may instantaneously update the continuous state according to its *reset function*.

The combined effect of staying and guard conditions is to introduce nondeterminism in the timing of state changes. Such nondeterminism can abstract complex or partially unknown real systems with crisp and clean formal models. However, it can also make verifying such systems very difficult or even simulating them numerically. The semantics of hybrid systems are given by *executions*, which are sequences of trajectories associated with discrete states such that each trajectory satisfies the staying condition and transitions (instantaneous state changes) satisfy the guard conditions and reset functions. An alternative way of modeling hybrid systems is with *hybrid programs* and *hybrid games* that mix discrete statements, differential equations, and potentially adversarial dynamics in an imperative programming language.

In the remainder of the paper, we will not distinguish between discrete and continuous states (unless absolutely necessary) and use the symbol x to denote the state of either continuous, discrete, or hybrid systems.

Stochastic systems. We consider stochastic systems that are affected by uncertainty with known probability distributions. The system's behavior can be captured via a state vector $x \in \mathbb{R}^n$, i.e., with n real-valued variables. In a *discrete-time stochastic system*, the state evolves according to the stochastic difference equation

$$x_{k+1} = f(x_k, u_k, w_k), \qquad x_0 \in \mathcal{X}_0, u_k \in \mathcal{U}, \quad k = 0, 1, 2, \ldots \tag{1}$$

where k represents time, $u_k \in \mathcal{U}$ is the input, and $(w_0, w_1, w_2, \ldots)$ is a sequence of independent and identically distributed (iid) random variables representing the

uncertainty affecting the evolution of the system. In a *continuous-time stochastic system*, the state evolves according to the stochastic differential equation

$$dx = f(x,u)dt + g(x,u)dW_t, \qquad x_0 \in \mathcal{X}_0, u \in \mathcal{U}, \tag{2}$$

where time changes continuously, $u \in \mathcal{U}$ is the input, $f(x,u)$ is called the drift, $g(x,u)$ is called the diffusion, and W_t is the Brownian motion representing the stochastic uncertainty.

Equations (1)-(2) represent the simplest structure of models studied in the ARCH Stochastic Category. In general, the stochastic models could have the following different features:

- The timeline could be discrete or continuous.
- The state space could be continuous or hybrid.
- The drift in the differential equation could be linear, piecewise linear, or nonlinear.
- The noise could be Brownian motion or iid. The type of control actions affecting the system could also be different. The control actions could appear as switching signals, or changing the drift of the differential equation, or a combination of these two cases.

2.3 Verification Problems

Most of the problem instances in ARCH-COMP can be characterized as *safety problems*: Given a set of initial states $\mathcal{X}_0$ and a set of *forbidden* or *unsafe* states $\mathcal{F}$, check whether the state remains at all times outside of $\mathcal{F}$. If the system is unsafe, there exists an execution such that for some state x during this execution, $x \in \mathcal{F}$. We call such an execution a *witness* of the violation.

Safety properties can express complex properties (such as bounded liveness) that can be encoded in a so-called *observer* automaton. An observer has an error state that is reachable if (and only if) the property is violated. Setting $\mathcal{F}$ as this error state, safety is equivalent to satisfying the property.

In the case of stochastic models, as those in Eqs. (1) and (2) above, the verification problem is modified into finding the probability of satisfaction of a temporal specification (such as, in the easiest instance, safety); or checking whether such likelihood meets a user-defined threshold in the unit interval.

Set propagation, Reachability, and Model Checking. Reachability algorithms are used to compute a finite representation of a set of states that cover all reachable states; note that there are typically infinitely many reachable states in the dense state space. This may be done similarly to that by which ODEs are solved numerically, i.e., iteratively computing one-step successors. We call this process *set propagation*, and it proceeds as follows: Starting from the set of initial states, a successor set is computed that covers the next states over a given time interval or over the next discrete transition. This process is repeated iteratively to generate, in a tree-like fashion, sets that cover all reachable states. In principle, this tree can be infinite because discrete transitions enter a cycle or continuous time diverges

(assuming we only cover finite time intervals in each step). However, if a set is already covered by previously used initial sets, its successors do not need to be computed again. If all sets are eventually covered, the tree can be finite even though executions over infinite time and infinitely many transitions are covered. Checking for cover by predecessors amounts to a fixed-point check and constitutes a rudimentary form of *model checking*.

For all but the most simple dynamics, it is impossible to cover the solutions of ODEs with a single computable set, even over small time intervals. Instead, the cover is approximate, and the degree of overapproximation (ratio between excess and total states covered) is difficult to quantify precisely, particularly in a hybrid system. We call these covers *approximate reach sets*.

In the case of stochastic models, as those in Eqs. (1) and (2), general verification of temporally extended specifications boils down to computing state-dependent likelihoods, namely computing the probability that trajectories, initialized anywhere in the state space, verify the given temporal specification. This is attained via dynamic programming algorithms, namely leveraging Bellman iterations.

Decision vs reach set approximation. In principle, the job of a verification tool is to decide if the specification is satisfied. In continuous and hybrid systems, the shape and size of the computed approximate reach sets contain plenty of information about the system, and it is common for engineers to plot and inspect them visually. Most verification tools can produce an approximate reach set as part of the output, in addition to the satisfaction of the property.

Bounded vs unbounded problems. In software verification, it is common to distinguish between bounded and unbounded instances. The bound refers to the number of discrete time steps (clock ticks) considered. In a hybrid system, we must distinguish between two types of boundedness that are orthogonal and sometimes confused in the literature:

- *bounded time problems* consider executions or trajectories whose total length, measured by time (imagine a clock running in parallel to the system), do not exceed a given bound,
- *bounded transition problems* consider executions with a bounded number of discrete state changes.

In the above classification, the "bounded model checking" problems from software verification are bounded transition problems. A bounded transition problem may be unbounded in time. A bounded time problem can correspond to an unbounded number of discrete transitions. Even in simple hybrid systems, a system may take infinite discrete transitions in a bounded time interval since discrete transitions are considered instantaneous in (discrete or continuous) time.

Some confusion can arise because the continuous trajectories are only approximated up to a given upper bound in time in most reachability algorithms. This is for practical reasons: The trajectories are covered with sets, each covering a finite time interval. Since only a finite number of sets can be computed, this limits

the time horizon to the sum of the finite time intervals. However, this limitation can be sidestepped entirely if we admit an unbounded number of transitions. The trick is simple: add a clock variable to the system and impose the staying condition to include an arbitrary upper bound $T > 0$ on the clock, as well as self-loop transitions to all discrete states (self-loops do not modify the discrete state); let each self-loop reset the clock to zero. The augmented system thus obtained has exactly the same behavior as the original system, except that the clock ensures that the system does not remain in a discrete state longer than T time before taking a transition. Therefore, it is sufficient to cover continuous-time trajectories up to time T.

To summarize, most tools cover only bounded continuous time intervals between discrete transitions. Many tools consider only bounded transitions, i.e., they do the equivalent of "bounded model checking". All tools that can handle unbounded numbers of transitions, e.g., through fixed-point checking, can also handle problems that are unbounded in time.

Theorem proving. Unlike numerical reachability analysis tools, hybrid systems theorem provers use logical reasoning to analyze fully symbolic and parametric models with unbounded initial sets and for unbounded time. Depending on the expressiveness of the underlying logics, hybrid systems theorem provers can analyze safety properties, liveness properties, stability properties, and game properties. Theorem provers attempt to construct proofs from axioms of programs, differential equations, and arithmetic. Such proofs can typically be conducted interactively, steered with tactics, or attempted fully automatically.

For differential equation analysis, instead of computing reachable sets numerically, theorem provers often use invariant techniques, such as Lyapunov functions and Barrier certificates, to prove that dynamics stay inside certain safe regions. For liveness analysis, progress properties are used. Stability analysis can be expressed as a combination of safety and liveness [168]. Highly trustworthy theorem provers separate searching for such invariant properties from certifying them from axioms, which enables the use of untrusted numerical procedures during search; some theorem provers use invariant search directly as part of their trusted code base or defer this task to the user. In addition to establishing correctness properties about models, auxiliary development tasks, such as stepwise refinement [133], runtime monitoring [146], compilation of models to executable code [35, 92, 167], and derivation of artifacts for machine learning [90, 156], are expressible in logic and supported by some theorem provers.

2.4 Synthesis Problems

Many of the previously mentioned systems exhibit *non-determinism* in the form of uncertainty and/or control actions. For the analysis of stochastic systems, it is relevant to resolve this non-determinism to analyze the system's properties.

Different approaches, often referred to as *schedulers*, *policies*, or *controllers*, exist to resolve non-determinism that affects the behavior of the resulting system. For instance, stochastic schedulers resolve non-determinism according to

a given distribution. State-dependent schedulers, on the other hand, resolve non-determinism based on the current system state. A richer class is given by history-dependent schedulers, which incorporate not only the current state but also past evolution to resolve non-deterministic choices.

In this setup, the synthesis problem demands a scheduler such that the probability of satisfying a given specification is maximized, an expected total cost is minimized, or an expected total reward is maximized.

Note that the synthesis problem is much more complex than the verification problem since we search for optimal schedulers on a possibly uncountable functional space. Therefore, formal and sound solutions are developed to synthesize schedulers with correctness guarantees. The main challenges to consider include: (a) the given specification to be checked on the model could have a finite or infinite time horizon (characterizing the infinite-horizon behavior of the system is more complicated than the finite-horizon behavior); (b) The required specification could go beyond safety and reachability, which induces augmented hybrid models with both continuous and discrete state components; (c) Resolving the probabilistic non-determinism is difficult on the considered benchmarks since the system could be influenced by Brownian motions and Poisson processes; and (d) The evolution of the (augmented) system could be derived from both deterministic and probabilistic equations.

In the case of stochastic models, as those in Eqs.(1) and (2), general synthesis goals reduce to computing optimal policies using Bellman iterations - this is a slight generalization of the approaches and algorithms for verification described above.

2.5 Problem Instances

We distinguish three types of problem instances, each serving a different purpose:

- *Toy problems* mainly serve educational purposes. However, some, such as the bouncing ball, showcase fundamental properties: How reach sets are approximated, how the approximation error and the number of sets increase with each transition, whether there's a fixed-point check, etc. So, to experts, certain toy problems can be quite representative and useful for developing new software tools.
- *Academic benchmarks* serve to illustrate a particular aspect. For instance, a parametric benchmark with varying numbers of discrete states can be useful to compare different tools on this aspect.
- *Industrial benchmarks* ideally originate from actual design problems in application domains. They constitute a Litmus test of whether tools can handle problems of practical interest. However, they are hard to come by and can be biased towards certain classes of problems. They also tend to be either excessively hard or too easy for effective comparisons.

In ARCH-COMP, we use all three types of problem instances to assess tools on a wide range of criteria, while also evaluating their capability to solve problems of practical or industrial interest.

2.6 Inherent Challenges in Evaluating Results

When evaluating verification tools for continuous and hybrid systems in the spirit of the formal verification approach outlined in Sect. 2.1 there are fundamental and practical problems.

Undecidable Certificates. The role of certificates is to provide a way to check that a system satisfies a property. In continuous and hybrid systems, even rudimentary problems such as verifying an invariant in a single discrete state are generally undecidable. Thus, checking the validity of a certificate can be as difficult as checking the property itself. So far, we have not used certificates in ARCH-COMP. However, we use visualizations of reach set approximations for sanity checks, which can be considered a first step in this direction.

Missing Witnesses. A witness is a collection of data, e.g., an execution, that allows one to check that a property is violated by a system. For continuous and hybrid systems, this is problematic in a fundamental way. For all but the simplest classes, trajectories can be described at best by transcendental functions, such as exponential functions of time. The sequence of states that describe a witness would almost surely involve irrational numbers. More practically speaking, a witness produced by a verification can be refuted by a referee tool, but it rarely can be confirmed as an actual execution of the system. Therefore, finding a cover guaranteed to contain at least one solution can be helpful. However, no tool currently outputs such a witness cover.

Differing Semantics. Various modeling and specification formalisms have been proposed for continuous and hybrid systems. A survey from 2005 mentions no less than five fundamentally different semantics for hybrid automata alone [87], and other formalisms for hybrid systems abound [27]. While some differences are more technical, others directly affect whether a problem instance can be effectively characterized and the verification outcome. We have sometimes included different variations, such as continuous-time and discrete-time instances, for inclusivity. Similarly, the stochastic model category has seen the presence of very diverse and semantically rich modeling formalisms and corresponding benchmarks that are not always easy to adapt to other models.

Missing Exchange Format. Because of the multitude of different classes of models and properties, there is no readily applicable way to specify problem instances in a uniform manner. By way of a compromise, some tools have adopted a common syntax based on the hybrid automaton formalism. Efforts to define interchange formats have not seen widespread success [26]. De facto, each tool has its way of specifying models and properties. This is particularly true in the instance of stochastic models.

Hand-crafted Hyperparameters. Most verification algorithms involve many hyper-parameters, such as the time step used for reach set approximation, parameters to

define set approximations (orders of zonotopes or Taylor models), and other ways to abstract from or simplify representations of sets of behaviors. Some attempts at automating the choice of hyperparameters have been made [21,175–179], but in general, the proper way to use many tools remains expert knowledge that is not easily obtained or widely disseminated.

3 Competition Format and Organization

The main objective of ARCH-COMP is to evaluate and compare different approaches across a set of benchmark problems. While performance measures such as time and memory consumption are evaluated, we consider them secondary to the test of whether participants can solve interesting problems – call it the Olympic spirit if you like.

The competition is accompanied by a workshop, which provides a means to propose and present benchmark problems and showcase tools in detail. The workshop maintains a curated benchmark repository, where models and specifications are archived and can be updated where necessary. The accompanying discussions occur in a public electronic forum associated with the workshop.

3.1 A Friendly Format

As we described in Sect. 2.6, we are confronted with many problem classes, diverse solution methods, and a lack of unified interfaces and hyperparameters. Consequently, we opt for the format of a *friendly competition*: Participants create and select problem instances and can tune hyperparameters for each instance before submitting their solution instances in a repeatability package. This ensures that each tool is showcased under its most suitable configuration, which is not necessarily the case when running with a default configuration or when hyperparameters are chosen by the organizers instead.

We have two mechanisms to assure that solution instances are indeed true solutions and not the result of cheating (which we never had) or the lucky result of modeling or specification errors (of which we had several):

- Solution instances are accessible to other participants during the competition and publicly archived afterward. An impressive performance is, therefore, likely to provoke scrutiny by experts. This incentivizes participants to submit solutions that can withstand such scrutiny over time.
- Problem instances can be designed to bracket the computed solutions. Take a problem instance where the model satisfies the specification. A *bracket instance* would be a slight modification of either the model or the specification such that the specification is now violated. If a tool gives correct answers for both bracket instances *with the same set of hyperparameters*, the tool's accuracy lies within the solution space left by the bracket instances. For some classes of problems (like convex invariants), this could be formalized so that correctly solving a set of bracket instances formally guarantees a given level of accuracy of the tool. So far, we have not felt the need to take it to such a formal level.

3.2 Organization and Schedule

The competition is divided into groups that address different problems and will be described in more detail in Sect. 4. Each group is managed by a group leader, who organizes discussion rounds and the joint writing of a final report for each group. All decisions are taken by consensus. This decentralized organization allows each group to develop rules and criteria suitable for their problems.

Each year, the competition follows the following schedule:

1. Following a public call for participation, participants register with a group.
2. The group meets to select problem instances and propose new ones. Groups are encouraged to include benchmarks from the ARCH proceedings.
3. Participants submit preliminary results discussed in a second group meeting. Difficulties and misunderstandings can be resolved in this phase, and surprise performances can be discussed. If necessary, problem instances are refined or clarified.
4. Participants submit their final solution instances in a repeatability package. The evaluation chair runs all packages, and the resulting performance logs are returned to participants. The packages are publicly archived.
5. Each group writes a final report, which includes descriptions of problem instances, performance results, and a discussion of those results. The reports are published in the ARCH proceedings.
6. The competition closes with a presentation by each group leader, given at the ARCH workshop. The audience votes for a winning tool in each group and casts a final vote for the overall most impressive result.

3.3 Artifacts and Results

Problem instances are described in natural language in the ARCH workshop group reports or benchmark papers. Where possible, formal models for each instance are also provided in an online repository in formats recognized by the community [9].

Each participating tool deposits its artifacts for solving the problem instances in the repository (program code or executables, scripts, configuration files, etc.) and a script that runs each instance and provides the result in a given format. Typically, the result is whether the specification is satisfied, not satisfied, or the instance cannot be decided. This script is used in the repeatability evaluation, which will be described in Sect. 5, and to measure the runtimes and memory consumption.

Depending on the group and the problem instances, tools also produce additional output, such as plots of reachable states. We have found this approach helpful to gain further insight, e.g., when one tool unexpectedly outperforms another, for quick and intuitive estimation of the precision of the results, and as a sanity check for surprisingly good or bad performance. Where useful, such plots are included in the group reports.

4 Thematic Groups of the Competition

The competition is organized by groups (tracks), which operate and report independently since they address different problems and solution methods. Each group tackles a different class of models (e.g., linear vs. nonlinear), methods (reach set approximation vs. theorem proving), and objectives (verifying safety vs. falsification). This section provides a summary related to each group of the competition. For each group, we report its (i) goal, i.e., the problem addressed by the track, (ii) benchmarks, i.e., the benchmark problems considered in the competition; (iii) participants, i.e., the tools participating in the track; and (iv) outcome, i.e., a description of the outcomes produced by each track across the years. Table 1 reports the tools participating in each group for each year of the competition. Table 2 reports the number of benchmarks considered for each group and year. We do not report illustrative graphics for the results of each group since the goal of this work is to provide general reflections about the competition and not to discuss the results in detail. The reader can refer to the corresponding group reports for this analysis.

4.1 Piecewise Constant Dynamics

Goal. In ARCH-COMP, we have a track PCDB for *continuous and hybrid systems with piecewise constant dynamics* (HPCD) and *bounded model checking* (BMC) of HPCD systems. The PCDB category concerns hybrid systems where in each location (mode, piece of the hybrid state space), the dynamics are given by a differential inclusion of the form $\dot{x}(t) \in \mathcal{U}$, where $\mathcal{U}$ is a convex subset of $\mathbb{R}^n$. Specifically, the BMC task concerns the bounded model checking of HPCD systems where the bound is described as the depth of the discrete jump of the system. The verification specifications used in PCDB track focus on the (bounded) reachability verification, e.g., whether two processes can be in the critical section at the same time, or whether a vehicle can reach a specific dangerous position, and so on. As the main techniques, implementations used by different checkers vary from each other, the goal of the PCDB track is to present the landscape of existing solutions in a breadth and showcase the current state of the art.

Benchmarks. The benchmark collection has evolved continuously with each edition of the competiton [41, 42, 45–47, 84–86]. Since HPCD and BMC are two parallel tracks at the beginning, and merged to PCDB in the edition of 2020, the benchmarks contain both unbounded and bounded specifications for each cases. Most of the benchmarks are extendable, so that the models can be concretized with different values of parameters to increase the difficulty of the problem in the aspects of number of continuous variables, number of discrete locations, and number of components in the system. Thus, the benchmarks can be used in the evaluation of the efficiency, scalability of different tools.

Participants. Throughout the years, different checkers have joined the competition, while not everyone of them participated every year. These tools are

Table 1: Tools participating in each group for each year of the competition

	Participating Tools						
Category	**2017**	**2018**	**2019**	**2020**	**2021**	**2022**	**2023**
Piecewise Constant Dynamics	BACH [43, 44], Lyse [33], PHAVer/SX [82], VeriSiMPL [6, 7].	BACH [43, 44], Lyse [33], PHAVer/SX [82], PHAVerLite [25], VeriSiMPL [6, 7].	BACH [43, 44], HyCOMP [55], Lyse [33], PHAVer/SX [82], PHAVerLite [25], VeriSiMPL [6, 7].	BACH [43, 44], PHAVer/SX [82], PHAVerLite [25], ROPICAL [149], XSpeed [158].	N/A	BACH [43, 44], PHAVer/SX [82], PHAVerLite [25], SAT-Reach [158], XSpeed [158].	N/A
Linear Dynamics	Axelerator [49] CORA [10] Flow* [51] HyDRA [161] Hylaa [22] SpaceEx [83] XSpeed [158]	CORA [10] CORA/SX [14] C2E2 [74] Flow* [51] HyDRA [161] Hylaa [22] JuliaReach [31] SpaceEx [83] XSpeed [158]	CORA [10] CORA/SX [14] HyDRA [161] Hylaa [22] JuliaReach [31] SpaceEx [83] XSpeed [158]	CORA [10] C2E2 [74] HyDRA [161] Hylaa [22] Hylaa-Continuous [23] JuliaReach [31] SpaceEx [83] XSpeed [158]	CORA [10] HyDRA [161] JuliaReach [31] SpaceEx [83]	CORA [10] JuliaReach [31]	CORA [10] JuliaReach [31] Verse [132].
Nonlinear Dynamics	CORA [10] Flow* [51] Isabelle /HOL [109]	CORA [10] CORA/SX [14] C2E2 [74] Flow* [51] Isabelle /HOL [109] SymReach [72]	Ariadne [40] CORA [10] DynIbex [160] Flow* [51] Isabelle /HOL [109] JuliaReach [31]	Ariadne [40] CORA [10] DynIbex [160] Flow* [51] Isabelle /HOL [109] JuliaReach [31]	Ariadne [40] CORA [10] DynIbex [160] JuliaReach [31] Kaa [125]	Ariadne [40] CORA [10] DynIbex [160] JuliaReach [31] Kaa [125] KeYmaera X [89]	Ariadne [40] CORA [10] DynIbex [160] JuliaReach [31] KeYmaera X [89] Verse [132] .
AINNCS	N/A	N/A	NNV [136, 170], Sherlock [63–65], Verisig [112, 113],	NNV [136, 170], OVERT [164], ReachNN* [75, 106], VenMAS [8].	JuliaReach [31], NNV [136, 170], Verisig [112, 113],	CORA [10, 126], JuliaReach [31], NNV [136, 170], POLAR [105].	CORA [10, 126], JuliaReach [31], NNV [136, 170].
Stochastic Models	N/A	Barrier [114], FAUST^2 [166], FIRM − GDTL [131], Modest [103], SDCPN&IPS [71], SReachTools [172].	FAUST^2 [166], HYPEG [153], LyapMMC [159], Modest [103], SDCPN&IPS [71], SReachTools [172], StocHy [50], SySCoRe [102].	AMYTISS [129], FAUST^2 [166], hpnmg [108], Mascot-SDS [140], Modest [103], ProbReach [163], SDCPN&IPS [71, 138], SReachTools [172], StocHy [50].	AMYTISS [129], FIGARO [36, 37], hpnmg [108], HYPEG [153], Mascot-SDS [139, 140], Modest [103], ProbReach [163], PyCATSHOO [53], SDCPN&IPS [71, 138], SReachTools [172], StocHy [50], SySCoRe [101].	FIGARO [36, 37], HYPEG [153], Modest [103], RealySt [58], PyCATSHOO [53], SDCPN&IPS [71, 138], SySCoRe [101].	HYPEG [153], ProbReach [163], RealySt [58], SySCoRe [171].
Falsification	S-TaLiRo [19].	S-TaLiRo [19], FalStar [70].	Breach [61], FalStar [70], falsify [181], S-TaLiRo [19].	ARIsTEO [142], Breach [61], falsify [181], FalStar [70], S-TaLiRo [19], zlscheck [183].	ARIsTEO [142], Breach [61], FalCAuN [173], falsify [181], FalStar [70], ForeSee [182], S-TaLiRo [19], Ψ-TaLiRo [169], zlscheck [183].	ARIsTEO [142], FalCAuN [173], falsify [181], FalStar [70], ForeSee [182], S-TaLiRo [19], Ψ-TaLiRo [169].	ARIsTEO [142], ATheNA [78], FalCAuN [173], ForeSee [182], NNFal [151], STGEM [152], S-TaLiRo [19], Ψ-TaLiRo [169].
Hybrid Systems Theorem Proving	N/A	KeYmaera X [89] KeYmaera 3 [155] HHL Prover [174]	KeYmaera X [89] KeYmaera 3 [155] HHL Prover [174]	KeYmaera X [89] KeYmaera 3 [155] HHL Prover [174], HybridVCs [150]	KeYmaera X [89] HHL Prover [174]	KeYmaera X [89] HHL Prover [174] HHLPy [162], IsaVODEs [79]	KeYmaera X [89], HHL Prover [174], HHLPy [162], IsaVODEs [79]

Table 2: Number of benchmark models considered for each group and year

	Number of benchmark Models						
Category	2017	2018	2019	2020	2021	2022	2023
Piecewise Constant Dynamics	5	5	5	6	N/A	6	N/A
Linear Dynamics	3	6	6	8	9	9	8
Nonlinear Dynamics	3	4	4	6	5	6	6
AINNCS	N/A	N/A	5	7	7	10	10
Stochastic Models	N/A	5	4	7	10	6	2
Falsification	1	1	6	7	7	6	7
Hybrid Systems Theorem Proving	N/A	139	169	214	214	220	221

using different techniques, e.g. SMT encoding and solving based, polyherdal based geometric computation, support function based verification, and so on. Past participant tools in this category in alphabetical order are BACH [43, 44], HyCOMP [55], HyDra [161], Lyse [33], PHAVer/SX [82], PHAVerLite [25], SAT-Reach [158], SpaceEx [83], TROPICAL [149], VeriSiMPL [6, 7] , XSpeed [158].

Outcome. In the evaluation reports, we can see the size of individual automata that can be solved has increased significantly. It will be an important topic to evaluate whether existing methods/tools can handle large compositional system efficiently. Besides of the evaluation results, the PCDB track has achieved a stable benchmark set in a well recognized format of models.

4.2 Continuous and Hybrid Systems with Linear Dynamics

Goal. While there exists an analytical solution for piecewise constant dynamics, the solution of linear systems can be computed exactly in some specific cases only, e.g., when all eigenvalues are real or imaginary [128]. However, linear dynamics can be considered the most straightforward dynamics besides piecewise constant dynamics because the superposition principle can be used, the homogeneous solution can be computed analytically using the matrix exponential, and the convexity of reachable sets of points is preserved in time. This makes it possible to use convex set representations and compute the reachable set without the wrapping effect [99]. Because the verification of purely linear systems is already fairly well understood, the goal of this track is to assess how to scale the computation using order reduction methods with formal error bounds and decomposition techniques [11, 23, 32]. The verification of hybrid systems with linear dynamics is much less understood. While novel ideas have already been proposed to solve the problem of precise and scalable guard intersection [16, 20, 98], benchmarking different concepts for guard intersections is a primary goal. Finally, a further goal is to evaluate fully automatic verification processes as proposed, e.g., in [175, 177].

Benchmarks. Over the years, we have added more challenging benchmarks while some easy benchmarks have been removed [12–15, 17, 18]. In particular, we have added high-dimensional problems with up to one million state variables and removed low-dimensional ones, such as the building benchmark with 48 state variables. In general, the number of continuous state variables for hybrid systems is much lower than for purely continuous systems due to the difficulty of computing precise and scalable guard intersections. In the 2023 edition, three benchmarks were purely continuous, while five were hybrid. The hybrid benchmarks cover cases where guards trigger the discrete transitions and where the discrete transition changes can happen arbitrarily.

Participants. So far, eleven tools have participated in this category. All tools essentially use some form of reachability analysis. Past participating tools in this category in alphabetical order are Axelerator [49], CORA [10], C2E2 [62], Flow* [51], HyDRA [161], Hylaa [22], Hylaa-Continuous [23], JuliaRech [31], SpaceEx [83], Verse [132], and XSpeed [158].

Outcome. By comparing the results in the yearly competitions, each tool could improve more than without the gained insights. The number of state variables that can be considered today is several orders of magnitude larger than what we could verify in the competition's first edition (2017). Some verification results are now automatically obtained, which was realized for the first time in 2023. Besides these scientific achievements, several achievements regarding the reproducibility of results have been pioneered in this category. This includes using the same modeling format for all benchmarks (SpaceEx model definition), Docker files for executing all results and fully automatically evaluating Docker files on the same server to better compare computation times.

4.3 Nonlinear Dynamics

Goal. As opposed to piecewise constant and linear systems, nonlinear differential systems do not have an analytical solution and the superposition principle is not applicable. As a result, in the numerical case, an (over-)approximated solution is typically computed. Given the nonconvexity of sets in general, set representation is particular important. Specifically, how to provide a finite polynomial approximation that introduces a small remainder for rigorous analysis. Currently, Taylor polynomials [28] are the most commonly used representation for the enclosure of nonlinear sets, although there has been some work on Bernstein polynomials in recent years [57]. Given the sensitivity of those approximations to the problem at hand, lately some work has been done in automating numerical reachability parameters [93, 179].

Since the majority of the tools in this category fall into the numerical approach, the main goal is to show improvements in the ability to represent the finite-time reachable set as tightly as possible within reasonable computation times. While most of the problems with nonlinearity are associated to continuous evolution, hybrid evolution also introduces its own challenges. Nonlinear guards and their

intersections are not trivially understood, with respect to varying concavity and the difficulty to handle tangential crossings. Consequently, a secondary goal has been to capture those critical cases and evaluate how to effectively address them.

Benchmarks. The benchmarks suite has steadily evolved over the years [52, 94–97, 110, 111]. The general policy for choosing benchmarks is to allow as many tools as possible to return an answer to the corresponding verification problem. Given the mixed numerical and symbolic approaches used, this objective can be particularly daunting. As a result, some benchmarks have been adapted across the years to become more/less challenging based on average progress from the existing participants or the presence of new participants. To explicitly manage this versioning, all benchmarks are currently identified by a four letter contraction of the name followed by two digits representing the most recent year they were updated.

Recently, there has been a greater effort in two specific directions: addressing inherent issues with numerical stability (on the continuous side) and assessing the quality of sets after transitions (on the hybrid side). Since 2020, out of the average six benchmarks, two are specifically hybrid to deal with transverse crossings (LOVO21) and large sets crossings (SPRE22), respectively.

Participants. Throughout the years, 12 different tools joined the competition. Most tools had a purely numerical approach, with a minority using symbolic approaches. Typically there have been 6 participants each year. Past participating tools in this category in alphabetical order are Ariadne [40], C2E2 [62], CORA [10], CORA/SX [14], DynIbex [160], Flow* [51], Isabelle/HOL [109], JuliaReach [31], Kaa [125], KeYmaera X [89], SymReach [72] and Verse [132].

Outcome. The interaction between research groups that was solicited from the competition (during it and outside of it) produced improvements in many tools across the years. While there has been no particular focus on increasing the number of variables handled, combinations of dynamics and large initial sets originally not addressable progressively became the standard for verification. There is still a significant amount of work to be done on numerical tools to support some categories of transitions necessary to analyze hybrid systems. On the other side, some symbolic tools were not originally designed to work with hybrid dynamics either, requiring extra effort to support the existing benchmarks.

4.4 Artificial Intelligence and Neural Network Control Systems (AINNCS)

Goal. Autonomous systems increasingly incorporate artificial intelligence (AI) and machine learning components, such as neural networks (NNs), for various sensing/measurement and control tasks ranging from perception, sensor fusion, planning, and feedback control. Control theory, particularly in the intelligent control area, investigated significantly the usage of neural networks as feedback controllers. This category primarily considers such neural network control systems

(NNCS), where a feedback control policy is designed and implemented as a neural network providing input to some plant, the latter of which is modeled as differential equations or hybrid systems. Specifications considered have been safety properties (invariants) and some limited reachability properties. The category was first held in 2019 [137] and has been held annually since then [123, 124, 134, 135], with varying participants over the years.

Benchmarks. The benchmarks have varied over the years of the competition. Most benchmarks consist of a continuous-time and continuous-state plant modeled using linear or nonlinear ODEs, with properties—mostly safety—defined as predicates over the state-space of the plant model. Most of the plant models have been of low dimensionality, with around two to ten state variables. Typically, the neural network controller generates inputs for the plant model periodically, in a sample-and-hold fashion, so the control input produced by the neural network is applied to the plant for the entire control period. Most controllers have consisted of fairly small (orders of tens to hundreds) of neurons, mostly with ReLU activation functions. A few benchmarks have varied from this typical structure, with some in discrete time.

Participants. The participating tools have varied over the years of the AINNCS category. The tools that have participated in the AINNCS category in any year since 2019 are (in alphabetic order): COntinuous Reachability Analyzer (CORA) [10, 126], JuliaReach [31], NNV (Neural Network Verification Tool) [136, 170], Sherlock [63–65], POLAR [105], OVERT [164], ReachNN* [75, 106], VenMAS [8], and Verisig [112, 113]. One tool has participated in every iteration (NNV), and several tools have participated in two or more iterations (CORA, JuliaReach, POLAR/ReachNN*, and Verisig).

Outcomes. The AINNCS category has led to several outcomes in the verification of systems controlled by neural networks. The first major contribution is the support of this new research area and community for verification of hybrid and continuous systems that use AI components, where now nine verification tools have participated over the years. The next significant contribution is the curation of a set of around a dozen challenging benchmarks now, that have been used within the community in the development of new methods, for comparisons and motivation of new challenges. Within this has been standardization effort for the representation of the neural network controllers, specifically in the ONNX format, along with representation of the plant models in interchange formats. Another outcome is the identification of significant challenges in this space, for example, the scalability in both the sizes of the neural network controllers and the plant dimensionality currently are fairly low, and motivate the development of further scalable verification methods and effective abstractions that are not overly conservative. There are many such challenges to address for the AINNCS category in future iterations of the competition that we hope to continue in the next iterations of ARCH-COMP.

4.5 Stochastic Models

Goal. Approaches and tools in the ARCH stochastic category aim to verify and synthesize systems that combine discrete, continuous, and stochastic behavior. In this context, *verification* tries to answer questions about the probability of reachability and other specifications. An example of such specifications is to check whether "the probability to reach a subset of the state space A where variable $x \geq 3$ holds is larger than 0.8." Such specifications are encoded appropriately in formal specification languages. Furthermore, part of this category is dedicated to solving *synthesis* problems, i.e., finding a suitable resolution of existing non-determinism such that a given specification (as before) is satisfied. Apart from evaluating tools, each year the participants of the category decide on a goal that serves the community such as categorizing and classifying benchmarks or, more recently, the development of a set of toy examples in different modeling formalisms that each participating tool can solve.

Benchmarks. The set of benchmarks as described in previous reports [1–5] incorporates eleven systems from different communities, which have been collected by the participants over the years. Several benchmarks are also known from other communities but have been extended by including stochastic behaviors. Others come from an industrial application, e.g., from energy systems, robotics, or healthcare. As the approaches and goals of the participating tools vary, the collection of benchmarks exhibits a diverse set of challenges such that there is currently no tool that can solve all benchmarks.

Participants. Up to this day, 15 tools have participated in this category. While not every approach participated every year, on average five tools have taken part in the event. Furthermore, we have hosted various experts from the field, taking part in the bi-weekly meetings and contributing to discussions. Tools that participate are based on a diverse set of theoretical approaches, such as abstraction-based methods, simulation relations, coupled stochastic relations, statistical model checking, rare event simulation, kernel-embedding methods, and approaches based on reachability computation (stochastic extension of classic reachability analysis for hybrid system). We refer the reader to the paper [130] for a survey on the theoretical developments of these approaches on formal verification and synthesis of stochastic systems. Past participating tools in the ARCH stochastic category in alphabetical order are AMYTISS [129], FAUST^2 [166], Figaro [38], hpnmg [108], HYPEG [153], Mascot-SDS [139], modes [48], ProbReach [163], prohver [80], PyCATSHOO [54], RealySt [58], SDCPN&IPS [30], SReachTools [172], StocHy [50], and SySCoRe [171].

Outcome. Over the years, apart from benchmark evaluation, the ARCH stochastic category has fostered a lively exchange on community-relevant topics that has resulted in several initiatives besides the main goal of the competition. One central aspect in this regard has been the discussion of different modeling paradigms for stochastic systems, how to represent a given system within these modeling

paradigms, and whether an interchange/exchange format could be developed, as done earlier for other models. This overall goal, however, is arguably harder than in other categories, in view of the semantical richness and diversity of stochastic (and additionally hybrid) modeling frameworks.

4.6 Falsification

Goals. It targets the black-/greybox analysis of executable models considering requirements expressed in temporal logic with time bounds, encoded in metric temporal logic (MTL [127]) or signal temporal logic (STL [141]) over a finite time horizon. The participants need to find initial conditions and time-varying inputs subject to certain constraints that steer the system into a violation of the respective requirement. The goal is to compare how quickly tools find witness signals for such violations, and moreover to compare the statistical variability in these results, since many approaches are stochastic.

Benchmarks. As described in past reports [59, 60, 66–69], the benchmark set has been growing continuously and encompasses seven system models, including some well-known models in the automotive domain that have been widely used in the literature [115]. Each comes with a number of requirements, comprised of a definition of the search space of input signals, as well as the temporal logic formula to be falsified. The models encompass a variety of difficulties, such as nonlinear or discontinuous behavior, and large search spaces.

Participants. The participants typically rely on simulation-based approaches and employ quantitative metrics [73,81] to measure how close a given input is to violating a requirement ("robustness semantics"). Research in this area has produced a variety of techniques, mature tools, and practical applications; these are described in overview survey articles [24, 56]. More recently, approaches based on system identification have participated, which first learn a surrogate model of the behavior, over which the falsification problem is easier to solve. Past participants of the competition include ARIsTEO [142], ATheNA [78], Breach [61], FalCAuN [173], ForeSee [182], NNFal [151], STGEM [152], falsify [181], FalStar [70], S-TaLiRo [19], Ψ-TaLiRo [169], and zlscheck (based on Zélus [39]).

Outcome. The falsification category has achieved a stable benchmark set that is gradually becoming a standard in the falsification literature. Moreover, recently a validation step [66] has been introduced to detect any discrepancies between tools, simulation environments, model versions, and experimental setup; all major such issues have been found and fixed. This increases trust in the empirical comparison, which is available now for a wide range of tools as a reference.

4.7 Hybrid Systems Theorem Proving

Scope and Goals. The characteristic feature of the hybrid systems theorem proving category is its emphasis of programming languages as structuring principles for

hybrid systems. The unambiguity and precision of program language semantics paves the way for mathematical rigor of logical reasoning principles. Typical approaches in this category perform deduction based on a program logic for hybrid systems and hybrid games, such as differential dynamic logic (dL [154]) or Hybrid Hoare Calculus [100]. Unlike other categories in the competition, hybrid systems theorem proving uses *fully symbolic, non-deterministic, parametric models*, and focuses on *infinite-time* verification from *unbounded starting states*. As a result, the proofs in this category typically identify fundamental design characteristics of the analyzed models, such as loop invariants and invariant properties of differential equations. The examples vary in scale from basic hybrid programs to industrial case studies, such as verification of collision avoidance in autonomous ground vehicles. The correctness specifications in this category express necessity (safety, programs only reach safe states), eventuality (liveness, programs eventually reach goal states), and winning strategies (games, one of the competing players wins). Future extension to stability [168] is planned.

Benchmarks. The competition benchmark set includes common design shapes at a small scale to test theorem proving base functionality, nonlinear and parametric continuous models to assess the continuous reasoning capabilities, hybrid games, and full-scale hybrid systems case studies, each in three modes:

- fully automatic verification without any additional input beyond the original hybrid system and its safety specification (in particular, without any proof scripts or other parametrization of the proof procedures);
- semi-automatic verification from design insights that are annotated to the original problem specification, allowing users to communicate specific advice about the system such as loop invariants;
- proof checking from proof scripts, which perform a significant part of the verification or provide problem-specific proof tactics.

Recent editions of the benchmarks in this category [144, 147, 148] use select examples to make a more detailed side-by-side comparison of modeling features and verification approaches.

Participants. Benchmark examples in this category are written in differential dynamic logic [154] which has axioms and an unambiguous semantics available in Isabelle/HOL and Coq [34], and in KeYmaera 3 [155] and KeYmaera X [89]. If known, the benchmark examples come with proof hints for semi-automatic verification and proof scripts for proof checking. A tutorial on the modeling principles in differential dynamic logic can be found in [157], whereas details on the ASCII syntax are in [145]. From this common format, participating tools translate benchmark examples into their own input syntax. Participants in this category included KeYmaera X [89] with Pegasus invariant generator [165], Bellerophon tactic script language [88], implicit definitions of functions [91], and implicit and explicit proof management [143]; IsaVODEs [79, 107]; HHL Prover [174], and HHLPy [162].

Outcome. One of the benefits of hybrid systems theorem proving is its transparency in terms of human-inspectable proofs or disproofs that justify why a hybrid systems model does or does not satisfy the desired properties. Many significant theorem proving results (in general, not only for hybrid systems) were obtained with manual guidance to find such proofs, which highlights another benefit: even if fully automated tools may get stuck, theorem proving with manual guidance can still make progress. For benchmarking purposes, however, this poses a challenge when it comes to comparing the reasoning performance of hybrid systems theorem provers.

For full performance comparison transparency, the hybrid systems theorem proving category introduced the automated mode, in which tools are required to find proofs in their default configuration automatically without hints from a user. In automated mode, the number of solved examples and their duration reflect the capabilities that tools provide to novice users or users that focus on modeling, but not on proving. A challenge related to proof automation is proof portability: since tools generally integrate a mix of proof search techniques whose use is typically balanced with timeouts, the compute power of a machine searching and checking a proof may influence whether or not a proof can be found (for example, a proof search heuristic may succeed within its default timeout on one machine, but abort with a timeout on a lesser machine).

Naturally, even as automation made significant advances over the course of the competition, the number of proofs found fully automated remained lower than in interactive or hints mode, since better automation lets users become more efficient in doing manual proofs. The number of solved examples and the tactic script lengths in hints and interactive mode are indicative of the state-of-the-art performance and the user effort necessary to achieve such results.

In summary, theorem proving is a complementary technique to reachability analysis that can find concise, easily checkable, and human-interpretable correctness proofs with significantly lower computation time [96]. Progress in case studies of significant size is achievable with human guidance. Recent competition instances additionally introduced a qualitative side-by-side comparison of modeling features, semantic similarities and differences, and proof examples to provide better intuition about the mechanics of modeling and conducting proofs in different theorem proving systems.

5 Repeatability

Repeatability aims to enable the replication of results and experiments on the same (or another platform) and provides additional evidence of the validity of the results. The submission and collection of the benchmarks, execution scripts, installation scripts, and corresponding instructions may provide the capability for future researchers to build upon and reuse these computational results for other purposes, for instance, in comparisons in research papers.

The competition makes significant efforts to ensure the repeatability of the experiments. Each iteration of the competition has had a repeatability evaluation

process to validate the results of the participants in the different categories, the specific details of which may be found in each iteration's repeatability evaluation report [116–122]. The general repeatability process and its goals are similar to artifact and repeatability evaluations done in some computer science conferences over the past decade or so (see e.g. [180]), to enhance trust and validity of computational results and make available computational artifacts that support claims and conclusions made through research papers and reports. The process has evolved over the years, initially beginning with a significant manual effort, to near full automation in the most recent iterations of the competition.

Generally, the evaluation has been led primarily by the evaluation chair (Taylor Johnson), who installed and reran the tools on the benchmarks across all the categories, using artifacts provided by the participants through the centralized repository [9]. In the early years, this process was performed by providing installation scripts for the tools as well as execution scripts for the benchmarks, which were then installed on a virtual machine and executed on a laptop, requiring a significant amount of manual effort (around an hour per tool to install and then execute the benchmarks). The process evolved to require Docker files to be submitted so that Docker containers could be built automatically, then with batch execution scripts for all the benchmarks for a given tool in a given category. With the Dockerized setup, the manual intervention typically required around a day of effort to set everything up and start batch execution, with typically around a week of total time required for execution of all benchmarks across all categories. Once standardized with the Dockerized execution, we typically have run our experiments on an Amazon Web Services (AWS) Elastic Compute Cloud (EC2) g4dn.4xlarge machine with an Intel Xeon Scalable (2nd Generation Cascade Lake), 16 vCPUs, 2.5 GHz base, (AWS/EC2 custom chip, roughly Xeon Gold 5200 Series with 24 physical cores) and 64GB RAM. On this platform, the execution runtimes in a given category varied depending on the benchmarks, with some typically completing in seconds with others requiring hours of execution. Further improvements include executing all tools across all the supported benchmarks in each category, along with some collection of performance metrics (runtime, verification result where applicable, etc.).

6 Overall Achievements and Outlook

Since 2017, the series of friendly ARCH-COMP competitions has matured and gained some routine around the yearly schedule. The event enjoys a steady popularity around an active community. New teams with new approaches and tools enter the competition yearly, while categories become more firmly established. For example, for the falsification group, the number of tools participating in the competition (see Table 1) increased from one tool (2017) to eight tools (2021 and 2023). Thanks to theoretical advances and tool improvements, the scale of treatable problems has increased by several orders of magnitude, e.g., in the categories for piecewise continuous, linear, and non-linear categories. In some

instances, the insights gained by investigating the results of the competition played an important role.

All groups have worked to increase the number and diversity of the available benchmarks. For example, for the falsification group, the number of benchmark models (see Table 2) increased from one model (2017) to seven models (2020, 2021, and 2023). They are now collected in a central, shared repository for easier access [9]. These benchmarks are regularly used in publications for evaluating new approaches, improving the relevance of such experiments and the comparability across different publications. ARCH-COMP has also gained some visibility in the industry, tools are increasingly applicable to industrial products, and some contributions to the associated workshop are closely related to the competition. For example, recent work [77] enables the use of falsification-based testing techniques with Test Sequence and Test Assessment Blocks, i.e., Simulink blocks commonly used by industrial practitioners to test their models.

Progress has also been made for the concerns of fair evaluation, as discussed in section 2.6: Many groups have established standard formats for problem specifications and exchange formats for witnesses, which increases trust in the results. Similarly, the efforts towards complete repeatability of the evaluation, presented in section 5, are progressing with each installment of ARCH-COMP. Most groups now have automated at least some aspects of this process.

While the competition has seen a lot of progress over the years, some challenges remain. In some categories, fully automating the repeatability evaluation requires other issues to be resolved first, such as standardizing input and witness formats. Due to the diversity and complexity of methods and problems, this, in turn, will require further theoretical and technical advances complemented by implementation work. This is in particular more evident in the Stochastic Category due to the large differences between the class of models and problem formulations on these models. This category is pushing activities on theoretical analysis of stochastic models, developing new tools based on these advances, moving towards an interchange format of stochastic models, and importing minimal case studies where methods developed for different model classes can be compared. With its collaborative spirit, the friendly ARCH-COMP competition will do its best to nurture discussions and collaborations that help to tackle these challenges in the rich and rewarding field of continuous and hybrid systems.

Acknowledgements

We are grateful to all the participants in all the iterations of ARCH-COMP. In addition, we gratefully acknowledge financial support by the project TRAITS, funded by the German Federal Ministry of Education and Research under grant number 01IS21087 and by the French Agence Nationale de Recherche under grant number ANR-21-FAI1-0005-01. Some material presented in this paper is based upon work supported by the National Science Foundation (NSF) through grant numbers 2028001, 2220401, and 2220426, the Defense Advanced Research Projects Agency (DARPA) under contract number FA8750-23-C-0518, and the Air Force

Office of Scientific Research (AFOSR) under contract number FA9550-22-1-0019 and FA9550-23-1-0135.

References

1. Abate, A., Blom, H., Bouissou, M., Cauchi, N., Chraibi, H., Delicaris, J., Haesaert, S., Hartmanns, A., Khaled, M., Lavaei, A., Ma, H., Mallik, K., Niehage, M., Remke, A., Schupp, S., Shmarov, F., Soudjani, S., Thorpe, A., Turcuman, V., Zuliani, P.: Arch-comp21 category report: Stochastic models. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21). EPiC Series in Computing, vol. 80, pp. 55–89. EasyChair (2021). https://doi.org/10.29007/dprv
2. Abate, A., Blom, H., Cauchi, N., Degiorgio, K., Fraenzle, M., Hahn, E.M., Haesaert, S., Ma, H., Oishi, M., Pilch, C., Remke, A., Salamati, M., Soudjani, S., van Huijgevoort, B., Vinod, A.: ARCH-COMP19 category report: Stochastic modelling. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH19). EPiC Series in Computing, vol. 61, pp. 62–102. EasyChair (2019). https://doi.org/10.29007/f2vb
3. Abate, A., Blom, H., Cauchi, N., Delicaris, J., Hartmanns, A., Khaled, M., Lavaei, A., Pilch, C., Remke, A., Schupp, S., Shmarov, F., Soudjani, S., Vinod, A., Wooding, B., Zamani, M., Zuliani, P.: Arch-comp20 category report: Stochastic models. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20). EPiC Series in Computing, vol. 74, pp. 76–106. EasyChair (2020). https://doi.org/10.29007/mqzc
4. Abate, A., Blom, H., Cauchi, N., Haesaert, S., Hartmanns, A., Lesser, K., Oishi, M., Sivaramakrishnan, V., Soudjani, S., Vasile, C.I., Vinod, A.P.: ARCH-COMP18 category report: Stochastic modelling. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH18). EPiC Series in Computing, vol. 54, pp. 71–103. EasyChair (2018). https://doi.org/10.29007/7ks7
5. Abate, A., Blom, H., Delicaris, J., Haesaert, S., Hartmanns, A., van Huijgevoort, B., Lavaei, A., Ma, H., Niehage, M., Remke, A., Schön, O., Schupp, S., Soudjani, S., Willemsen, L.: ARCH-COMP22 Category Report: Stochastic Models. vol. 90, pp. 113–141. EasyChair (2022). https://doi.org/10.29007/LSVC
6. Adzkiya, D., Abate, A.: VeriSiMPL: Verification via biSimulations of MPL models. In: International Conference on Quantitative Evaluation of Systems. Lecture Notes in Computer Science, vol. 8054, pp. 253–256. Springer (2013)
7. Adzkiya, D., Zhang, Y., Abate, A.: VeriSiMPL 2: An open-source software for the verification of max-plus-linear systems. Discrete Event Dynamic Systems **26**(1), 109–145 (2016). https://doi.org/10.1007/s10626-015-0218-x
8. Akintunde, M.E., Botoeva, E., Kouvaros, P., Lomuscio, A.: Formal verification of neural agents in non-deterministic environments. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS. pp. 25–33 (2020)
9. et al., G.F.: ARCH-COMP repository of benchmark models, documentation, and repeatability packages. https://gitlab.com/goranf/ARCH-COMP
10. Althoff, M.: An introduction to CORA 2015. In: Workshop on Applied Verification for Continuous and Hybrid Systems. p. 120–151 (2015)
11. Althoff, M.: Reachability analysis of large linear systems with uncertain inputs in the Krylov subspace. IEEE Transactions on Automatic Control **65**(2), 477–492 (2020)

12. Althoff, M., Bak, S., Bao, Z., Forets, M., Frehse, G., Freire, D., Kochdumper, N., Li, Y., Mitra, S., Ray, R., Schilling, C., Schupp, S., Wetzlinger, M.: ARCH-COMP20 category report: Continuous and hybrid systems with linear continuous dynamics. In: International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 74, p. 16–48 (2020)
13. Althoff, M., Bak, S., Cattaruzza, D., Chen, X., Frehse, G., Ray, R., Schupp, S.: ARCH-COMP17 category report: Continuous and hybrid systems with linear continuous dynamics. In: International Workshop on Applied Verification for Continuous and Hybrid Systems. p. 143–159 (2017)
14. Althoff, M., Bak, S., Chen, X., Fan, C., Forets, M., Frehse, G., Kochdumper, N., Li, Y., Mitra, S., Ray, R., Schilling, C., Schupp, S.: ARCH-COMP18 category report: Continuous and hybrid systems with linear continuous dynamics. In: International Workshop on Applied Verification for Continuous and Hybrid Systems. p. 23–52 (2018)
15. Althoff, M., Bak, S., Forets, M., Frehse, G., Kochdumper, N., Ray, R., Schilling, C., Schupp, S.: ARCH-COMP19 category report: Continuous and hybrid systems with linear continuous dynamics. In: International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 61, p. 14–40 (2019)
16. Althoff, M., Krogh, B.H.: Avoiding geometric intersection operations in reachability analysis of hybrid systems. In: Hybrid Systems: Computation and Control. p. 45–54 (2012)
17. Althoff, M., Ábrahám, E., Forets, M., Frehse, G., Freire, D., Schilling, C., Schupp, S., Wetzlinger, M.: ARCH-COMP21 category report: Continuous and hybrid systems with linear continuous dynamics. In: International Workshop on Applied Verification of Continuous and Hybrid Systems. vol. 80, p. 1–31 (2021). https://doi.org/10.29007/lhbw, https://easychair.org/publications/paper/81BS
18. Althoff, M., Forets, M., Schilling, C., Wetzlinger, M.: ARCH-COMP22 category report: Continuous and hybrid systems with linear continuous dynamics. In: International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 90, p. 58–85. EasyChair (2022). https://doi.org/10.29007/mmzc, https://easychair.org/publications/paper/b6cN
19. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 254–257. Springer (2011)
20. Bak, S., Bogomolov, S., Althoff, M.: Time-triggered conversion of guards for reachability analysis of hybrid automata. In: International Conference on Formal Modelling and Analysis of Timed Systems. p. 133–150 (2017)
21. Bak, S., Bogomolov, S., Johnson, T.T.: HYST: a source transformation and translation tool for hybrid automaton models. In: Proc. of the 18th International Conference on Hybrid Systems: Computation and Control (2015)
22. Bak, S., Duggirala, P.S.: HyLAA: A tool for computing simulation-equivalent reachability for linear systems. In: Proc. of the 20th International Conference on Hybrid Systems: Computation and Control. p. 173–178 (2017)
23. Bak, S., Tran, H.D., Johnson, T.T.: Numerical verification of affine systems with up to a billion dimensions. In: Proc. of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. p. 23–32 (2019)
24. Bartocci, E., Deshmukh, J.V., Donzé, A., Fainekos, G., Maler, O., Nickovic, D., Sankaranarayanan, S.: Specification-based monitoring of cyber-physical systems:

A survey on theory, tools and applications. In: Lectures on Runtime Verification, Lecture Notes in Computer Science, vol. 10457, pp. 135–175. Springer (2018). https://doi.org/10.1007/978-3-319-75632-5_5
25. Becchi, A., Zaffanella, E.: A direct encoding for NNC polyhedra. In: Computer Aided Verification. pp. 230–248. Springer (2018)
26. van Beek, D.A., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.: Concrete syntax and semantics of the compositional interchange format for hybrid systems. IFAC Proceedings Volumes **41**(2), 7979–7986 (2008)
27. Bemporad, A.: Efficient conversion of mixed logical dynamical systems into an equivalent piecewise affine form. IEEE transactions on Automatic Control **49**(5), 832–838 (2004)
28. Berz, M., Makino, K.: Performance of taylor model methods for validated integration of odes. In: Applied Parallel Computing. State of the Art in Scientific Computing. pp. 65–73. Springer (2006)
29. Beyer, D.: Competition on software verification and witness validation: SV-COMP 2023. In: TACAS (2). Lecture Notes in Computer Science, vol. 13994, pp. 495–522. Springer (2023)
30. Blom, H.A., Ma, H., Bakker, G.B.: Interacting particle system-based estimation of reach probability for a generalized stochastic hybrid system. IFAC-PapersOnLine **51**(16), 79–84 (2018)
31. Bogomolov, S., Forets, M., Frehse, G., Potomkin, K., Schilling, C.: JuliaReach: A toolbox for set-based reachability. In: Proc. of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. p. 39–44 (2019). https://doi.org/10.1145/3302504.3311804
32. Bogomolov, S., Forets, M., Frehse, G., Viry, F., Podelski, A., Schilling, C.: Reach set approximation through decomposition with low-dimensional sets and high-dimensional matrices. In: Proc. of the 21st International Conference on Hybrid Systems: Computation and Control. p. 41–50 (2018)
33. Bogomolov, S., Frehse, G., Giacobbe, M., Henzinger, T.A.: Counterexample-guided refinement of template polyhedra. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 589–606. Springer (2017)
34. Bohrer, R., Rahli, V., Vukotic, I., Völp, M., Platzer, A.: Formally verified differential dynamic logic. In: CPP. pp. 208–221. ACM (2017)
35. Bohrer, R., Tan, Y.K., Mitsch, S., Myreen, M.O., Platzer, A.: Veriphy: verified controller executables from verified cyber-physical system models. In: ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018. pp. 617–630 (2018). https://doi.org/10.1145/3192366.3192406
36. Bouissou, M., Houdebine, J.: Inconsistency detection in KB3 models. ESREL 2002 (2002)
37. Bouissou, M., Houdebine, J., S., H.: Reference manual of the Figaro probabilistic modelling language (2019)
38. Bouissou, M., Khan, S.: Bridging the dependability and model checking worlds. In: Congrès Lambda Mu 23 «Innovations et maîtrise des risques pour un avenir durable»-23e Congrès de Maîtrise des Risques et de Sûreté de Fonctionnement, Institut pour la Maîtrise des Risques (2022)
39. Bourke, T., Pouzet, M.: Zélus: A synchronous language with ODEs. In: International Conference on Hybrid Systems: Computation and Control (HSCC). pp. 113–118 (2013)

40. Bresolin, D., Collins, P., Geretti, L., Segala, R., Villa, T., Gonzalez, S.v.: A Computable and Compositional Semantics for Hybrid Automata. In: International Conference on Hybrid Systems: Computation and Control. HSCC, ACM (2020)
41. Bu, L., Abate, A., Adzkiya, D., Mufid, M.S., Ray, R., Wu, Y., Zaffanella, E.: ARCH-COMP20 category report: Hybrid systems with piecewise constant dynamics and bounded model checking. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20). EPiC Series in Computing, vol. 74, pp. 1–15. EasyChair (2020)
42. Bu, L., Frehse, G., Kundu, A., Ray, R., Shi, Y., Zaffanella, E.: ARCH-COMP22 category report: Hybrid systems with piecewise constant dynamics and bounded model checking. In: International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 90, pp. 44–57. EasyChair (2022)
43. Bu, L., Li, Y., Wang, L., Chen, X., Li, X.: BACH 2 : Bounded reachability checker for compositional linear hybrid systems. In: Design, Automation and Test in Europe (DATE). pp. 1512–1517 (2010)
44. Bu, L., Li, Y., Wang, L., Li, X.: BACH : Bounded reachability checker for linear hybrid automata. In: Formal Methods in Computer-Aided Design (FMCAD). pp. 1–4 (2008)
45. Bu, L., Ray, R., Schupp, S.: ARCH-COMP17 category report: Bounded model checking of hybrid systems with piecewise constant dynamics. In: ARCH17. International Workshop on Applied Verification of Continuous and Hybrid Systems, collocated with Cyber-Physical Systems Week (CPSWeek). EPiC Series in Computing, vol. 48, pp. 134–142. EasyChair (2017)
46. Bu, L., Ray, R., Schupp, S.: ARCH-COMP18 category report: Bounded model checking of hybrid systems with piecewise constant dynamics. In: ARCH18. International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 54, pp. 14–22. EasyChair (2018)
47. Bu, L., Ray, R., Schupp, S.: ARCH-COMP19 category report: Bounded model checking of hybrid systems with piecewise constant dynamics. In: ARCH19. International Workshop on Applied Verification of Continuous and Hybrid Systems, part of CPS-IoT Week. EPiC Series in Computing, vol. 61, pp. 120–128. EasyChair (2019)
48. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: An efficient statistical model checker for nondeterminism and rare events. International journal on software tools for technology transfer **22**(6), 759–780 (2020)
49. Cattaruzza, D., Abate, A., Schrammel, P., Kroening, D.: Unbounded-time analysis of guarded LTI systems with inputs by abstract acceleration. In: Static Analysis. p. 312–331 (2015)
50. Cauchi, N., Abate, A.: StocHy: automated verification and synthesis of stochastic processes. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2019)
51. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Proc. of Computer-Aided Verification. p. 258–263. LNCS 8044, Springer (2013)
52. Chen, X., Althoff, M., Immler, F.: Arch-comp17 category report: Continuous systems with nonlinear dynamics. In: International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 48, pp. 160–169. EasyChair (2017). https://doi.org/10.29007/v6g4

53. Chraibi, H., Houbedine, J., Sibler, A.: PyCATSHOO: Toward a new platform dedicated to dynamic reliability assessments of hybrid systems. In: 13th International Conference on Probabilistic Safety Assessment and Management (PSAM 13). Seoul, Korea (2016)
54. Chraibi, H., Houbedine, J., Sibler, A.: Pycatshoo: Toward a new platform dedicated to dynamic reliability assessments of hybrid systems. In: PSAM congress (2016)
55. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: HyComp: An SMT-Based Model Checker for Hybrid Systems. In: TACAS. pp. 52–67 (2015)
56. Corso, A., Moss, R.J., Koren, M., Lee, R., Kochenderfer, M.J.: A survey of algorithms for black-box safety validation of cyber-physical systems. J. Artif. Intell. Res. **72**, 377–428 (2021). https://doi.org/10.1613/jair.1.12716
57. Dang, T., Testylier, R.: Reachability analysis for polynomial dynamical systems using the bernstein expansion. Reliable Computing **17** (12 2012)
58. Delicaris, J., Schupp, S., Ábráham, E., Remke, A.: Maximizing Reachability Probabilities in Rectangular Automata with Random Clocks. In: Theoretical Aspects of Software Engineering (TASE). LNCS, Springer (2023), accepted for publication
59. Dokhanchi, A., Yaghoubi, S., Hoxha, B., Fainekos, G.: ARCH-COMP17 category report: Preliminary results on the falsification benchmarks. In: ARCH17. International Workshop on Applied Verification of Continuous and Hybrid Systems. pp. 170–174. EPiC Series in Computing, EasyChair (2017). https://doi.org/10.29007/wmf5
60. Dokhanchi, A., Yaghoubi, S., Hoxha, B., Fainekos, G., Ernst, G., Zhang, Z., Arcaini, P., Hasuo, I., Sedwards, S.: ARCH-COMP18 category report: Results on the falsification benchmarks. In: ARCH18. International Workshop on Applied Verification of Continuous and Hybrid Systems. pp. 104–109. EPiC Series in Computing, EasyChair (2018). https://doi.org/10.29007/t85q
61. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Proc. of Computer-Aided Verification. p. 167–170 (2010)
62. Duggirala, P.S., Mitra, S., Viswanathan, M., Potok, M.: C2E2: A verification tool for stateflow models. In: Tools and Algorithms for the Construction and Analysis of Systems. p. 68–82 (2015)
63. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: ACM International Conference on Hybrid Systems: Computation and Control, HSCC. pp. 157–168 (2019). https://doi.org/10.1145/3302504.3311807
64. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Learning and verification of feedback control systems using feedforward neural networks. IFAC-PapersOnLine **51**(16), 151 – 156 (2018). https://doi.org/10.1016/j.ifacol.2018.08.026, iFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018
65. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: NASA Formal Methods. pp. 121–138. Springer (2018)
66. Ernst, G., Arcaini, P., Bennani, I., Chandratre, A., Donzé, A., Fainekos, G., Frehse, G., Gaaloul, K., Inoue, J., Khandait, T., Mathesen, L., Menghi, C., Pedrielli, G., Pouzet, M., Waga, M., Yaghoubi, S., Yamagata, Y., Zhang, Z.: ARCH-COMP 2021 category report: Falsification with validation of results. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21). pp. 133–152. EPiC Series in Computing, EasyChair (2021). https://doi.org/10.29007/xwl1

67. Ernst, G., Arcaini, P., Bennani, I., Donzé, A., Fainekos, G., Frehse, G., Mathesen, L., Menghi, C., Pedrielli, G., Pouzet, M., Yaghoubi, S., Yamagata, Y., Zhang, Z.: ARCH-COMP 2020 category report: Falsification. In: ARCH20. International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20). pp. 140–152. EPiC Series in Computing, EasyChair (2020). https://doi.org/10.29007/trr1
68. Ernst, G., Arcaini, P., Donzé, A., Fainekos, G., Mathesen, L., Pedrielli, G., Yaghoubi, S., Yamagata, Y., Zhang, Z.: ARCH-COMP 2019 category report: Falsification. In: ARCH19. International Workshop on Applied Verification of Continuous and Hybrid Systems. pp. 129–140. EPiC Series in Computing, EasyChair (2019). https://doi.org/10.29007/68dk
69. Ernst, G., Arcaini, P., Fainekos, G., Formica, F., Inoue, J., Khandait, T., Mahboob, M.M., Menghi, C., Pedrielli, G., Waga, M., Yamagata, Y., Zhang, Z.: Arch-comp 2022 category report: Falsification with ubounded resources. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22). pp. 204–221. EPiC Series in Computing, EasyChair (2022). https://doi.org/10.29007/fhnk
70. Ernst, G., Sedwards, S., Zhang, Z., Hasuo, I.: Falsification of hybrid systems using adaptive probabilistic search. ACM Transactions on Modeling and Computer Simulation (TOMACS) **31**(3), 1–22 (2021)
71. Everdij, M., Blom, H.: Hybrid state Petri nets which have the analysis power of stochastic hybrid systems and the formal verification power of automata. In: Pawlewski, P. (ed.) Petri Nets, chap. 12, pp. 227–252. I-Tech Education and Publishing, Vienna (2010)
72. Fabian Immler, Matthias Althoff, M.S.T.e.a.: Symreach. https://github.com/mahendrasinghtomar/SymReach
73. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications. In: Formal Approaches to Software Testing and Runtime Verification. pp. 178–192. LNCS, Springer (2006)
74. Fan, C., Qi, B., Mitra, S., Viswanathan, M., Duggirala, P.S.: Automatic reachability analysis for nonlinear hybrid models with C2E2. In: Computer Aided Verification. p. 531–538 (2016)
75. Fan, J., Huang, C., Li, W., Chen, X., Zhu, Q.: Reachnn*: A tool for reachability analysis ofneural-network controlled systems. In: International Symposium on Automated Technology for Verification and Analysis (ATVA) (2020)
76. Fijalkow, N., Ouaknine, J., Pouly, A., Sousa-Pinto, J., Worrell, J.: On the decidability of reachability in linear time-invariant systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. pp. 77–86 (2019)
77. Formica, F., Fan, T., Rajhans, A., Pantelic, V., Lawford, M., Menghi, C.: Simulation-based testing of simulink models with test sequence and test assessment blocks. IEEE Transactions on Software Engineering pp. 1–19 (2023). https://doi.org/10.1109/TSE.2023.3343753
78. Formica, F., Tony, F., Menghi, C.: Search-based software testing driven by automatically generated and manually defined fitness functions. ACM Transactions on Software Engineering and Methodology (2023). https://doi.org/10.1145/3624745
79. Foster, S., Huerta y Munive, J., Gleirscher, M., Struth, G.: Hybrid systems verification with isabelle/hol: Simpler syntax, better models, faster proofs. In: Intl. Symp. on Formal Methods (FM 2021). vol. LNCS 13047, pp. 367–386 (2021)

80. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and safety verification for stochastic hybrid systems. In: International Conference on Hybrid Systems: Computation and Control. p. 43–52. HSCC '11, ACM (2011). https://doi.org/10.1145/1967701.1967710
81. Fränzle, M., Hansen, M.R.: A robust interpretation of duration calculus. In: International Colloquium on Theoretical Aspects of Computing. pp. 257–271. Springer (2005)
82. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. International Journal on Software Tools for Technology Transfer **10**, 263–279 (2008)
83. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Proc. of the 23rd International Conference on Computer Aided Verification. p. 379–395. LNCS 6806, Springer (2011)
84. Frehse, G., Abate, A., Adzkiya, D., Becchi, A., Bu, L., Cimatti, A., Giacobbe, M., Griggio, A., Mover, S., Mufid, M.S., Riouak, I., Tonetta, S., Zaffanella, E.: ARCH-COMP19 category report: Hybrid systems with piecewise constant dynamics. In: ARCH19. International Workshop on Applied Verification of Continuous and Hybrid Systemsi, part of CPS-IoT Week. EPiC Series in Computing, vol. 61, pp. 1–13. EasyChair (2019)
85. Frehse, G., Abate, A., Adzkiya, D., Bu, L., Giacobbe, M.: ARCH-COMP17 category report: Hybrid systems with piecewise constant dynamics. In: ARCH17. International Workshop on Applied Verification of Continuous and Hybrid Systems, collocated with Cyber-Physical Systems Week (CPSWeek). EPiC Series in Computing, vol. 48, pp. 124–133. EasyChair (2017)
86. Frehse, G., Abate, A., Adzkiya, D., Bu, L., Giacobbe, M., Mufid, M.S., Zaffanella, E.: ARCH-COMP18 category report: Hybrid systems with piecewise constant dynamics. In: ARCH18. International Workshop on Applied Verification of Continuous and Hybrid Systems, ARCH@ADHS. EPiC Series in Computing, vol. 54, pp. 1–13. EasyChair (2018)
87. Frehse, G.F.: Compositional verification of hybrid systems using simulation relations. Ph.D. thesis, Radboud University (2005)
88. Fulton, N., Mitsch, S., Bohrer, B., Platzer, A.: Bellerophon: Tactical theorem proving for hybrid systems. In: ITP. pp. 207–224 (2017). https://doi.org/10.1007/978-3-319-66107-0_14
89. Fulton, N., Mitsch, S., Quesel, J., Völp, M., Platzer, A.: Keymaera X: an axiomatic tactical theorem prover for hybrid systems. In: International Conference on Automated Deduction (CADE). pp. 527–538 (2015). https://doi.org/10.1007/978-3-319-21401-6_36
90. Fulton, N., Platzer, A.: Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In: Conference on Artificial Intelligence, (AAAI). pp. 6485–6492 (2018)
91. Gallicchio, J., Tan, Y.K., Mitsch, S., Platzer, A.: Implicit definitions with differential equations for keymaera X - (system description). In: Automated Reasoning - International Joint Conference (IJCAR). pp. 723–733 (2022). https://doi.org/10.1007/978-3-031-10769-6_42
92. Garcia, L., Mitsch, S., Platzer, A.: Hyplc: hybrid programmable logic controller program translation for verification. In: ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS. pp. 47–56 (2019). https://doi.org/10.1145/3302509.3311036

93. Geretti, L., Collins, P., Bresolin, D., Villa, T.: Automating numerical parameters along the evolution of a nonlinear system. Lecture Notes in Computer Science **13498 LNCS**, 336 – 345 (2022). https://doi.org/10.1007/978-3-031-17196-3_22
94. Geretti, L., Sandretto, J.A.D., Althoff, M., Benet, L., Chapoutot, A., Chen, X., Collins, P., Forets, M., Freire, D., Immler, F., Kochdumper, N., Sanders, D.P., Schilling, C.: Arch-comp20 category report: Continuous and hybrid systems with nonlinear dynamics. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20). EPiC Series in Computing, vol. 74, pp. 49–75. EasyChair (2020). https://doi.org/10.29007/zkf6, https://easychair.org/publications/paper/nrdD
95. Geretti, L., Sandretto, J.A.D., Althoff, M., Benet, L., Chapoutot, A., Collins, P., Duggirala, P.S., Forets, M., Kim, E., Linares, U., Sanders, D.P., Schilling, C., Wetzlinger, M.: ARCH-COMP21 category report: Continuous and hybrid systems with nonlinear dynamics. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21). EPiC Series in Computing, vol. 80, pp. 32–54. EasyChair (2021). https://doi.org/10.29007/2jw8, https://easychair.org/publications/paper/GWwz
96. Geretti, L., Sandretto, J.A.D., Althoff, M., Benet, L., Collins, P., Duggirala, P., Forets, M., Kim, E., Mitsch, S., Schilling, C., Wetzlinger, M.: ARCH-COMP22 category report: Continuous and hybrid systems with nonlinear dynamics. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22). EPiC Series in Computing, vol. 90, pp. 86–112. EasyChair (2022). https://doi.org/10.29007/fnzc, https://easychair.org/publications/paper/JrQ4
97. Geretti, L., Sandretto, J.A.D., Althoff, M., Benet, L., Collins, P., Forets, M., Ivanova, E., Li, Y., Mitra, S., Mitsch, S., Schilling, C., Wetzlinger, M., Zhuang, D.: Arch-comp23 category report: Continuous and hybrid systems with nonlinear dynamics. In: Frehse, G., Althoff, M. (eds.) Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23). EPiC Series in Computing, vol. 96, pp. 61–88. EasyChair (2023). https://doi.org/10.29007/93f2, https://easychair.org/publications/paper/T7LG
98. Girard, A., Le Guernic, C.: Zonotope/hyperplane intersection for hybrid systems reachability analysis. In: Proc. of Hybrid Systems: Computation and Control. p. 215–228. LNCS 4981, Springer (2008)
99. Girard, A., Le Guernic, C., Maler, O.: Efficient computation of reachable sets of linear time-invariant systems with inputs. In: Hybrid Systems: Computation and Control. p. 257–271. LNCS 3927, Springer (2006)
100. Guelev, D.P., Wang, S., Zhan, N.: Compositional hoare-style reasoning about hybrid CSP in the duration calculus. In: SETTA. Lecture Notes in Computer Science, vol. 10606, pp. 110–127. Springer (2017)
101. Haesaert, S., Soudjani, S.: Robust dynamic programming for temporal logic control of stochastic systems. IEEE Transactions on Automatic Control **66**(6), 2496–2511 (2020)
102. Haesaert, S., Zadeh Soudjani, S.E., Abate, A.: Verification of general Markov decision processes by approximate similarity relations and policy refinement. SIAM Journal on Control and Optimization **55**(4), 2333–2367 (2017)
103. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 8413, pp. 593–598. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_51

104. Henzinger, T.: The theory of hybrid automata. In: Inan, K., Kurshan, R.P. (eds.) Verification of Digital and Hybrid Systems, NATO ASI Series F: Computer and Systems Sciences, vol. 170, p. 265–292. Springer (2000)
105. Huang, C., Fan, J., Chen, X., Li, W., Zhu, Q.: POLAR: A polynomial arithmetic framework for verifying neural-network controlled systems. In: International Symposium on Automated Technology for Verification and Analysis (ATVA) (2022)
106. Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: Reachnn: Reachability analysis of neural-network controlled systems. ACM Transactions on Embedded Computing Systems (TECS) **18**(5s), 1–22 (2019)
107. Huerta y Munive, J.J., Struth, G.: Predicate transformer semantics for hybrid systems. JAR **66**(1), 93–139 (2022)
108. Hüls, J., Niehaus, H., Remke, A.: Hpnmg: A C++ Tool for Model Checking Hybrid Petri Nets with General Transitions. In: International NASA Formal Methods Symposium (NFM). Springer (2020)
109. Immler, F.: Verified reachability analysis of continuous systems. In: TACAS'15. LNCS, vol. 9035, pp. 37–51. Springer (2015)
110. Immler, F., Althoff, M., Benet, L., Chapoutot, A., Chen, X., Forets, M., Geretti, L., Kochdumper, N., Sanders, D.P., Schilling, C.: ARCH-COMP19 category report: Continuous and hybrid systems with nonlinear dynamics. In: ARCH19. International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 61, pp. 41–61. EasyChair (2019). https://doi.org/10.29007/m75b, https://easychair.org/publications/paper/4FSh
111. Immler, F., Althoff, M., Chen, X., Fan, C., Frehse, G., Kochdumper, N., Li, Y., Mitra, S., Tomar, M.S., Zamani, M.: ARCH-COMP18 category report: Continuous and hybrid systems with nonlinear dynamics. In: Frehse, G. (ed.) ARCH18. International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 54, pp. 53–70. EasyChair (2018). https://doi.org/10.29007/mskf, https://easychair.org/publications/paper/gjfh
112. Ivanov, R., Carpenter, T., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In: International Conference on Computer-Aided Verification (2021)
113. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: Verifying safety properties of hybrid systems with neural network controllers. In: International Conference on Hybrid Systems: Computation and Control. p. 169–178. HSCC, ACM (2019). https://doi.org/10.1145/3302504.3311806
114. Jagtap, P., Soudjani, S., Zamani, M.: Temporal logic verification of stochastic systems using barrier certificates. In: International Symposium on Automated Technology for Verification and Analysis. pp. 177–193. Springer (2018)
115. Jin, X., Deshmukh, J.V., Kapinski, J., Ueda, K., Butts, K.: Powertrain control verification benchmark. In: International Conference on Hybrid Systems: Computation and Control. pp. 253–262. ACM (2014)
116. Johnson, T.T.: Arch-comp17 repeatability evaluation report. In: ARCH17. International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 48, pp. 175–180. EasyChair (2017). https://doi.org/10.29007/7hvk, https://easychair.org/publications/paper/nMvb
117. Johnson, T.T.: Arch-comp18 repeatability evaluation report. In: ARCH18. International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 54, pp. 128–134. EasyChair (2018). https://doi.org/10.29007/n9t3, https://easychair.org/publications/paper/9J6v

118. Johnson, T.T.: Arch-comp19 repeatability evaluation report. In: ARCH19. International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 61, pp. 162–169. EasyChair (2019). https://doi.org/10.29007/wbl3, https://easychair.org/publications/paper/xvBM
119. Johnson, T.T.: Arch-comp20 repeatability evaluation report. In: ARCH20. International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20). EPiC Series in Computing, vol. 74, pp. 175–183. EasyChair (2020). https://doi.org/10.29007/8dp4, https://easychair.org/publications/paper/3W11
120. Johnson, T.T.: ARCH-COMP21 repeatability evaluation report. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21). EPiC Series in Computing, vol. 80, pp. 153–160. EasyChair (2021). https://doi.org/10.29007/zqdx, https://easychair.org/publications/paper/cfpN
121. Johnson, T.T.: Arch-comp22 repeatability evaluation report. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22). EPiC Series in Computing, vol. 90, pp. 222–230. EasyChair (2022). https://doi.org/10.29007/djqx, https://easychair.org/publications/paper/LnDH
122. Johnson, T.T.: Arch-comp23 repeatability evaluation report. In: Frehse, G., Althoff, M. (eds.) Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23). EPiC Series in Computing, vol. 96, pp. 189–195. EasyChair (2023). https://doi.org/10.29007/q313, https://easychair.org/publications/paper/TdVx
123. Johnson, T.T., Lopez, D.M., Benet, L., Forets, M., Guadalupe, S., Schilling, C., Ivanov, R., Carpenter, T.J., Weimer, J., Lee, I.: Arch-comp21 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21). EPiC Series in Computing, vol. 80, pp. 90–119. EasyChair (2021). https://doi.org/10.29007/kfk9
124. Johnson, T.T., Lopez, D.M., Musau, P., Tran, H.D., Botoeva, E., Leofante, F., Maleki, A., Sidrane, C., Fan, J., Huang, C.: Arch-comp20 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In: ARCH20. International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20). EPiC Series in Computing, vol. 74, pp. 107–139. EasyChair (2020). https://doi.org/10.29007/9xgv
125. Kim, E., Duggirala, P.S.: Kaa: A python implementation of reachable set computation using bernstein polynomials. EPiC Series in Computing **74**, 184–196 (2020)
126. Kochdumper, N., Schilling, C., Althoff, M., Bak, S.: Open- and closed-loop neural network verification using polynomial zonotopes. In: NASA Formal Methods. pp. 16–36. Springer (2023)
127. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2**(4), 255–299 (1990)
128. Lafferriere, G., Pappas, G.J., Yovine, S.: Symbolic reachability computation for families of linear vector fields. Symbolic Computation **32**, 231–253 (2001)
129. Lavaei, A., Khaled, M., Soudjani, S., Zamani, M.: AMYTISS: Parallelized automated controller synthesis for large-scale stochastic systems. In: Computer Aided Verification (CAV). pp. 461–474. Springer (2020)
130. Lavaei, A., Soudjani, S., Abate, A., Zamani, M.: Automated verification and synthesis of stochastic hybrid systems: A survey. Automatica **146**, 110617 (2022)
131. Leahy, K., Cristofalo, E., Vasile, C.I., Jones, A., Montijano, E., Schwager, M., Belta, C.: Control in belief space with temporal logic specifications using vision-

based localization. The International Journal of Robotics Research **38**(6), 702–722 (2019)
132. Li, Y., Zhu, H., Braught, K., Shen, K., Mitra, S.: Verse: A python library for reasoning about multi-agent hybrid system scenarios. In: Computer Aided Verification. pp. 351–364 (2023)
133. Loos, S.M., Platzer, A.: Differential refinement logic. In: Annual ACM/IEEE Symposium on Logic in Computer Science, LICS. pp. 505–514 (2016). https://doi.org/10.1145/2933575.2934555
134. Lopez, D.M., Althoff, M., Benet, L., Chen, X., Fan, J., Forets, M., Huang, C., Johnson, T.T., Ladner, T., Li, W., Schilling, C., Zhu, Q.: Arch-comp22 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22). EPiC Series in Computing, vol. 90, pp. 142–184. EasyChair (2022). https://doi.org/10.29007/wfgr
135. Lopez, D.M., Althoff, M., Forets, M., Johnson, T.T., Ladner, T., Schilling, C.: Arch-comp23 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In: Frehse, G., Althoff, M. (eds.) Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23). EPiC Series in Computing, vol. 96, pp. 89–125. EasyChair (2023). https://doi.org/10.29007/x38n, https://easychair.org/publications/paper/Vfq4b
136. Lopez, D.M., Choi, S.W., Tran, H.D., Johnson, T.T.: Nnv 2.0: The neural network verification tool. In: Computer Aided Verification (CAV). p. 397–412. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-37703-7_19
137. Lopez, D.M., Musau, P., Tran, H.D., Dutta, S., Carpenter, T.J., Ivanov, R., Johnson, T.T.: Arch-comp19 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In: ARCH19. International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 61, pp. 103–119. EasyChair (2019). https://doi.org/10.29007/rgv8
138. Ma, H., Blom, H.A.: Interacting particle system based estimation of reach probability of general stochastic hybrid systems. Nonlinear Analysis: Hybrid Systems **47**, 101303 (2023)
139. Majumdar, R., Mallik, K., Rychlicki, M., Schmuck, A.K., Soudjani, S.: A flexible toolchain for symbolic rabin games under fair and stochastic uncertainties. In: Computer Aided Verification (CAV). Springer (2023), (to appear)
140. Majumdar, R., Mallik, K., Soudjani, S.: Symbolic controller synthesis for büchi specifications on stochastic systems. In: International conference on hybrid systems: computation and control. pp. 1–11 (2020)
141. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. pp. 152–166. Springer (2004)
142. Menghi, C., Nejati, S., Briand, L., Isasi Parache, Y.: Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification. In: International Conference on Software Engineering (ICSE). IEEE / ACM (2020)
143. Mitsch, S.: Implicit and explicit proof management in keymaera X. In: Proceedings of the 6th Workshop on Formal Integrated Development Environment, F-IDE@NFM 2021, Held online, 24-25th May 2021. pp. 53–67 (2021). https://doi.org/10.4204/EPTCS.338.8, https://doi.org/10.4204/EPTCS.338.8

144. Mitsch, S., Jin, X., Zhan, B., Wang, S., Zhan, N.: Arch-comp21 category report: Hybrid systems theorem proving. In: Frehse, G., Althoff, M. (eds.) 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21). EPiC Series in Computing, vol. 80, pp. 120–132. EasyChair (2021). https://doi.org/10.29007/35cf
145. Mitsch, S., y Munive, J.J.H., Jin, X., Zhan, B., Wang, S., Zhan, N.: ARCH-COMP20 category report: Hybrid systems theorem proving. In: ARCH. EPiC Series in Computing, vol. 74, pp. 153–174. EasyChair (2020)
146. Mitsch, S., Platzer, A.: Modelplex: verified runtime validation of verified cyber-physical system models. Formal Methods Syst. Des. **49**(1-2), 33–74 (2016). https://doi.org/10.1007/s10703-016-0241-z
147. Mitsch, S., Sheng, H., Zhan, B., Wang, S., Foster, S., Munive, J.J.H.Y.: Arch-comp23 category report: Hybrid systems theorem proving. In: Frehse, G., Althoff, M. (eds.) Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23). EPiC Series in Computing, vol. 96, pp. 170–188. EasyChair (2023). https://doi.org/10.29007/57g4
148. Mitsch, S., Zhan, B., Sheng, H., Bentkamp, A., Jin, X., Wang, S., Foster, S., Laursen, C.P., Munive, J.J.H.Y.: Arch-comp22 category report: Hybrid systems theorem proving. In: Frehse, G., Althoff, M., Schoitsch, E., Guiochet, J. (eds.) Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22). EPiC Series in Computing, vol. 90, pp. 185–203. EasyChair (2022). https://doi.org/10.29007/4lxf
149. Mufid, M.S., Adzkiya, D., Abate, A.: Symbolic reachability analysis of high dimensional max-plus linear systems. IFAC-PapersOnLine **53**(4), 459–465 (2020). https://doi.org/https://doi.org/10.1016/j.ifacol.2021.04.060
150. y Munive, J.J.H.: Verification components for hybrid systems. Arch. Formal Proofs **2019** (2019), https://www.isa-afp.org/entries/Hybrid_Systems_VCs.html
151. NNFal. https://gitlab.com/Atanukundu/NNFal (04 2023 [Online])
152. Peltomäki, J., Porres, I.: Requirement falsification for cyber-physical systems using generative models. arXiv preprint arXiv:2310.20493 (2023)
153. Pilch, C., Remke, A.: HYPEG: Statistical Model Checking for hybrid Petri nets: Tool Paper. In: International Conference on Performance Evaluation Methodologies and Tools. pp. 186–191. VALUETOOLS 2017, ACM (2017)
154. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reason. **59**(2), 219–265 (2017)
155. Platzer, A., Quesel, J.: Keymaera: A hybrid theorem prover for hybrid systems (system description). In: Automated Reasoning, 4th International Joint Conference (IJCAR). pp. 171–178 (2008). https://doi.org/10.1007/978-3-540-71070-7_15
156. Qian, M., Mitsch, S.: Reward shaping from hybrid systems models in reinforcement learning. In: NASA Formal Methods - International Symposium (NFM). pp. 122–139 (2023). https://doi.org/10.1007/978-3-031-33170-1_8
157. Quesel, J., Mitsch, S., Loos, S.M., Aréchiga, N., Platzer, A.: How to model and prove hybrid systems with keymaera: a tutorial on safety. Int. J. Softw. Tools Technol. Transf. **18**(1), 67–91 (2016)
158. Ray, R., Gurung, A., Das, B., Bartocci, E., Bogomolov, S., Grosu, R.: XSpeed: Accelerating reachability analysis on multi-core processors. In: Hardware and Software: Verification and Testing. p. 3–18. Springer International Publishing (2015)
159. Salamati, M., Soudjani, S., Majumdar, R.: Approximate time bounded reachability for ctmcs and ctmdps: A lyapunov approach. In: QEST. Lecture Notes in Computer Science, vol. 11024, pp. 389–406. Springer (2018)

160. Alexandre dit Sandretto, J., Chapoutot, A.: Validated Explicit and Implicit Runge-Kutta Methods. Reliable Computing electronic edition **22** (2016)
161. Schupp, S., Abraham, E., Ben Makhlouf, I., Kowalewski, S.: HyPro: A C++ library for state set representations for hybrid systems reachability analysis. In: Proc. of the NASA Formal Methods Symposium. p. 288–294 (2017)
162. Sheng, H., Bentkamp, A., Zhan, B.: HHLPy: Practical verification of hybrid systems using hoare logic. In: Formal Methods. pp. 160–178. Springer (2023)
163. Shmarov, F., Zuliani, P.: ProbReach: Verified probabilistic δ-reachability for stochastic hybrid systems. In: HSCC. pp. 134–139. ACM (2015)
164. Sidrane, C., Kochenderfer, M.J.: OVERT: Verification of nonlinear dynamical systems with neural network controllers via overapproximation. Safe Machine Learning workshop at ICLR (2019)
165. Sogokon, A., Mitsch, S., Tan, Y.K., Cordwell, K., Platzer, A.: Pegasus: sound continuous invariant generation. Formal Methods Syst. Des. **58**(1-2), 5–41 (2021). https://doi.org/10.1007/s10703-020-00355-z
166. Soudjani, S., Gevaerts, C., Abate, A.: FAUST^2: Formal Abstractions of Uncountable-STate STochastic processes. In: TACAS. vol. 15, pp. 272–286 (2015)
167. Strauss, M., Mitsch, S.: Slow down, move over: A case study in formal verification, refinement, and testing of the responsibility-sensitive safety model for self-driving cars. In: Tests and Proofs - International Conference (TAP). pp. 149–167 (2023). https://doi.org/10.1007/978-3-031-38828-6_9
168. Tan, Y.K., Mitsch, S., Platzer, A.: Verifying switched system stability with logic. In: ACM International Conference on Hybrid Systems: Computation and Control (HSCC). pp. 2:1–2:11 (2022). https://doi.org/10.1145/3501710.3519541
169. Thibeault, Q., Anderson, J., Chandratre, A., Pedrielli, G., Fainekos, G.: PSY-TaLiRo: A Python Toolbox for Search-Based Test Generation for Cyber-Physical Systems. In: Formal Methods for Industrial Critical Systems. pp. 223–231. Springer (2021)
170. Tran, H.D., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: International Conference on Computer-Aided Verification (CAV) (July 2020). https://doi.org/10.1007/978-3-030-53288-8_1
171. Van Huijgevoort, B., Schön, O., Soudjani, S., Haesaert, S.: Syscore: Synthesis via stochastic coupling relations. In: International Conference on Hybrid Systems: Computation and Control. HSCC '23, ACM (2023). https://doi.org/10.1145/3575870.3587123
172. Vinod, A.P., Gleason, J.D., Oishi, M.M.: Sreachtools: a MATLAB stochastic reachability toolbox. In: ACM international conference on hybrid systems: computation and control. pp. 33–38 (2019)
173. Waga, M.: Falsification of cyber-physical systems with robustness-guided black-box checking. In: International Conference on Hybrid Systems: Computation and Control (HSCC). pp. 11:1–11:13. ACM (2020). https://doi.org/10.1145/3365365.3382193
174. Wang, S., Zhan, N., Zou, L.: An improved HHL prover: an interactive theorem prover for hybrid systems. In: ICFEM 2015, LNCS, vol. 9407, pp. 382–399. Springer (2015)
175. Wetzlinger, M., Kochdumper, N., Althoff, M.: Adaptive parameter tuning for reachability analysis of linear systems. In: IEEE Conference on Decision and Control. pp. 5145–5152 (2020). https://doi.org/10.1109/CDC42340.2020.9304431

176. Wetzlinger, M., Kochdumper, N., Bak, S., Althoff, M.: Fully automated verification of linear systems using inner and outer approximations of reachable sets. IEEE Transactions on Automatic Control **68**(12), 7771–7786 (2023). https://doi.org/10.1109/TAC.2023.3292008
177. Wetzlinger, M., Kochdumper, N., Bak, S., Althoff, M.: Fully-automated verification of linear systems using reachability analysis with support functions. In: Proc. of the 26th ACM International Conference on Hybrid Systems: Computation and Control (2023). https://doi.org/10.1145/3575870.3587121
178. Wetzlinger, M., Kulmburg, A., Althoff, M.: Adaptive parameter tuning for reachability analysis of nonlinear systems. In: International Conference on Hybrid Systems: Computation and Control. HSCC '21, Association for Computing Machinery (2021). https://doi.org/10.1145/3447928.3456643, https://doi.org/10.1145/3447928.3456643
179. Wetzlinger, M., Kulmburg, A., Le Penven, A., Althoff, M.: Adaptive reachability algorithms for nonlinear systems using abstraction error analysis. Nonlinear Analysis: Hybrid Systems **46** (2022). https://doi.org/10.1016/j.nahs.2022.101252
180. Winter, S., Timperley, C.S., Hermann, B., Cito, J., Bell, J., Hilton, M., Beyer, D.: A retrospective study of one decade of artifact evaluations. In: ESEC/FSE 2022: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 145–156. ACM (2022). https://doi.org/10.1145/3540250.3549172
181. Yamagata, Y., Liu, S., Akazaki, T., Duan, Y., Hao, J.: Falsification of cyber-physical systems using deep reinforcement learning. IEEE Transactions on Software Engineering **47**(12), 2823–2840 (2021). https://doi.org/10.1109/TSE.2020.2969178
182. Zhang, Z., Lyu, D., Arcaini, P., Ma, L., Hasuo, I., Zhao, J.: Effective Hybrid System Falsification Using Monte Carlo Tree Search Guided by QB-Robustness. In: Computer Aided Verification. pp. 595–618. Springer (2021). https://doi.org/10.1007/978-3-030-81685-8_29
183. zlscheck. https://github.com/ismailbennani/zlscheck (04 2023 [Online])