EPiC Series in Computing

Volume 61, 2019, Pages 1–13

ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems

ARCH-COMP19 Category Report: Hybrid Systems with Piecewise Constant Dynamics

Goran Frehse[1], Alessandro Abate[2], Dieky Adzkiya[3], Anna Becchi[4], Lei Bu[5], Alessandro Cimatti[9], Mirco Giacobbe[6], Alberto Griggio[9], Sergio Mover[7], Muhammad Syifa'ul Mufid[2], Idriss Riouak[4], Stefano Tonetta[9], and Enea Zaffanella[8]

[1] ENSTA ParisTech, Palaiseau, France
goran.frehse@ensta-paristech.fr
[2] University of Oxford, Oxford, U.K.
{aabate,muhammad.syifaul.mufid}@cs.ox.ac.uk
[3] Department of Mathematics, Institut Teknologi Sepuluh Nopember, Indonesia
dieky@matematika.its.ac.id
[4] Department of Mathematics, Computer Science, and Physics University of Udine, Italy
{becchi.anna,riouak.idriss}@spes.uniud.it
[5] State Key Laboratory of Novel Software Techniques, Nanjing University, Nanjing, Jiangsu, P.R. China
bulei@nju.edu.cn
[6] IST Austria, Klosterneuburg, Austria
mgiacobbe@ist.ac.at
[7] LIX, Ecole Polytechnique, Palaiseau, France
sergio.mover@lix.polytechnique.fr
[8] Department of Mathematical, Physical and Computer Sciences University of Parma, Italy
enea.zaffanella@unipr.it
[9] Fondazione Bruno Kessler, Italy
{cimatti,griggio,tonettas}@fbk.eu

Abstract

This report presents the results of a friendly competition for formal verification of continuous and hybrid systems with piecewise constant dynamics. The friendly competition took place as part of the workshop Applied Verification for Continuous and Hybrid Systems (ARCH) in 2019. In this third edition, six tools have been applied to solve five different benchmark problems in the category for piecewise constant dynamics: BACH, Lyse, HyCOMP, PHAVer/SX, PHAVerLite, and VeriSiMPL. Compared to last year, a new tool has participated (HyCOMP) and PHAVerLite has replaced PHAVer-lite. The result is a snapshot of the current landscape of tools and the types of benchmarks they are particularly suited for. Due to the diversity of problems, we are not ranking tools, yet the presented results probably provide the most complete assessment of tools for the safety verification of continuous and hybrid systems with piecewise constant dynamics up to this date.

1 Introduction

Disclaimer The presented report of the ARCH friendly competition for *continuous and hybrid systems with piecewise constant dynamics* aims at providing a landscape of the current capabilities of verification tools. We would like to stress that each tool has unique strengths—not all of the specificities can be highlighted within a single report. To reach a consensus in what benchmarks are used, some compromises had to be made so that some tools may benefit more from the presented choice than others. The obtained results have been verified by an independent repeatability evaluation. To establish further trustworthiness of the results, the code with which the results have been obtained is publicly available at gitlab.com/goranf/ARCH-COMP.

This report summarizes results obtained in the 2019 friendly competition of the ARCH workshop[1] for verifying hybrid systems with piecewise constant bounds on the dynamics. 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 $\mathbb{R}^n$. Tool developers run their tools summarized in Sec. 2 on different benchmark problems presented in Sec. 3 and report the results obtained from their own machines also in Sec. 3.

The results reported by each participant have not been checked by an independent authority and are obtained on the machines of the tool developers. Thus, one has to factor in the computational power of the used processors summarized in Sec. A as well as the efficiency of the programming language of the tools. It is not the goal of the friendly competition to rank the results, the goal is to present the landscape of existing solutions in a breadth that is not possible by scientific publications in classical venues. Those would require the presentation of novel techniques, while this report showcases the current state of the art.

The selection of the benchmarks has been conducted within the forum of the ARCH website (cps-vo.org/group/ARCH), which is visible for registered users and registration is open for anybody. All tools presented in this report use some form of reachability analysis. This, however, is not a constraint set by the organizers of the friendly competition. We hope to encourage further tool developers to showcase their results in future editions.

2 Participating Tools

The tools participating in the category *Continuous and Hybrid Systems with Piecewise Constant Dynamics* are introduced below in alphabetical order.

BACH BACH [14, 13] is a bounded reachability checker for Linear Hybrid Automata (LHA) model, Hybrid Systems with Piecewise Constant Dynamics (HPWC) in the term of ARCH competition. The tool provides GUI for LHA modeling and also bounded reachability checkers for both single automaton and automata network. Be different from classical bounded checkers of LHA, which encodes the "complete" bounded state space of the system into a huge SMT

[1] Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH), cps-vo.org/group/ARCH

problem, BACH conducts the bounded checking in a "path-oriented" layered style. It finds potential paths which can reach the target location on the graph structure first, then encodes the feasibility of such path into a linear programming problem and solve it afterwards. In this way, as the number of paths in the discrete graph structure of an LHA under a given bound is finite, all candidate paths can be enumerated and checked one by one to tackle the bounded reachability analysis of LHA. Furthermore, the memory usage is well controlled as it only encodes and solves one path at a time. Meanwhile, BACH provides an efficient way to locate the infeasible path segment core when a path is reported as infeasible to guide the backtracking in the graph structure traversing to achieve good performance [28]. Such infeasible path segments can also be used to derive complete state arguments under certain conditions [29].

Lyse Lyse is a tool for the reachability analysis of convex hybrid automata, namely hybrid automata with piecewise constant dynamics, whose constraints are possibly non-linear but required to be convex. In this class are HPWC whose flow is contrained in rectangles, polyhedra, but also ellipses and parabolae. Linear hybrid automata are a special case. Lyse performs forward reachability analysis by means of template-polyhedra, whose directions are incrementally extracted from spurious counterexamples. The extraction is performed by a novel technique that generates interpolants by means of convex programming [9].

HyCOMP HyCOMP [16] is a model checker for Hybrid Systems based on Satisfiability Modulo Theory (SMT) [6]. HyCOMP takes as input a hybrid system with either piecewise constant dynamics or linear dynamics represented as a symbolic network of hybrid automata.

HyCOMP verifies safety properties for piecewise constant hybrid systems in two steps. First, it encodes the hybrid system in an infinite-state discrete transition system expressed with first order logic formulas (the formulas are usually interpreted in the theory of Linear Real Arithmetic). The encoding is precise for safety properties: the property holds in the hybrid system if and only if it holds in the discrete transition system. Second, it verifies the property in the discrete transition system. The tool can use different algorithms like Bounded Model Checking (BMC), K-Induction, and IC3 [12]. Such algorithms are implemented in the nuXmv model checker [15] using the MathSAT SMT solver [19]. HyCOMP directly uses nuXmv as a library. In the experiments HyCOMP uses the IC3-IA [17, 18] algorithm. IC3-IA tightly integrates (implicit) predicate abstraction [27] with the IC3 algorithm [12]. IC3-IA performs a Counter Example Guided Abstraction Refinement (CEGAR) loop using predicate abstraction to abstract the system, IC3 to verify the abstraction, BMC to simulate abstract counterexamples, and interpolation to find new predicates to refine the abstraction. However, IC3-IA never computes the whole predicate abstraction explicitly, trying to avoid its exponential blow up in the number of states. In the experiments we run both IC3-IA and BMC-IA (a version of BMC also using implicit abstraction, as shown in [27]). We refer to the former configuration as HyCOMP-IC3, and to the latter configuration as HyCOMP-BMC. We further run BMC only on the unsafe instances of the benchmarks.

PHAVer/SX PHAVer [21] is a formal verification tool for computing reachability and equivalence (simulation relation) of hybrid systems. It can handle the class of Linear Hybrid Automata (LHA), whose continuous dynamics is characterized by piecewise constant bounds on the derivatives and whose discrete jumps can be a convex linear predicate over the variables before and after the jump. PHAVer uses standard operations on polyhedra for the reachability computation over an infinite time horizon (similar to those used in HyTech), and the algorithm for computing simulation relations is a straightforward extension of these. Using unbounded

integer arithmetic, the computations are exact and formally sound. While termination of LHA is undecidable, PHAVer provides formally sound, precise overapproximation and widening operators that can force termination at the cost of reduced precision. These operators also simplify the computed continuous sets and dynamics of the system, and may result in a considerable speed-up without much loss in precision. Since 2011, PHAVer is continued as a plugin to the tool platform SpaceEx. This plugin is the tool actually used for the competition: for clarity, in the following we refer to it under the name PHAVer/SX.

PHAVerLite PHAVerLite is a variant of the stand-alone verification tool PHAVer, sharing the same capabilities and formal soundness guarantees. It is worth stressing that PHAVerLite, being a *stand-alone* tool, differs from the PHAVer-lite *SpaceEx plugin* that participated in the friendly competition in 2018. For instance, while PHAVer-lite was able to accept input specified using the SpaceEx syntax for hybrid automata, at present PHAVerLite can only accept input specified using the PHAVer syntax. The main difference with respect to PHAVer is the adoption of the new polyhedra library PPLite [8]: thanks to a novel representation and conversion algorithm [7] for NNC (Not Necessarily Closed) polyhedra, PPLite is able to obtain significant efficiency improvements with respect to the classical polyhedra implementation used in PHAVer (which is based on the Parma Polyhedra Library [5]). The development of PHAVerLite was motivated by the desire to go beyond the main change above and also revisit many of the key design and implementation choices of the original PHAVer: this allowed to experiment with novel algorithms or design tradeoffs, also exploiting some of the more recent advances in the implementation of operators on the polyhedral domains. At present, PHAVerLite has only been used to analyze systems characterized by piecewise constant dynamics; also note that a few of the PHAVer functionalities (e.g., the computation of simulation relations) have been deliberately removed.

VeriSiMPL This toolbox [1, 4] is used to generate finite abstractions and reachability of max-plus-linear (MPL) systems. VeriSiMPL leverages the piecewise affine (PWA) dynamics generated from an MPL system and some operations over difference-bound matrices (DBM) [20]. Abstractions are characterized as finite-state labeled transition systems (LTS). The finite LTS abstractions are shown to either simulate or to bisimulate the original MPL system [2]. The resulting LTS are to be verified against given specifications expressed as formulae in linear temporal logic (LTL) and computation tree logic (CTL). The toolbox intends to leverage the SPIN and NuSMV model checkers. With regards to the reachability of MPL systems, VeriSiMPL is able to compute the forward and backward reach sets of MPL systems exactly [3]. The initial and final states are expressed as a union of finitely many DBM. The reachability algorithm uses the PWA dynamics associated with an MPL system and some operations on DBM.

3 Verification of Benchmarks

3.1 Adaptive Cruise Controller

Model The adaptive cruise controller is a distributed system for assuring that safety distances in a platoon of cars are satisfied [10]. It is inspired by a related benchmark in [24]. For n cars, the number of discrete states is 2^n and the number of continuous variables is n. Each variable x_i encodes the relative position of the i-th car, for $i = 0, \ldots, n-1$. The car i-th car is considered

to be in front of the $i+1$-th car. The relative velocity of each car has a drift $|\dot{x}_i - \dot{x}_{i+1}| \leq 1$ when cruising and $|\dot{x}_i - \dot{x}_{i+1} - \varepsilon| \leq 1$ when recovering, where ε is the slow-down parameter. The cars can stay in cruise mode as long as the distance to the preceding vehicle is greater 1. The can go into recovery mode when the distance is smaller than 2.

ACCS*nn* The model with *nn* cars, $\varepsilon = 2$. This model is considered safe with respect to specification UBS*nn* (no collisions).

ACCU*nn* The model with *nn* cars, $\varepsilon = 0.9$. This model is considered unsafe with respect to specification UBS*nn* (collisions are possible).

Specification The distance between adjacent cars should be positive:

$$x_{\texttt{ldr}} - x > 0,$$

where x and $x_{\texttt{ldr}}$ are the positions of the car and the car in front, respectively.

UBD*nn* For $i = 0, \ldots, n-1$: $x_i - x_{i+1} > 0$.

Results The computation times of various tools are listed in Tab. 1.

Note about the HyCOMP's results. HyCOMP proves that the property UBD01 is safe for a model ACCS*nn* almost instantaneously because the property is inductive. The property that no cars collide ($\forall i \in [0, n), x_i - x_{i+1} > 0$) holds in the initial state. In the initial states no cars collide (base case). Then, assuming that no cars collide (by inductive hypothesis) we see that there are no transitions in the system that make the system unsafe. If the system is safe and an automaton changes location, then the system is still safe (the automata transition do not change the value of the car's position). The safety property cannot be false when an automaton is in the cruise location (due to the location invariant), and in the recovery location the difference of a car's velocity with the following car's velocity is non-negative (so, if the position was positive before, it is the case by inductive hypothesis, it will be positive even after some time elapses).

3.2 Distributed controller

Model The benchmark is an extension of the benchmarks presented in [23], to which multiple sensors with multiple priorities have been added. It models the distributed controller for a robot that reads and processes data from different sensors. A scheduler component determines what sensor data must be read according to the priority of the sensor. The controller has 1 continuous and n discrete variables, the scheduler has n continuous and n discrete variables, and each sensor has 1 continuous variable. The controller has 4 locations, the scheduler has $1 + n$, and each sensor has 4 locations. The product automaton has $4 \times (1 + n) \times 4^n$ locations, $2n + 1$ continuous variables and $2n$ discrete variables.[2] Note that some tools, such as PHAVer/SX and PHAVerLite, do not support discrete variables and may model the discrete variables as continuous variables.

DISC*nn* The model with *nn* sensors. This model is considered safe with respect to specification UBS*nn*.

[2] In previous editions of this report, it was erroneously claimed to be $4 \times (1 + n) \times 4$ locations and $n + 2$ continuous variable.

Table 1: Computation Times of the Adaptive Cruise Controller.

instance	ACCS05 UBD01	ACCU05 UBD01	ACCS06 UBD01	ACCU06 UBD01	ACCS07 UBD01	ACCU07 UBD01	ACCS08 UBD01	ACCU08 UBD01
safety	safe	unsafe	safe	unsafe	safe	unsafe	safe	unsafe
#vars.	5	5	6	6	7	7	8	8
#locs.	32	32	64	64	128	128	256	256
tool	computation time in [s]							
Lyse	1.08	≈ 0	–	–	573.35	0.233	–	–
PHAVer/SX	9.4	13.7	461	13430	∞	∞	–	–
PHAVerLite	0.10	0.06	0.55	0.27	4.26	1.39	47.10	7.15
HyCOMP-IC3	0.1	0.2	0.1	0.3	0.1	0.4	0.1	0.4
	bounded-depth tools							
HyCOMP-BMC	–	0.2	–	0.2	–	0.2	–	0.2

Table 2: Computation Times of the Distributed Controller.

instance	DISC02 UBS02	DISC03 UBS03	DISC04 UBS04	DISC05 UBS05
safety	safe	safe	safe	safe
#vars.	9	13	17	21
#locs.	192	1024	5120	24976
tool	computation time in [s]			
PHAVer/SX	1.1	∞	∞	∞
PHAVerLite	0.04	0.68	77.51	∞
HyCOMP-IC3	0.1	0.3	0.7	0.9
	bounded-depth tools[3]			
BACH	–	–	0.1($B:10$)	0.2($B:10$)

Specification The system is considered safe if at no point in time all sensors send data simultaneously.

UBS*nn* It is never the case that all *nn* sensors are in location `send`.

Results The computation times of various tools are listed in Tab. 2.

[3]The search depth p is indicated as $(B:p)$, and counted as the number of discrete transitions taken.

3.3 Dutch Railway Network

We consider a finite-horizon safety problem over max-plus-linear (MPL) systems. More precisely, given a PWA system generated from an MPL system, a time horizon N, a set of initial conditions X_0 expressed as a difference-bound matrix (DBM) [20], an unsafe set S described as a DBM, we wanted to know whether the system can reach the unsafe set within the given time horizon.

Model In [26, p. 30], a subset of Dutch railway networks is modeled as a max-plus-linear (MPL) system. That model has 14 state variables $x_1(k), \ldots, x_{14}(k)$ representing the k-th departure time of trains. For a complete description, see [22]. The model instance is defined formally as follows:

DRNW02 initial condition $X_0 = \{x : 0 \leq x_i \leq 5, \text{ for all } i = 1, \ldots, 14\}$

The model is easily embedded in a hybrid automaton with a single location, where the time derivative of all variables is zero, and a self-loop transition that models the discrete dynamics for each region.

Specification We have four specifications of interest:

BDR01 there exists a $k = 0, \ldots, 100$ such that $50 \leq x_2(k) - x_7(k) \leq 60$ (satisfied)

BDR02 there exists a $k = 0, \ldots, 100$ such that $70 \leq x_2(k) - x_7(k) \leq 80$ (satisfied)

BDR03 there exists a $k = 0, \ldots, 100$ such that $90 \leq x_2(k) - x_7(k) \leq 100$ (not satisfied)

BDR04 there exists a $k = 0, \ldots, 100$ such that $10 \leq x_2(k) - x_7(k) \leq 20$ (satisfied)

In the sense of a safety specification, the above specifications specify unsafe states. If the unsafe are reachable, the corresponding specification BDR01,. . . ,BDR04 is satisfied.

Results The computation times of various tools are listed in Tab. 3.

Note PHAVer/SX and PHAVerLite Since the iteration count in PHAVer/SX and PHAVerLite does not guarantee the actual search depth, we added a counter automaton that models each value of k with a discrete location. The counter is limited to 100 transitions, after which it deadlocks. The tool is then run until a fixed point is found, which guarantees that all values up to $k = 100$ are explored. The flow predicate in the hybrid automaton model was set to `false`, which means that there is no computation of time elapse in the reachability. We therefore expect that the overhead of embedding the discrete-time model in a continuous-time model is minimal. Note that PHAVer/SX computes the full reach set before checking whether the unsafe states are reachable. This explains why all instances take the same time, regardless of the specification. In contrast, PHAVerLite was configured to check for reachable unsafe states during the fixpoint computation, thereby terminating slightly sooner on unsafe models. To compare the performance of exact polyhedral computations with that of template polyhedra, we also include the results of SpaceEx running the LGG scenario and box directions. This gives the same variable ranges as PHAVer/SX, but it should be noted that the result is not formally sound due to the double precision floating point used by LGG.

[4] In contrast to the other tools, the results of SpaceEx given here are numerically unsound due to the use of double precision floating point arithmetic.

[5] The search depth p is indicated as $(B : p)$, and counted as the number of discrete transitions taken.

[6] We run BMC up to 100 steps showing that $90 \leq x_2(k) - x_7(k) \leq 100$ is not satisfied at any k.

Table 3: Computation Times of the Dutch Railway Benchmark.

instance	DRNW02 BDR01	DRNW02 BDR02	DRNW02 BDR03	DRNW02 BDR04
safety	unsafe	unsafe	safe	unsafe
# vars.	14	14	14	14
# locs.	1	1	1	1
tool	computation time in [s]			
VeriSiMPL	0.057	0.030	6.081	0.033
PHAVer/SX	528.1	528.1	528.1	528.1
PHAVerLite	11.11	10.46	17.36	10.49
SpaceEx[4]	1.2	1.2	1.2	1.2
HyCOMP-IC3	57.91	0.15	4.76	0.55
	bounded-depth tools[5]			
BACH	1.99($B:100$)	0.09($B:100$)	–	0.9($B:100$)
HYCOMP-BMC	1.0	0.1	8.9[6]	0.2

3.4 Fischer's Protocol

Model Fischer's protocol is a time based protocol of mutual exclusion between processes, originally from [25]. The flow constraints are given by $\frac{1}{2} \leq \dot{x}_1 \leq \frac{3}{2}, \ldots, \frac{1}{2} \leq \dot{x}_{\mathtt{m}} \leq \frac{3}{2}$, where $x_{\mathtt{i}}$ is the clock of the `i`-th process. The product automaton has $(n+1) \times 4^n$ locations and n variables.

FISCSnn protocol with nn processes, considered safe with respect to specification UBDnn.

FISCUnn protocol with nn processes, considered unsafe with respect to specification UBDnn.

Specification The protocol is correct if no two processes are ever in the critical section at the same time.

UBDnn There are no two processes such that both are in location `cs` (critical section) at the same time.

Results The computation times of various tools are listed in Tab. 4.

Note on PHAVerLite As said before, PHAVerLite is sometimes configured to check for reachable unsafe states during the fixpoint computation. However, the analysis always starts by eagerly computing the full parallel composition of the automata components. This can be seen as a waste of computational effort: for instance, the FISCU06-UBD01 benchmark is analysed in 4.90 seconds, but only 0.16 seconds are actually spent for checking reachability.

Table 4: Computation Times of the Fischer Benchmark.

instance	FISCS04 UBD01	FISCU04 UBD01	FISCS05 UBD01	FISCU05 UBD01	FISCS06 UBD01	FISCU06 UBD01
safety	safe	unsafe	safe	unsafe	safe	unsafe
# vars.	4	4	5	5	6	6
# locs.	1280	1280	6144	6144	28672	28672
tool	computation time in [s]					
Lyse	33.59	0.06	859.84	0.16	–	–
PHAVer/SX	90.5	579	∞	∞	–	–
PHAVerLite	0.22	0.13	1.40	0.81	11.42	4.90
HyCOMP-IC3	6.7	0.2	26.5	0.5	139.7	0.2

3.5 TTEthernet

Model The TTEthernet protocol is a protocol for the remote synchronization of possibly drifted clocks distributed over multiple components, taken from [11]. The system consists of two compression masters (CM) and k synchronisation masters (SM). Each CM has two clocks cm_i, each SM has one clock sm_i. Both CM and SM are modeled by a hybrid automaton with 4 locations each. The product automaton has $4 + k$ variables and 4^{k+2} locations.

TTESnn protocol with nn SM. This model is considered safe with respect to specification UBDnn. The global time horizon is limited to 3000 ms.

Specification The difference between the clocks of the SM should not exceed a threshold of $2max\_drift$.

UBDnn For all i, j, $sm_i - sm_j \leq 2max\_drift$, where $max\_drift = 0.001$ ms.

Results The computation times of various tools are listed in Tab. 5.

4 Conclusions and Outlook

This report presents the results of the third edition of a friendly competition for the formal verification of continuous and hybrid systems of the ARCH'19 workshop, in the category on piecewise constant dynamics. The reports of other categories can be found in the proceedings and on the ARCH website: cps-vo.org/group/ARCH. The code with which the results have been obtained is publicly available at gitlab.com/goranf/ARCH-COMP.

In the spirit of a friendly competition, this report does not provide any ranking of tools. We report a few casual observations. For the reported instances, PHAVer/SX computes the exact set of reachable states and can therefore be regarded as a base line. PHAVerLite is a variation of PHAVer that uses a different polyhedra library and includes some algorithmic improvements that lead to a considerable speedup, here included configuration settings to use approximated, cheaper operators that are anyway precise enough for the considered verification

[7] The search depth p is indicated as $(B : p)$, and counted as the number of discrete transitions taken.

Table 5: Computation Times of the TTEthernet Benchmark.

instance	TTES05 UBS01	TTES07 UBS01	TTES09 UBS01
safety	safe	safe	safe
# vars.	9	11	13
# locs.	15384	262144	4194304
tool	computation time in [s]		
PHAVer/SX	25.2	113	–
PHAVerLite	0.33	1.76	12.91
HyCOMP-IC3	0.4	0.7	1.1
	bounded-depth tools[7]		
BACH	$0.15(B:11)$	$0.2(B:11)$	–

tasks. Lyse uses abstraction refinement, which leads to considerable performance gains in many instances. VeriSiMPL was developed for a very specific subclass of problems, in which shows very good performance. This year, HyCOMP has joined the competition. HyCOMP (when using the IC3-IA algorithm) finds an inductive invariant sufficient to prove the safety property, instead of computing the system's reachable states (further using predicate abstraction to tackle the system's complexity). The experimental results on the competition's instances show that finding an inductive invariant is often more effective than computing the set of reachable states, as PHAVerLite and PHAVer/SX. The bounded model checker BACH was included for rough comparison and to create a link to the ARCH-COMP category on bounded model checking (HBMC). For the reported depths, BACH performed very well.

5 Acknowledgments

The authors gratefully acknowledge financial support by the European Commission project UnCoVerCPS under grant number 643921. Lei Bu is supported by the National Natural Science Foundation of China (No.61572249).

References

[1] D. Adzkiya and A. Abate. VeriSiMPL: Verification via biSimulations of MPL models. In K. Joshi, M. Siegle, M. Stoelinga, and P.R. D'Argenio, editors, *International Conference on Quantitative Evaluation of Systems*, volume 8054 of *Lecture Notes in Computer Science*, pages 253–256. Springer, Heidelberg, September 2013. http://sourceforge.net/projects/verisimpl/.

[2] D. Adzkiya, B. De Schutter, and A. Abate. Finite abstractions of max-plus-linear systems. *IEEE Transactions on Automatic Control*, 58(12):3039–3053, December 2013.

[3] D. Adzkiya, B. De Schutter, and A. Abate. Computational techniques for reachability analysis of max-plus-linear systems. *Automatica*, 53(0):293–302, 2015.

[4] D. Adzkiya, Y. Zhang, and A. Abate. VeriSiMPL 2: An open-source software for the verification of max-plus-linear systems. *Discrete Event Dynamic Systems*, 26(1):109–145, 2016.

[5] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. Applications of polyhedral computations to the analysis and verification of hardware and software systems. *Theor. Comput. Sci.*, 410(46):4672–4691, 2009.

[6] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. 2009.

[7] A. Becchi and E. Zaffanella. A direct encoding for NNC polyhedra. In H. Chockler and G. Weissenbacher, editors, *Computer Aided Verification*, pages 230–248, Cham, 2018. Springer International Publishing.

[8] A. Becchi and E. Zaffanella. An efficient abstract domain for not necessarily closed polyhedra. In A. Podelski, editor, *Static Analysis - 25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings*, volume 11002 of *Lecture Notes in Computer Science*, pages 146–165. Springer, 2018.

[9] Sergiy Bogomolov, Goran Frehse, Mirco Giacobbe, and Thomas A Henzinger. Counterexample-guided refinement of template polyhedra. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 589–606. Springer, 2017.

[10] Sergiy Bogomolov, Goran Frehse, Mirco Giacobbe, and Thomas A. Henzinger. Counterexample-guided refinement of template polyhedra. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 589–606, 2017.

[11] Sergiy Bogomolov, Christian Herrera, and Wilfried Steiner. Benchmark for verification of fault-tolerant clock synchronization algorithms. In *ARCH (2016)*.

[12] Aaron R. Bradley. Sat-based model checking without unrolling. In *VMCAI*, pages 70–87, 2011.

[13] Lei Bu, You Li, Linzhang Wang, Xin Chen, and Xuandong Li. BACH 2 : Bounded reachability checker for compositional linear hybrid systems. In *Design, Automation and Test in Europe, DATE 2010, Dresden, Germany, March 8-12, 2010*, pages 1512–1517, 2010.

[14] Lei Bu, You Li, Linzhang Wang, and Xuandong Li. BACH : Bounded reachability checker for linear hybrid automata. In *Formal Methods in Computer-Aided Design, FMCAD 2008, Portland, Oregon, USA, 17-20 November 2008*, pages 1–4, 2008.

[15] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *CAV*, pages 334–342, 2014.

[16] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Hycomp: An smt-based model checker for hybrid systems. In *Proc. of TACAS'15*, volume 9035 of *LNCS*, pages 52–67. Springer, 2015.

[17] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. IC3 modulo theories via implicit predicate abstraction. In *TACAS*, pages 46–61, 2014.

[18] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Infinite-state invariant checking with IC3 and predicate abstraction. *Formal Methods in System Design*, 49(3):190–218, 2016.

[19] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The mathsat5 SMT solver. In *TACAS*, pages 93–107, 2013.

[20] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, chapter 17, pages 197–212. Springer, Heidelberg, 1990.

[21] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. *International Journal on Software Tools for Technology Transfer*, 10:263–279, 2008.

[22] Goran Frehse, Alessandro Abate, Dieky Adzkiya, Lei Bu, Mirco Giacobbe, Muhammad Syifa'Ul Mufid, and Enea Zaffanella. Arch-comp18 category report: Hybrid systems with piecewise constant dynamics. In Goran Frehse, editor, *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 54 of *EPiC Series in Computing*, pages 1–13. EasyChair, 2018.

[23] Thomas A. Henzinger and Pei-Hsin Ho. HYTECH: the cornell hybrid technology tool. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, pages 265–293. Springer, 1994.

[24] Sumit Kumar Jha, Bruce H. Krogh, James E. Weimer, and Edmund M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In *HSCC*, 2007.

[25] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems (TOCS)*, 5(1):1–11, 1987.

[26] Subiono. *On classes of min-max-plus systems and their applications*. PhD thesis, Delft University of Technology, 2000. http://resolver.tudelft.nl/uuid:e5181e99-fb8f-4588-a37a-46ff2007c5c7.

[27] Stefano Tonetta. Abstract model checking without computing the abstraction. In *FM*, pages 89–105, 2009.

[28] Dingbao Xie, Lei Bu, Jianhua Zhao, and Xuandong Li. SAT-LP-IIS joint-directed path-oriented bounded reachability analysis of linear hybrid automata. *Formal Methods in System Design*, 45(1):42–62, 2014.

[29] Dingbao Xie, Wen Xiong, Lei Bu, and Xuandong Li. Deriving unbounded reachability proof of linear hybrid automata during bounded checking procedure. *IEEE Trans. Computers*, 66(3):416–430, 2017.

A Implementation Languages and Used Machines

A.1 BACH

- Implementation language: C++
- Processor: Intel(R) Core(TM)2 Quad CPU Q9500 @ 2.83GHz x 4
- Memory: 4 GB
- Average CPU Mark on www.cpubenchmark.net: 3636 (full), 1203 (single thread)

A.2 HyCOMP

- Implementation language: C, C++
- Processor: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
- Memory: 16 GB
- Average CPU Mark on www.cpubenchmark.net: 11003 (full), 2221 (single thread)

A.3 Lyse

- Implementation language: C++
- Processor: Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz x 2
- Memory: 4 GB
- Average CPU Mark on www.cpubenchmark.net: 3818 (full), 1521 (single thread)

A.4 PHAVer/SX

- Implementation language: C++
- Processor: Intel Core i7-4850HQ CPU @ 2.30GHz x 4
- Memory: 15.9 GB
- Average CPU Mark on www.cpubenchmark.net: 9057 (full), 1966 (single thread)

A.5 PHAVerLite

- Implementation language: C++
- Processor: Intel Core i7-3632QM CPU @ 2.20GHz x 4
- Memory: 15.5 GB
- Average CPU Mark on www.cpubenchmark.net: 6939 (full), 1566 (single thread)

A.6 VeriSiMPL

- Implementation language: MATLAB
- Processor: Intel Core i7-4720HQ CPU @ 2.6GHz x 4
- Memory: 4 GB
- Average CPU Mark on www.cpubenchmark.net: 8010 (full), 1912 (single thread)