Current Challenges in the Verification of Hybrid Systems

Stefan Schupp[1(✉)], Erika Ábrahám[1], Xin Chen[1], Ibtissem Ben Makhlouf[1], Goran Frehse[2], Sriram Sankaranarayanan[3], and Stefan Kowalewski[1]

[1] RWTH Aachen University, Aachen, Germany
stefan.schupp@cs.rwth-aachen.de
[2] Verimag, Gières, France
[3] University of Colorado, Boulder, CO, USA

Abstract. Latest developments brought interesting theoretical results and powerful tools for the reachability analysis of hybrid systems. However, there are still challenging problems to be solved in order to make those technologies applicable to large-scale applications in industrial context. To support this development, in this paper we give a brief overview of available algorithms and tools, and point out some of their individual characteristics regarding various properties which are crucial for the verification of hybrid systems. We present exemplary evaluations on three benchmarks to motivate the need for further development and discuss some of the main challenges for future research in this area.

Keywords: Hybrid systems · Verification · Reachability analysis · Tool support · Benchmarks

1 Introduction

Hybrid systems are systems containing both physical components which evolve continuously over time, as well as discrete components which can influence the continuous dynamics. Also cyber-physical systems can be seen as hybrid systems, where communication between distributed components plays a further important role.

As hybrid systems are often safety critical, in the last two decades much effort was put into the development of efficient algorithms and powerful tools to support their safety analysis. Whereas there is a deep-rooted research for pure continuous and for pure discrete systems, their hybrid combination requires novel methodologies and the adaptation, integration and extension of previous results.

Nowadays, a number of analysis tools for hybrid systems are available, such as Ariadne [13], Cora [1], dReach [26], Flow* [12], HSolver [36], HyCreate [25], iSAT-ODE [15], KeYmaera [32] and SpaceEx [20]. These tools implement

This work was partially supported by the German Research Council (DFG) in the context of the HyPro project.

C. Berger and M.R. Mousavi (Eds.): CyPhy 2015, LNCS 9361, pp. 8–24, 2015.
DOI: 10.1007/978-3-319-25141-7_2

different analysis techniques, leading to individual strength and weaknesses. For further development it is crucial to learn from previous results by evaluating these tools to observe and compare their behaviours, and to identify common obstacles and open problems. Our aim is to support this development by

- describing current analysis techniques, available tools and their individual properties,
- providing exemplary evaluation of a few tools on some benchmarks, and discussing general problems related to tool evaluation and comparison, and
- collecting some important challenges for future research in this area.

The paper is organised as follows: In Sect. 2 we provide some background on hybrid systems, their modelling, and techniques for their reachability analysis. In Sect. 3 we give a brief overview of some tools and discuss their individual properties. On the basis of some evaluations in Sect. 4, we collect challenges and open problems for future research in Sect. 5, and conclude the paper in Sect. 6.

2 Hybrid Systems Modelling and Reachability Analysis

Hybrid systems are systems with combined discrete-continuous behaviour. Typical examples are digitally controlled physical processes, or physical processes with inherent discrete state changes such as phase transitions.

2.1 Modelling

Besides hybrid Petri nets and hybrid programs, a popular modelling formalism for hybrid systems are *hybrid automata* [23,24]. We give a simplified notion of hybrid automata, where we neglect components which are only relevant for their parallel composition.

Definition 1 (Hybrid automata: Syntax [23]). *A* hybrid automaton *is a tuple* $\mathcal{H} = (Loc, Var, Flow, Inv, Edge, Init)$ *consisting of:*

- *A finite set* Loc *of* locations *or* control modes.
- *A finite ordered set* $Var = \{x_1, \ldots, x_n\}$ *of real-valued* variables*; we also use the vector notation* $\boldsymbol{x} = (x_1, \ldots, x_n)$*. The number* n *is called the* dimension *of* $\mathcal{H}$*. By* $\dot{Var}$ *we denote the set* $\{\dot{x}_1, \ldots, \dot{x}_n\}$ *of dotted variables (which represent first derivatives during continuous change), and by* Var' *the set* $\{x'_1, \ldots, x'_n\}$ *of primed variables (which represent values directly after a discrete change). Furthermore,* $Pred_X$ *is the set of all predicates with free variables from* X*.*
- $Flow : Loc \to Pred_{Var \cup \dot{Var}}$ *specifies for each location its* flow *or* dynamics.
- $Inv : Loc \to Pred_{Var}$ *assigns to each location an* invariant.
- $Edge \subseteq Loc \times Pred_{Var} \times Pred_{Var \cup Var'} \times Loc$ *is a finite set of* discrete transitions *or* jumps*. For a jump* $(l_1, g, r, l_2) \in Edge$*,* l_1 *is its* source *location,* l_2 *is its* target *location,* g *specifies the jump's* guard*, and* r *its* reset *function, where primed variables represent the state after the step.*
- $Init : Loc \to Pred_{Var}$ *assigns to each location an* initial *predicate.*

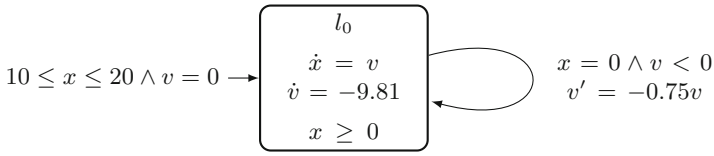

Fig. 1. The hybrid automaton modelling a bouncing ball with height x and velocity v.

Example 1 (Bouncing ball). In the classical *bouncing ball* example, a ball is dropped from some initial height with zero initial velocity. Due to gravity, the ball has an acceleration pointing towards the earth. Therefore the ball falls until it hits the ground, it bounces back into the air, raises until its velocity gets zero, and starts to fall again. Upon bouncing, the ball loses a fraction of its kinetic energy.

An example hybrid automaton model for the bouncing ball is shown graphically in Fig. 1. The dynamics of raising and falling is modelled in a single mode $Loc = \{l_0\}$ using two variables $Var = \{x, v\}$, where x models the vertical position (height) and v the vertical velocity of the ball. The flow $Flow(l_0)$ is specified by the predicate $\dot{x} = v \wedge \dot{v} = -9.81$ with the gravitational force as the only influence on the speed of the ball. The invariant $Inv(l_0)$ is $x \geq 0$, which enforces that the ball bounces when it reaches the ground. This bouncing is represented by the only jump $Edge = \{(l_0, g, r, l_0)\}$ with guard g given by $x = 0 \wedge v < 0$ (that means bouncing only occurs when the ball falls from above and reaches the ground) and reset r specified by $v' = 0.75v$ (i.e., the sign of the velocity gets inverted and the velocity is dampened by a constant factor 0.75). The initial states are described by $Init(l_0) = (10 \leq x \leq 20 \wedge v = 0)$.

The behaviour of a hybrid automaton can be given by an operational semantics. The *states* of an n-dimensional hybrid automaton are pairs $(l, \boldsymbol{v})$, where $l \in Loc$ is the current location and $\boldsymbol{v} \in \mathbb{R}^n$ specifies the current values of the variables. *Initial* states $(l, \boldsymbol{v})$ satisfy both the initial and the invariant conditions of location l. State changes are due to time and discrete steps. A *time step* models the passage of time: while control stays in a location, the values of the variables evolve continuously according to a function which satisfies the flow condition of the current location. Furthermore, the invariant of the location must not be violated during the whole time step. Given a set of states, the states which can be visited from it via time evolution according to the flow in the given location form a *flowpipe*. When flows are described by linear predicates (i.e., linear differential equations) we talk about *linear dynamics*, in the case of polynomial predicates about *non-linear dynamics*. *Discrete steps* follow a jump, moving the control from one location to another, given that the jump's guard is satisfied in the predecessor state. The successor state, resulting from variable resets satisfying the reset condition, must satisfy the invariant of the target location.

Definition 2 (Hybrid automata: Semantics). *The one-step semantics of a hybrid automaton* $\mathcal{H} = (Loc, Var, Flow, Inv, Edge, Init)$ *of dimension* n *is specified by the following operational semantics rules:*

$$\frac{\begin{array}{c} l \in Loc \quad \boldsymbol{v}, \boldsymbol{v}' \in \mathbb{R}^n \\ f : [0,\delta] \to \mathbb{R}^n \quad df/dt = \dot{f} : (0,\delta) \to \mathbb{R}^n \quad f(0) = \boldsymbol{v} \quad f(\delta) = \boldsymbol{v}' \\ \forall \epsilon \in (0,\delta).\ f(\epsilon), \dot{f}(\epsilon) \models Flow(l) \quad \forall \epsilon \in [0,\delta].\ f(\epsilon) \models Inv(l) \end{array}}{(l, \boldsymbol{v}) \xrightarrow{\delta} (l, \boldsymbol{v}')} \textit{Rule}_{\,flow}$$

$$\frac{e = (l, g, r, l') \in Edge \quad \boldsymbol{v}, \boldsymbol{v}' \in \mathbb{R}^n \quad \boldsymbol{v} \models g \quad \boldsymbol{v}, \boldsymbol{v}' \models r \quad \boldsymbol{v}' \models Inv(l')}{(l, \boldsymbol{v}) \xrightarrow{e} (l', \boldsymbol{v}')} \textit{Rule}_{\,jump}$$

A path *of* $\mathcal{H}$ *is a (finite or infinite) sequence* $(l_0, \boldsymbol{v}_0) \xrightarrow{\delta_0} (l_1, \boldsymbol{v}_1) \xrightarrow{e_1} (l_2, \boldsymbol{v}_2) \xrightarrow{\delta_2} (l_3, \boldsymbol{v}_3) \xrightarrow{e_3} (l_4, \boldsymbol{v}_4) \xrightarrow{\delta_4} \ldots$ *with* $(l_i, \boldsymbol{v}_i)$ *states of* H, $\delta_i \in \mathbb{R}_{\geq 0}$, $e_i \in Edge$, *and* $\boldsymbol{v}_0 \models Init(l_0) \wedge Inv(l_0)$. *A state* $(l, \boldsymbol{v})$ *is* reachable *in* $\mathcal{H}$ *if there is a path* $(l_0, \boldsymbol{v}_0) \xrightarrow{\delta_0} (l_1, \boldsymbol{v}_1) \xrightarrow{e_1} (l_2, \boldsymbol{v}_2) \xrightarrow{\delta_2} \ldots$ *of* $\mathcal{H}$ *with* $(l, \boldsymbol{v}) = (l_i, \boldsymbol{v}_i)$ *for some* $i \geq 0$.

2.2 Reachability Analysis

The *reachability problem* for hybrid automata, i.e. the problem to decide whether a given set of states is reachable in a hybrid automaton, is in general undecidable. Nevertheless, there exist subclasses of hybrid automata for which the reachability problem is decidable. For undecidable classes, tools often compute *jump-bounded reachability* (reachability via paths with a limited number of jumps) or *time- and jump-bounded reachability* (where additionally the time step lengths are bounded).

Some of those tools implement *flowpipe-construction-based methods*, which over-approximate the flowpipe over a bounded time horizon by dividing the time horizon into smaller segments (whose length is called the *time-step size*) and over-approximating the flowpipe for each time segment by a single state set. These methods use over-approximative *geometric and/or symbolic representations* [27] of state sets, e.g., by boxes (hyper-rectangles), convex polytopes, zonotopes, ellipsoids, support functions or Taylor models. Given an initial state set, its flowpipe and its discrete successors are computed using efficient operations on such state set representations and safe (over-approximative) conversions between them. User-defined parameters and different techniques for reducing the number of the state sets and the sizes of their representations (on the cost of a stronger over-approximation) allow to find a balance between *efficiency* and *precision* of the computations. These techniques have their strength in a high level of automation and in the possibility to increase efficiency or improve the precision according to the needs of the user. A weakness lies in the fact that, due to over-approximative techniques, only safety (non-reachability) can be proven this way, but not unsafety (reachability).

Some other solutions use *satisfiability checking* algorithms for the reachability analysis, which is based on the formulation of the one-step reachability relation as mixed integer-real arithmetic formulas. Fast SAT-modulo-theories (SMT) solvers

can be used if the solutions of the Ordinary Differential Equations (ODEs) in the models are known (e.g., in the case of constant derivatives). When the solutions are not known, the underlying theories in the solvers can also be extended to cope with ODEs. These techniques can efficiently combine a wide range of decision procedures for expressive theories and can theoretically prove both safety and unsafety. However, running times are hard to predict and computations might return inconclusive answers, even for decidable problems, if fast but incomplete solving techniques (e.g., interval constraint propagation) are used.

Last but not least, some other tools are based on *theorem proving* with an embedded theory for hybrid systems. On the one hand, these techniques are very powerful and can handle (at least in theory) a wide range of models using deduction. On the other hand, these approaches are interactive and need experienced users. Predefined and user-defined strategies can be of great help to increase the level of automation and reduce the need for interaction to a minimal level.

3 Tools

The vast variety of tools for hybrid systems verification makes it impossible to rate one particular tool above the others. Each tool brings its strengths and weaknesses, which make it suitable for a certain purpose. Knowing these differences allows users to choose the right tool for their problem requirements. In this section we provide an overview of some of the most popular tools (in alphabetical order) and describe their main capabilities and features; see Table 1 for a short summary.

Ariadne [13] is a software package implementing functionalities for the reachability analysis of hybrid systems. The package is based on the theory of computable analysis and on a rigorous function calculus with provable approximation bounds on the computations. Ariadnecan handle expressive models with non-linear differential equations, where state sets can be represented by Taylor models or grid pavings. Besides others, interval arithmetic along with interval solvers and propagation mechanisms are applied in the computations. The support for parallel composition and assume-guarantee reasoning improve scalability.

Cora [1] is an object-oriented Matlab toolbox which can be used for the fast implementation of different reachability analysis algorithms for continuous and hybrid systems. It implements different state set representation types, conversion algorithms between them, and operations needed for reachability analysis. Additionally to well-known representations such as boxes, polytopes and zonotopes, it provides also non-convex representations (polynomial zonotopes) and representations dedicated to stochastic verification (probabilistic zonotopes). Coracan be used for the analysis of systems with linear, linear stochastic and non-linear dynamics with uncertain parameters, where non-linear systems are abstracted by linear or polynomial systems.

dReach [26] is an SMT-based tool performing bounded model checking. Unsafe system runs of bounded length are described by formulas and passed

Table 1. Some hybrid systems reachability analysis tools and their characteristic functionalities.

Tool	
ARIADNE	non-linear ODEs; Taylor models, boxes; interval constraint propagation, deduction
CORA	non-linear ODEs; geometric state set representations; several reachability analysis algorithms, linear abstraction
DREACH	non-linear ODEs; logical state set representation; interval constraint propagation, δ-reachability, bounded model checking
FLOW*	non-linear ODEs; Taylor models; flowpipe computation
HSOLVER	non-linear ODEs; logical state set representation; interval constraint propagation
HYCREATE	non-linear ODEs; boxes; flowpipe computation
ISAT-ODE	non-linear ODEs; logical state set representation; interval constraint propagation, bounded model checking
KEYMAERA	differential dynamic logic; logical state set representation; deduction, computer algebra
SPACEEX	linear ODEs; geometric and symbolic state set representations; flowpipe computation

on to the internal SMT solver DREAL [22], which determines its δ-satisfiability using interval constraint propagation. Due to the generality of interval constraint propagation, DREACH is able to handle non-linear dynamics involving transcendentals. The user can access the SMT calls in SMT-LIB format [5] as well as a witness for the reachability of the set of bad states.

FLOW* is a tool to compute reachable set over-approximations using Taylor-model-based methods. It is able to handle an expressive class of hybrid system models such that the continuous dynamics can be defined by non-linear ODEs with uncertainties, while the jump guards and mode invariants are defined by polynomial inequalities. The basic technique in use is called *Taylor model flowpipe construction* which is described in [11] and later enhanced by more efficient algorithms [10]. By properly setting the parameters, the tool shows a good scalability on non-linear case studies and succeeds even on large initial sets. Since the tool focuses on non-linear systems, its performance on handling convex guards or invariants is not optimised.

HSOLVER [36] implements classical interval constraint propagation on top of the constraint solving package RSOLVER. Due to its general solving technique, it can handle expressive non-linear ODEs and non-linear jumps. Though HSOLVER uses floating point arithmetic, it uses sound rounding to assure correct results. Besides verification purposes, the tool can also be used to compute abstractions.

HYCREATE [25] is a tool implemented in Java for both time-bounded and unbounded (complete) reachability analysis from an initial state. The tool is designed for low-dimensional models with non-linear, non-deterministic dynamics. It uses box representation and provides error reduction by splitting boxes

at the cost of increased complexity. HyCreate allows further processing of the generated output as well as visualisation via projection on a 2D space.

iSAT-ODE [15] performs, similarly to dReach, bounded model checking. It is based on the iSAT [17] SMT solver, which tightly integrates interval constraint propagation into a SAT solver. iSAT-ODE extends iSAT with a theory solver module for ODEs to compute validated numerical enclosures for them using the VNODE-LP [31] library. This approach can handle expressive models with non-linear dynamics and transcendental functions. However, despite different embedded optimisation mechanisms, this expressiveness comes at the cost of scalability.

KeYmaera [32] is an interactive hybrid tool combining deductive, real algebraic, and computer-algebraic prover technologies. Hybrid systems are specified in differential dynamic logic [33] using the notation of hybrid programs, covering non-linear dynamics under uncertainties and non-linear jumps. KeYmaera tries to prove properties of a given system by finding invariants. On the one hand, this approach is automated but it is still inherently interactive. On the other hand it is flexible, can cope also with infinite time horizons and parametric models, and can provide verified counterexamples. A new re-implementation KeYmaera X [21] is in its early development phase and it can therefore handle only a restricted model class, but it additionally allows the user to define their own proof search techniques as tactics.

SpaceEx [20] is designed for complex, high-dimensional models with piecewise affine dynamics and non-deterministic inputs. SpaceEx comes with a web-based graphical user interface and a graphical model editor. Its input language facilitates the construction of complex models from automata components using a block-diagram representation. The analysis engine of SpaceEx offers different algorithms (LGG [20,28], STC [18,19]) which combine geometric state set representations (template polyhedra), symbolic state set representations (support functions) and linear programming to achieve maximal scalability while maintaining high accuracy. The prime goal of SpaceEx being scalability, it uses floating-point computations that do not formally guarantee soundness.

4 Benchmarking and Evaluation

Although there are many tools available, their comparative evaluation is problematic. First of all, they do not support the same model classes. The main differences concern the type of the supported ODEs. Though theoretically unspectacular, some tools cannot handle jumps with guard predicate *true*, or unspecified (arbitrary) dynamics. Even if the user identified those tools which can handle a given model class, it is hard to compare their performance: as each algorithm brings its own set of parameters, it requires expertise and knowledge about implementation details to properly instantiate the tool parameters to get optimal results.

Other obstacles are the relatively low number of available benchmarks and missing input language standards. In some other communities, e.g. in SAT and SMT solving or in software verification, the development of such standards

and the organisation of annual competitions gave impressive force and led to a new sequence of innovations in the given areas. A standardised specification language for hybrid system models could have a similar positive effect. Currently, the number of available benchmarks is not satisfactory, even though lately some improvements were achieved [2,8,16]. The situation is worsened by the fact that nearly each tool has its own input specification language. To solve this problem, a CIF 3 standard was proposed [6], however, it is not yet widely established in the community. Furthermore, some approaches for model conversion were proposed in [4]. A standardisation could drastically improve the situation, enable the establishment of a competition, give new drive to tool development and thus contribute to stronger tool functionalities and better efficiency, and ease the selection of a suitable tool.

To give an impression for the analysis capabilities of available tools and to motivate some challenges, in the following we give some exemplary verification results, where we focus on limitations.

Two tanks [8,29]: A two-tank system consists of two connected tanks. The first is filled with a constant inflow and an additional controlled inflow of a liquid. A drain at the bottom of the first tank leads to a constant outflow and thus a constant inflow in the second tank. Conversely, the second tank has a drain which creates a constant outflow, and a controlled valve which results in an additional controlled outflow. The hybrid automaton model of this tank system has four locations, corresponding to the different states of the valves for the controlled in- and outflows. The dynamics is described by linear differential equations. Initially both valves are closed, and for the filling levels x_1 and x_2 of the first respectively the second tank it holds that $x_1 \in [1.5, 2.5]$ and $x_2 = 1$. More details about the model can be found at [8].

Figure 2 shows the reachability analysis results of SpaceEx/STC (max. iterations: 50, local time horizon: 5, flowpipe tolerance: 0.1) and Flow* (jump depth: 2, local time horizon: 5, time-step size: 0.01) on this benchmark. The initial set is located in the upper right of each diagram. As we can see, the results on this benchmark are comparable, though Flow* gave a bit more precise results.

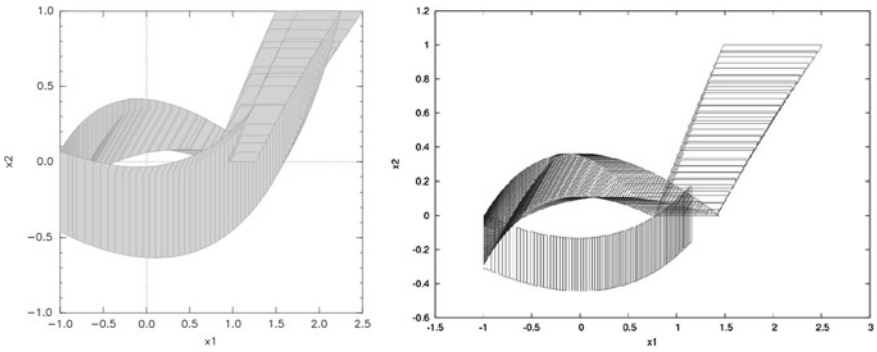

Fig. 2. SpaceEx/STC (left) and Flow*(right) results for the two-tanks benchmark.

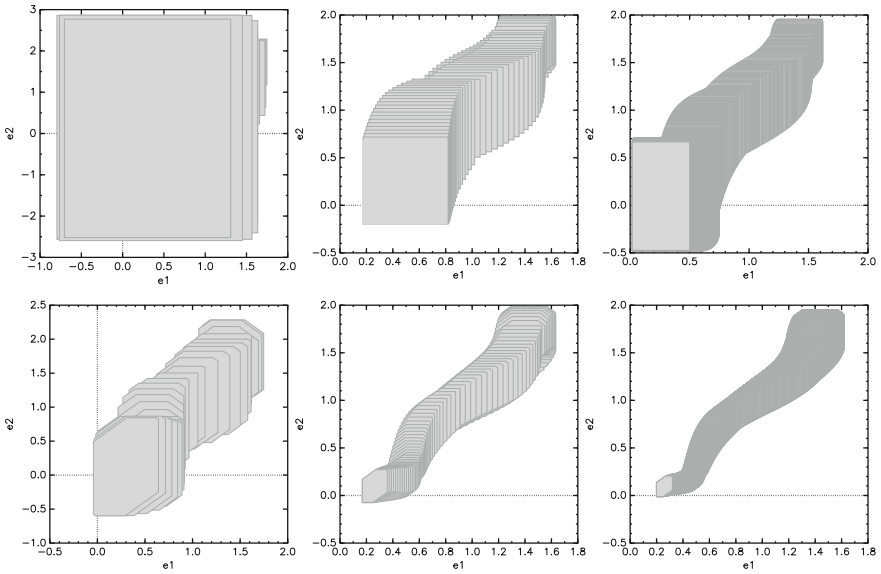

Fig. 3. SpaceEx/LGG results for the three-vehicle platoon benchmark.

Three-vehicle platoon [7,8]: The system consists of a human-driven vehicle and three communicating vehicles following it in a platoon. Two locations are used to model functioning and disrupted communication, respectively. The flows in the locations are described by linear differential equations (without uncertainties). For more details on the model and the initial states see [8].

Some reachability analysis results for this benchmark using SpaceEx/LGG are shown in Fig. 3, using local time horizon 12 and max. iterations 5. The results show the distance e_1 between the human-controlled vehicle and the first following platoon vehicle, and the distance e_2 between the first and the second following platoon vehicles, which are initially $e_1, e_2 \in [0.9, 1.1]$ units. The three results in the first row in Fig. 3 are created using boxes while for the results in the second row octagons are chosen. In the analysis we use time-step sizes 0.3, 0.1 and 0.01 s (from left to right in both rows). We can observe that in general boxes over-approximate more strongly, whereas octagons give more precise results. As expected, for both representations the error can in general be reduced by reducing the time-step size. The error reduction comes at the cost of longer running times: for boxes the computations needed 0.05, 0.1 resp. 0.14 s, whereas in the case of octagons the computational effort has grown from 2.97 over 9.5 to 42.4 s. Note that the plots in the left column use different scales.

Furthermore, an interesting effect can be observed in the top-right plot: the reachable set for precision 0.01 seems to be larger as for time-step size 0.1. However, this fact is *not* due to stronger over-approximation. In contrast to Flow*, where the user specifies a jump depth (i.e., all paths with this number of jumps are explored), SpaceEx takes the total number of jump successor computations

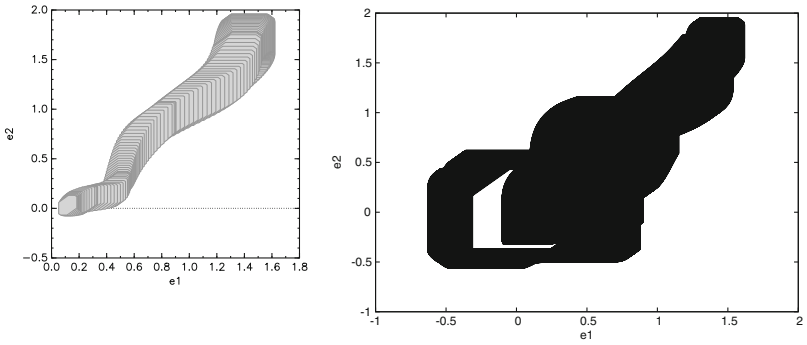

Fig. 4. SpaceEx/LGG (left) and Flow* (right) results for the three-vehicle platoon benchmark.

in the analysis (in this example 5) as input parameter. Some jumps, which were enabled due to over-approximation with larger step-sizes, are not enabled any more with step size 0.01. Thus the larger reachable set is due to the fact that with the finer precision longer paths can be explored.

Some more results for Flow* are presented in Fig. 4 (right) in comparison to the octagon setting in SpaceEx/LGG (left). Both tools used a time-step size of 0.01 s and local time horizon 12, max. iterations was set to 5 in SpaceEx, and jump depth to 5 in Flow*. The computed reachable set is clearly larger for Flow* than for SpaceEx. This has two reasons. Firstly, in Flow* all paths with 5 jumps are considered, in contrast to SpaceEx computing a total of 5 jumps. Secondly, the intersection computations for jumps lead to stronger over-approximations in Flow*, which accumulate in further computation steps. This case illustrates that sometimes tools, which were designed for more expressive model classes (Flow* was designed for non-linear dynamics), work less optimal on simpler models (here linear dynamics).

Navigation [16]: This benchmark models the movement of an object in a two-dimensional plane. In our case the plane is subdivided into a 3×3 grid structure, whereas other configurations with more cells are also possible. The linear dynamics inside each cell is determined by its position. The corresponding hybrid automaton models each cell by an own location. Jumps between the locations are enabled for all states at the boundaries between the cells; these jumps modify only the location but no other state components. Therefore, this hybrid automaton model exhibits Zeno behaviour, because such switches between the cells can be done back-and-forth infinitely often, without letting time elapse.

This Zeno behaviour can be observed on the reachability analysis results of SpaceEx/LGG (max. iterations: 5, local time horizon: 2, time-step size: 0.001) shown in Fig. 5 (top left). In the zoomed part (top right) the effects of the Zeno behaviour are exposed.

The two plots in the bottom of Fig. 5 show some Flow* results (jump depth: 1 for bottom left and 2 for bottom right, local time horizon: 2, time-step size: 0.1).

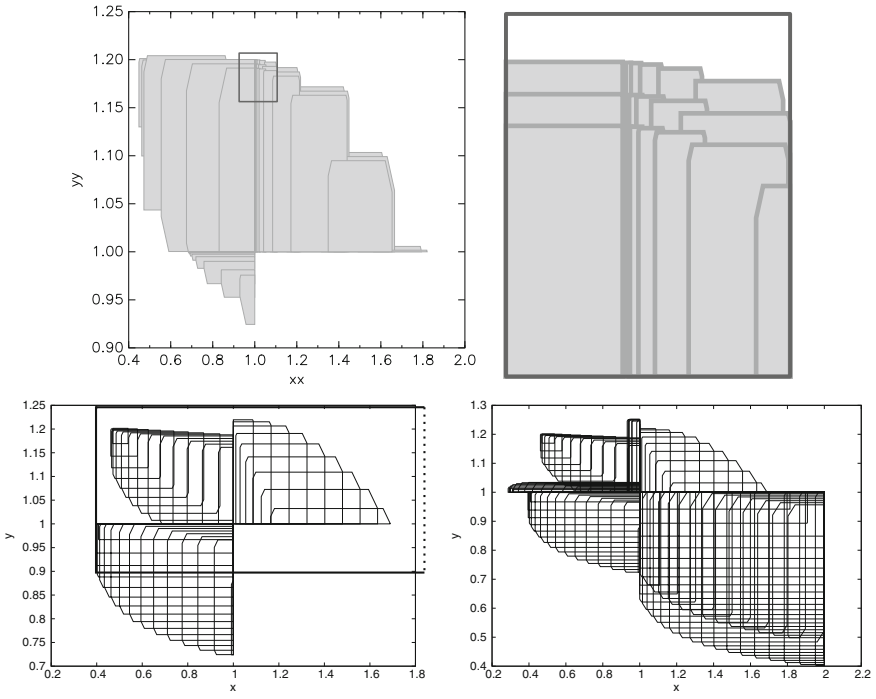

Fig. 5. SpaceEx/LGG results for the navigation benchmark (above), Flow* results below.

We have chosen a larger time-step size for Flow*, in order to make the same effect of the Zeno behaviour visible in the plots, however, a similar reachable set is computed also for the smaller time-step size 0.001. For comparability, in the bottom-left plot, we indicate the SpaceEx domain $[0.4, 2.0] \times [0.9, 1.25]$ of the plot above by a rectangle.

5 Further Challenges

The previously described tools cope with a wide range of models and offer powerful technologies for reachability analysis. Nevertheless, there are several challenges still to be addressed in order to increase the applicability and usability of the tools. In this section we discuss some of these challenges.

State set representation: The choice of the state set representation is always a trade-off between computational complexity and precision. There are many different representations usable for the analysis of a hybrid system. Boxes and polytopes are frequently used, also support functions and zonotopes are prominent for models with linear ODEs, whereas Taylor models can be used also for non-linear ODEs. However, none of the representations offers an optimal solution, since they

have individual strengths and weaknesses, mainly in the representation size and in the efficiency of certain operations (e.g., union, intersections, Minkowski sum, linear transformation, etc.) needed during the reachability analysis. Although several tools use conversions between representations for certain computations, context-sensitive approaches are still missing. For example, the representation could be adopted to the form of the ODEs in different locations. Also an automated dynamic conversion to reach an optimal trade-off between precision and efficiency during computation using an iterative refinement technique is not yet supported. Furthermore, there is rare support for non-convex representations. Last but not least, most representations are over-approximative, and therefore applicable for safety verification. However, for proving unsafety, novel under-approximative computations would be of help.

Precision: Precision is a crucial component during analysis. For systems, where the distance between the reachable and the unsafe states is small, the used precision can be crucial for the outcome of the reachability analysis. If the outcome is inconclusive (the over-approximation intersects with the unsafe state set), currently the only solution is to re-start the analysis from scratch with new parameters which lead to an error reduction (e.g., reduction of the time-step size in the flowpipe construction). However, since higher precision comes with longer running times, the new parameters must be chosen carefully by the user. An automatic adaptation of the parameters would be not only more user-friendly, but could also be applied dynamically to refine the search only along those paths which led to an intersection with the unsafe state set, instead of executing the whole analysis with high precision.

Fixed-point recognition: Recognising fixed-points in the reachability analysis, i.e., when the whole reachable state set of a hybrid system is already checked for safety, enables the solution of the unbounded reachability problem. However, in order to detect fixed-points, a huge number of state sets need to be stored, and successor sets must be tested for inclusion. As this comes at high costs, current tools use only heuristic checks for fixed-points. A more systematic check would require a highly efficient storage of state sets and fast operations on them - a possible approach could use memory-efficient under-approximations in a representation with fast inclusion and intersection computations (e.g. boxes).

Large uncertainties: Uncertainties can be included in the models when, e.g., some coefficients of the dynamics cannot be fixed precisely, or in the presence of time-varying external inputs like natural forces or users. Though systems with bounded uncertainties can be verified, models with large uncertainties are one more challenge in the verification of hybrid systems. Each uncertainty introduces a bloating factor which is carried onwards and even aggregated during the computation of the reachable set. Although a few approaches were proposed to overcome these limitations (see, e.g., [35]), most tools have problems to find conclusive answers for models with large uncertainties.

Zeno behaviour: Whenever it is possible to execute an infinite number of jumps in a finite amount of time, we observe Zeno behaviour (see the navigation benchmark example and Fig. 5). Naturally, no real system exhibits Zeno behaviour. However, it is hard to avoid Zeno paths in modelling. In [3] the authors distinguish between chattering Zeno (infinite jump sequences with zero dwell time) and genuine Zeno (infinite jump sequences with nonzero dwell time in-between converging to zero) behaviour.

Examples for chattering Zeno behaviours can be found in switching systems, where the state space is divided into grids, each grid having its own dynamics, modelled by an own location. Switching between different grids does not modify the continuous state and is always possible whenever the current state lies at the boundary between two grids. Therefore, infinite back-and-forth switching on boundaries can happen in such models, causing a problem for reachability analysis if the reach-set approximation is not idempotent: Even if no new states are reached, successor states in a sequence of jumps may grow and even diverge as the approximation errors accumulate. If the reach-set computation is exact (such as in HyTech or PHAVer), chattering Zeno has no particularly adverse effect (it may increase the number of image computations necessary to reach a fixed-point).

Genuine Zeno can be problematic for any computation that follows the execution of the system, because any finite number of successor computations may not be able to cover all reachable states. Over-approximations may resolve the problem if they cover the limit points of the sequence. This can be achieved automatically with widening operators [14]; here the difficulty lies in keeping the over-approximation reasonably small [30].

Non-convex invariants: Most tools require that the invariants of the locations are convex sets, mainly for representation reasons. However, similarly to programs which might have disjunctions in loop conditions, also non-convex invariants appear in hybrid system applications. Though one can apply model transformation to eliminate non-convex invariants by splitting the non-convex set into convex subsets and introducing a new location for each convex subset, with this approach the models are extended with Zeno behaviour, hardening their analysis (see Fig. 6). An efficient analysis without such model transformations could be enabled for example by non-convex state set representation techniques.

Urgent transitions: Invariants are one possibility in modelling to force the control to move from one mode to another. Another possibility are urgent transitions, which must be taken as soon as they are enabled. Urgent transitions have the advantage that they make the reason for the mode change more visible (observable), and therefore they are sometimes preferred instead of the usage of invariants. However, most tools do not support urgent transitions, though their analysis would even reduce the computation effort: both the expensive computations of intersections with invariants as well as the computation of flowpipes from those state sets which are included in the guard of an outgoing urgent transition become superfluous.

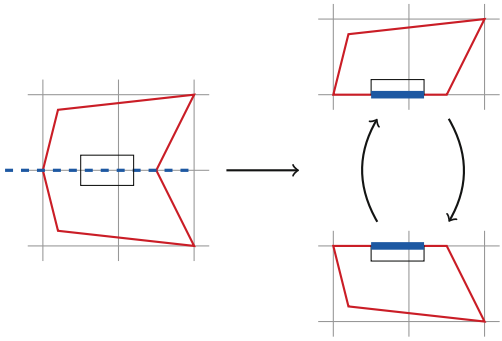

Fig. 6. The split of a location with a non-convex invariant (left) into two locations with convex invariants (right) might introduce Zeno behaviour.

Compositionality: Large systems are usually modelled compositionally as a set of modules running concurrently. Most available tools build the parallel composition of the modules to get a non-compositional model, which can be subsequently analysed. However, the composition results in high-dimensional systems, which pose challenges for the analysis. Compositional analysis techniques would be advantageous, but there is no straightforward way to extend the available techniques to support compositionality. As assume-guarantee methods proved to be useful in program verification, it might also be a promising option in hybrid systems reachability analysis. But when we aim at push-button approaches, suitable assumption-commitment specifications should be derived automatically. Another possibility could be to analyse the concurrent modules simultaneously and communicate between the concurrent analysers on synchronisation-relevant computations using, e.g., partial order reduction techniques.

Counterexamples: Although a few tools, like for example KeYmaera, can provide counterexamples for unsafe models, most tools do not have this functionality. However, counterexamples are extremely important and provide valuable information for system developers to correct unsafe designs. Furthermore, counterexamples play an important role in counterexample-guided abstraction refinement (CEGAR).

CEGAR: Frequently used in various other research areas, counterexample-guided abstraction refinement is not yet established in the field of hybrid systems. Utilising a relaxed version of the problem can introduce a significant speed up in verification. In case the verification fails, a counterexample path is used to refine relevant components of the model.

Parallelisation: Regarding the efficiency of the reachability analysis of hybrid systems, the current main focus lies on improving the efficiency of sequential algorithms. Approaches for parallelisation are rare and not yet well understood. However, the exploitation of multi-core hardware systems could help to improve the scalability and the applicability of available technologies to large-scale systems.

Modelling language expressiveness: To make hybrid automata as a modelling language more attractive and usable for a wider range of applications, also further extensions regarding expressiveness should be considered. For example, *cyber-physical systems* are distributed hybrid systems, where additionally to discrete and dynamic aspects, also communication plays an important role. Spatio-temporal hybrid automata [37] are a possible extension in this direction, supporting the modelling of communication and other spatial aspects.

Another relevant aspect is *randomised* behaviour, which can affect either the dynamics of a system via stochastic differential equations [9] or the discrete behaviour via probabilistic transitions [38]. The later can involve probabilistic properties regarding the choice between enabled transitions as well as the when to take an enabled transition. A pioneer tool in this area is ProHVer [34], which implements analysis algorithms using a transformation of probabilistic hybrid automata to hybrid automata without probabilistic components.

6 Conclusion

In this paper we gave a brief introduction to state-of-the-art tools for the reachability analysis of hybrid systems, and discussed current challenges for further research. Despite great achievements, there is still a need for efforts to increase applicability and scalability. Standardisation, competitions, and the strengthening of the functionality and the efficiency of techniques and tools may increase visibility and intensify the developments in this relevant research area.

References

1. Althoff, M., Dolan, J.M.: Online verification of automated road vehicles using reachability analysis. IEEE Trans. Robot. **30**(4), 903–918 (2014)
2. Althoff, M., Frehse, G.: Benchmarks of the workshop on applied verification of continuous and hybrid systems (ARCH) (2014). http://cps-vo.org/group/ARCH/benchmarks
3. Ames, A.D., Sastry, S.: Characterization of Zeno behavior in hybrid systems using homological methods. In: Proceedings of ACC 2005, pp. 1160–1165. IEEE Computer Society Press (2005)
4. Bak, S., Bogomolov, S., Johnson, T.T.: HYST: a source transformation and translation tool for hybrid automaton models. In: Proceedings of HSCC 2015, pp. 128–133. ACM (2015)
5. Barrett, C., Stump, A., Tinelli, C.: The satisfiability modulo theories library (SMT-LIB) (2010). http://www.SMT-LIB.org
6. van Beek, D.A., Fokkink, W.J., Hendriks, D., Hofkamp, A., Markovski, J., van de Mortel-Fronczak, J.M., Reniers, M.A.: CIF 3: model-based engineering of supervisory controllers. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014 (ETAPS). LNCS, vol. 8413, pp. 575–580. Springer, Heidelberg (2014)
7. Ben Makhlouf, I., Diab, H., Kowalewski, S.: Safety verification of a controlled cooperative platoon under loss of communication using zonotopes. In: Proceedings of ADHS 2012, pp. 333–338. IFAC-PapersOnLine (2012)

8. Benchmarks of continuous and hybrid systems. http://ths.rwth-aachen.de/research/hypro/benchmarks-of-continuous-and-hybrid-systems/
9. Bujorianu, M., Lygeros, J.: Toward a general theory of stochastic hybrid systems. In: Blom, H.A.P., Lygeros, J. (eds.) Stochastic Hybrid Systems. LNCIS, vol. 337, pp. 3–30. Springer, Heidelberg (2006)
10. Chen, X.: Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models. Ph.D. thesis, RWTH Aachen University, Germany (2015)
11. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Taylor model flowpipe construction for non-linear hybrid systems. In: Proceedings of RTSS 2012, pp. 183–192. IEEE Computer Society Press (2012)
12. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013)
13. Collins, P., Bresolin, D., Geretti, L., Villa, T.: Computing the evolution of hybrid systems using rigorous function calculus. In: Proceedings of ADHS 2012, pp. 284–290. IFAC-PapersOnLine (2012)
14. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Proceedings of SIGACT-SIGPLAN, pp. 84–96. ACM (1978)
15. Eggers, A.: Direct Handling of Ordinary Differential Equations in Constraint-solving-based Analysis of Hybrid Systems. Ph.D. thesis, Universität Oldenburg, Germany (2014)
16. Fehnker, A., Ivančić, F.: Benchmarks for hybrid systems verification. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 326–341. Springer, Heidelberg (2004)
17. Fränzle, M., Herde, C., Ratschan, S., Schubert, T., Teige, T.: Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. J. Satisf. Boolean Model. Comput. **1**, 209–236 (2007)
18. Frehse, G., Kateja, R., Le Guernic, C.: Flowpipe approximation and clustering in space-time. In: Proceedings of HSCC 2013, pp. 203–212. ACM (2013)
19. Frehse, G.: Reachability of hybrid systems in space-time. In: Proceedings of EMSOFT 2015. ACM (2015)
20. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011)
21. Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS, vol. 9195, pp. 527–538. Springer, Heidelberg (2015)
22. Gao, S., Kong, S., Clarke, E.M.: dReal: an SMT Solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS, vol. 7898, pp. 208–214. Springer, Heidelberg (2013)
23. Henzinger, T.: The theory of hybrid automata. In: Proceedings of LICS 1996, pp. 278–292. IEEE Computer Society Press (1996)
24. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**(1), 94–124 (1998)
25. HyCreate: a tool for overapproximating reachability of hybrid automata. http://stanleybak.com/projects/hycreate/hycreate.html
26. Kong, S., Gao, S., Chen, W., Clarke, E.: dReach: δ-reachability analysis for hybrid systems. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 200–205. Springer, Heidelberg (2015)

27. Le Guernic, C.: Reachability analysis of hybrid systems with linear continuous dynamics. Ph.D. thesis, Université Joseph-Fourier-Grenoble I, France (2009)
28. Le Guernic, C., Girard, A.: Reachability analysis of linear systems using support functions. Nonlinear Anal. Hybrid Syst. **4**(2), 250–262 (2010)
29. Lygeros, J.: Lecture notes on hybrid systems. In: Notes for the ENSIETA 2004 Workshop (2004)
30. Maka, H., Frehse, G., Krogh, B.H.: Polyhedral domains and widening for verification of numerical programs. In: NSV-II: Second International Workshop on Numerical Software Verification (2009)
31. Nedialkov, N.S.: VNODE-LP - A validated solver for initial value problems in ordinary differential equations. Technical Report CAS-06-06-NN, Department of Computing and Software, McMaster University, Ontario (2006)
32. Platzer, A., Quesel, J.-D.: KeYmaera: a hybrid theorem prover for hybrid systems (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 171–178. Springer, Heidelberg (2008)
33. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reason. **41**(2), 143–189 (2008)
34. ProHVer: Safety verification for probabilistic hybrid systems. http://depend.cs.uni-sb.de/tools/prohver/
35. Ramdani, N., Meslem, N., Candau, Y.: A hybrid bounding method for computing an over-approximation for the reachable set of uncertain nonlinear systems. IEEE Trans. Autom. Control **54**(10), 2352–2364 (2009)
36. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation based abstraction refinement. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 573–589. Springer, Heidelberg (2005)
37. Shao, Z., Liu, J.: Spatio-temporal hybrid automata for cyber-physical systems. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) ICTAC 2013. LNCS, vol. 8049, pp. 337–354. Springer, Heidelberg (2013)
38. Sproston, J.: Decidable model checking of probabilistic hybrid automata. In: Joseph, M. (ed.) FTRTFT 2000. LNCS, vol. 1926, p. 31. Springer, Heidelberg (2000)