Softw Syst Model (2015) 14:121–148
DOI 10.1007/s10270-012-0295-3

SPECIAL SECTION PAPER

Improving the SAT modulo ODE approach to hybrid systems analysis by combining different enclosure methods

Andreas Eggers · Nacim Ramdani · Nedialko S. Nedialkov · Martin Fränzle

Received: 22 March 2012 / Revised: 29 September 2012 / Accepted: 24 October 2012 / Published online: 16 November 2012

Abstract Aiming at automatic verification and analysis techniques for hybrid discrete-continuous systems, we present a novel combination of enclosure methods for ordinary differential equations (ODEs) with the iSAT solver for large Boolean combinations of arithmetic constraints. Improving on our previous work, the contribution of this paper lies in combining iSAT with VNODE-LP, as a state-of-the-art interval solver for ODEs, and with bracketing systems, which exploit monotonicity properties allowing to find enclosures for problems that VNODE-LP alone cannot enclose tightly. We apply the combined iSAT-ODE solver to the analysis of a variety of non-linear hybrid systems by solving predicative encodings of reachability properties and of an inductive stability argument, and evaluate the impact of the different enclosure methods, decision heuristics and their combination. Our experiments include classic benchmarks from the literature, as well as a newly-designed conveyor belt system that combines hybrid behavior of parallel components, a slip-stick friction model with non-linear dynamics and flow invariants and several dimensions of parameterization. In the paper, we also present and evaluate an extension of VNODE-LP tailored to its use as a deduction mechanism within iSAT-ODE, to allow fast re-evaluations of enclosures over arbitrary subranges of the analyzed time span.

Keywords Analysis of hybrid discrete-continuous systems · Satisfiability modulo theories · Enclosure methods for ODEs · Bracketing systems

A preliminary version of this paper appeared in [6]. This work has been supported by the German Research Council DFG within SFB/TR 14 "http://www.avacs.org", by the French National Research Agency under contract ANR 2011 INS 006 04 "http://projects.laas.fr/ANR-MAGIC-SPS", and by the Natural Sciences and Engineering Research Council of Canada.

A. Eggers (✉) · M. Fränzle
Department of Computing Science, Carl von Ossietzky Universität, 26111 Oldenburg, Germany
e-mail: andreas.eggers@informatik.uni-oldenburg.de

M. Fränzle
e-mail: fraenzle@informatik.uni-oldenburg.de

N. Ramdani
Université d'Orléans, PRISME, 63 av. de Lattre de Tassigny, 18020 Bourges, France
e-mail: nacim.ramdani@univ-orleans.fr

N. S. Nedialkov
McMaster University, Hamilton, ON, Canada
e-mail: nedialk@mcmaster.ca

1 Introduction

Model-based design is an industrially accepted approach to system development, exploiting abstract models of the device under design for early detection and correction of design errors, as well as for optimization of non-functional properties. Depending on the application domain, the models underlying model-based design and analysis come in various flavors. In traditional hardware and software design, they are discrete entities like (syntactically sugared) automata. When timely reaction to stimuli is an issue, like in communication protocols, they take the form of timed automata or annotated UML statecharts. For performance analysis of, e.g., client-server architectures, queueing models or Markov processes may be adequate. Embedded systems may call for modeling and analysis of their joint dynamics with the environment, leading to hybrid system models.

Hybrid systems seamlessly integrate traditional models for discrete behavior of computational devices with classical models for dynamical systems based on differential and algebraic equations, thus being able to represent the

feedback dynamics of embedded computers in a physical, chemical, or biological environment. With their mathematically precise semantics, hybrid systems form an appropriate formal model for the design and analysis of embedded control systems, where typical analysis goals are safety (can the system ever leave the set of safe states) and progress (is it guaranteed to converge to desired operational states) properties. As hybrid systems tend to be open systems, featuring uncontrolled inputs from the environment (e.g., user intervention) or uncontrollable disturbances due to noise or fluctuations of environmental or system-internal parameters, analysis of individual system trajectories by simulation or related computational methods is insufficient. Instead, an infinite set of possible system behaviors arising from the (in general) uncountably infinite set of possible environmental stimuli, entering through the open inputs, has to be analyzed. This calls for automatic verification techniques characterizing and exhaustively analyzing the set of all possible system behaviors or, if an exact characterization of the set of possible behaviors is impossible, characterizing and analyzing a superset of the possible system behaviors. The latter is called overapproximation and, in case all trajectories in the overapproximated behavior set satisfy the desired safety and progress properties, leads to certificates of system correctness that apply not only to the approximate model, but to the precise model as well.

A primary reason for adopting overapproximation is that a precise model, or a practical engineering model at hand, incorporates elements that no verification tool can handle in combination. This is often the case for hybrid system models due to their rich vocabulary. Analysis of such models can only commence after a chain of approximation steps, some of which can be achieved automatically, others—the majority in practice—requiring manual reformulation of the model under inspection. Each of these approximations may cause a loss of precision in the model, e.g., when capturing nonlinear behavior by a linear model, making the analysis less likely to succeed with a positive certificate as outcome. At the same time, as these approximations often have to be done manually, they require extremely skilled staff, are tedious and have to be repeated when the original model changes. We are therefore convinced that it is highly desirable to develop tools that can handle as rich dynamics as possible and hence allow model checking of hybrid systems in a direct way. In this article, we do not present a comprehensive tool that achieves this goal, but we show that our improvement of satisfiability (SAT) modulo ODE solving is a promising step into this direction, though still of academic nature in the size of problems solvable.

The underlying idea of hybrid system analysis by satisfiability modulo ODE solving is to offer a constraint language, plus the corresponding solvers, featuring as its atomic constraints exactly the equations and inequalities arising in hybrid-system models, especially algebraic constraints between variables and non-linear ODEs. With such an expressive constraint language, predicative encoding of hybrid system dynamics becomes straightforward, rendering intricate encodings and approximations superfluous.

Starting from a predicative encoding of a hybrid system, the task of a solver is to prove the absence of or search for a satisfying valuation of the variables, which encode snapshots of the system's state at points in time, connected by the transition relation that encodes the behavior of the system. In the case of bounded model checking (BMC), satisfying valuations represent trajectories of the modeled system, starting from an initial state, performing a bounded number of transitions (jumps and flows) and finally leading to a target state satisfying a property of interest. The basic principle of SAT modulo ODE solving is to handle directly ODEs as part of a constraint system by evaluating their consistency under the current partial assignment the solver is investigating, and learning implied facts for future search.

ODE enclosures as propagation mechanisms have been applied previously in Constraint Programming [9] for conjunctive Constraint Satisfaction Problems as well as by Ishii et al. [11] in a traditional Satisfiability Modulo Theories (SMT) scheme. In contrast to such an integration (i.e., a SAT solver selecting which theory atoms shall be satisfied, interleaved with theory solvers evaluating this conjunction of atoms), the iSAT [8] algorithm performs a search by splitting intervals and hence indirectly ruling out those atoms that become inconsistent under this valuation and thus deducing that other arithmetic constraints must be satisfied for satisfaction of the entire formula. These constraints then participate in the search by means of interval constraint propagation (ICP): as they have to be satisfied, interval valuations for their variables can be narrowed by pruning off subintervals that cannot contain a solution. Such ICP deductions are well known for algebraic constraints and narrow the search space very effectively.

Reasoning about ODEs can be directly integrated into this framework [5] using methods for safe interval enclosures of solutions of ODEs. These methods compute an interval cover for the states reachable from an interval box of initial states. Since their effectiveness in narrowing the overall search space of the constraint solver depends on the tightness of the enclosures provided by these methods, we have reconsidered the tools used for generating such enclosures, now incorporating the ODE solver VNODE-LP [17] and combining it with a second layer of reasoning about ODEs, which is only applicable under certain side-conditions, but may yield tighter enclosures. This additional layer generates bracketing systems [20] that reduce the problem of computing the image of a set of initial states to one of computing bounding trajectories. The core idea is to introduce an upper- and a lower-bound variable for each of the original dimensions

of the ODE system. If the signs of the partial derivatives of the ODEs meet certain criteria, a new ODE system of twice the original dimensionality is derived for these bracketing variables. Similarly, the interval boundaries of the starting point (*prebox*) for the enclosure problem are inserted into the corresponding boundary variables, replacing the original prebox with a point-valued one. The resulting bracketing system is then handed to VNODE-LP, hoping for the point-valued prebox to yield a tighter enclosure than VNODE-LP would produce on the original enclosure problem. Section 4 describes this approach in more detail.

Contributions and structure of this paper We describe the resulting iSAT-ODE algorithm and evaluate the tool on a number of different hybrid systems, comparing our results with the ones published by Ishii et al. for the `hydlogic` tool [11], which is the technologically most similar approach, and also aiming at the analysis of nonlinear hybrid systems without explicit computation of the set of reachable states.

The exposition starts with an overview of the iSAT algorithm and its interplay with ODE constraints in Sect. 2. Thereafter, Sect. 3 describes the VNODE-LP solver and details an accelerated evaluation of enclosures by extracting and storing the Taylor coefficients computed by VNODE-LP. We use these to enclose trajectories at time points and over interval subranges of the covered time span to improve the deduction mechanism compared with our previous work in [6].

Section 4 explains the bracketing systems approach, and Sect. 5 describes the combination of these enclosure methods and the newly introduced handling of flow invariants, which is another contribution of this article. We have added handling of axis-aligned flow invariants to the deduction mechanism to detect when enclosures leave a box-shaped region of the state space, and hence no longer contain any trajectories of the modeled hybrid system, whose mode invariants must never be violated by any admissible run. These flow invariant constraints thus allow us to express directly mode invariants of hybrid systems within a predicative encoding.

Section 6 discusses deducing trajectory directions, which are helpful in the refutation of trajectories that have no time-progress—a property needed in the context of some of our benchmarks.

Finally, in Sect. 7, we evaluate iSAT-ODE on a variety of nonlinear hybrid systems and different analysis goals. We compare the impact of different enclosure methods and interval splitting heuristics on deep unwindings of BMC problems, as traditionally covered by SMT methods, as well as on a novel temporal induction scheme able to prove a form of stability of hybrid systems.

We also introduce a newly designed case study of a conveyor belt system as a benchmark, which allows a high degree of parametrization and includes a hybrid slip-stick friction model with non-linear dynamics and flow invariants for parallel components. We show for this example that by introducing an auxiliary dimension to the ODE system and computing one derivative, a non-linear flow invariant can easily be turned into an axis-aligned flow invariant, that can be handled by iSAT-ODE.

Using hybrid system benchmarks from [11], we also perform a comparison with the `hydlogic` tool from [11], which Ishii et al. have already compared in that paper against other approaches for the analysis of nonlinear hybrid systems.

Our conclusions are presented in Sect. 8.

2 The iSAT Algorithm for SAT modulo ODE

In this section, we overview briefly the basic iSAT algorithm (for details cf. [8]) and focus on aspects relevant for the interaction with the ODE enclosure mechanisms that we present in the sections that follow.

Problem statement Let Φ be a quantifier-free Boolean combination of arithmetic constraints over real-, integer- and Boolean-valued variables with bounded domains, simple bounds, ODE constraints over real variables and flow invariants with the following properties:

- arithmetic constraints over variables x, y, and z are of the form $x \sim \circ(y, z)$ or $x \sim \circ(y)$, where $\sim$ is a relational operator from $\{<, \leq, =, \geq, >\}$, and $\circ$ is a total unary or binary operator from $\{+, -, \cdot, \sin, \cos, \mathrm{pow}_{\mathbb{N}}, \exp, \min, \max\}$;
- simple bounds are of the form $x \sim c$ with $\sim$ as above a relational operator, x a variable and c a constant;
- ODE constraints are time invariant and given by $\dot{x}_i = dx_i/dt = f(x_1, \ldots, x_n)$ with all occurring variables x_i themselves again being defined by ODE constraints and f being a function composed of $\{+, -, \cdot, /, \mathrm{pow}_{\mathbb{N}}, \exp, \ln, \sqrt{}, \sin, \cos\}$; and
- flow invariant constraints are of the form $x(\mathrm{time}) \leq c$ or $x(\mathrm{time}) \geq c$ with x being an ODE-defined variable and c being a constant.

The ODE constraints and flow invariants must occur only under an even number of negations in the formula, allowing, e.g., an implication like $m_1 \rightarrow ((\dot{x} = \sin(y)) \wedge (\dot{y} = -x))$, but forbidding, e.g., $(\dot{x} = \sin(y)) \rightarrow m_1$ to avoid subtleties in the semantics of the formula.

Additionally, Φ and the variables therein have the structure

$$\begin{aligned}\Phi = \ & \mathrm{decl}[0] \wedge \cdots \wedge \mathrm{decl}[k] \\ & \wedge\ \mathrm{init}[0] \\ & \wedge\ \mathrm{trans}[0, 1] \wedge \cdots \wedge\ \mathrm{trans}[(k-1), k] \\ & \wedge\ \mathrm{target}[k]\end{aligned}$$

```
DECL
    float [−1, 1] ax, ay, bx, by;
    float [−10, 10] x, y;
    float [0, 10] time;
    float [1, 1] delta_time;
INIT
    −− A and B on circle around (1,0).
    (ay − 0)^2 + (ax − 1)^2 = 1^2;
    (by − 0)^2 + (bx − 1)^2 = 1^2;
    −− A and B must be distinct points.
    ax != bx or ay != by;
    −− Trajectory must start in A.
    x = ax;
    y = ay;
    time = 0;
TRANS
    −− A and B stay the same.
    ax' = ax; ay' = ay;
    bx' = bx; by' = by;
    −− Trajectory.
    (d.x / d.time = y);
    (d.y / d.time = −x);
    time' = time + delta_time;
TARGET
    −− Equilateral triangle: equal
    −− distances between A, B, and X.
      (ay − by)^2 + (ax − bx)^2
     = (ax − x)^2 + (ay − y)^2;
      (ay − by)^2 + (ax − bx)^2
     = (bx − x)^2 + (by − y)^2;
```

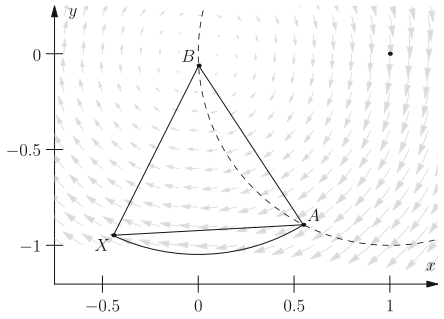


Fig. 1 Example of an iSAT-ODE input (before being automatically rewritten into the solver's internal format by its frontend). The *right graph* shows a candidate solution that has been found by the solver, illustrating a satisfying valuation of a one-step unwinding of this constraint system

arising from the k-fold unwinding of the transition system, where decl[i] is the ith instantiation of the system variables' domain bounds, init[0] is the predicative encoding of the initial state applied to the 0th variable instance, i.e., to the beginning of the trace, trans[$i, i+1$] is the application of the transition predicate to the ith and $(i+1)$th instances of the variables, e.g., instantiating $a' = a+1$ to $a[3] = a[2]+1$, and target[k] is the application of the target predicate to the last variable instance. ODE constraints and flow invariants occur only within the transition relation since they constrain the continuous flow behavior of the system.

Example To illustrate this input, Fig. 1 shows an encoding of a model from [9]. This problem can be stated as follows: find two points A and B on a circle with radius 1 around $(1, 0)$ and from the box $[-1, 1] \times [-1, 1]$, such that a trajectory of a harmonic oscillator around $(0, 0)$ with fixed temporal length (here, we choose 1), starting in A ends in a point X, forming an equilateral triangle A, B, X.

Satisfiability As usual, a valuation σ, which maps each variable to a point from its domain, satisfies Φ iff the constraints satisfied under σ satisfy the Boolean structure of Φ. Satisfiability is straightforward for simple bounds and arithmetic constraints, but requires some explanation in the case of ODE constraints.

As noted above, ODEs-describing the evolution of variables over continuous time-occur only in the transition relation, which constrains the pre-post relation between any two successive instances of variables in a trace. Semantically, a trace is a sequence of snapshots of a real-time trajectory of a hybrid system. Hence, ODE constraints describe the behavior of the system between two such snapshots, i.e., describe trajectories emerging from the pre-valuation, following the dynamics described by the ODE, and finally reaching the post-valuation. A valuation σ thus satisfies a definitionally closed system of ODE constraints (each occurring variable itself being defined by one of the component ODE constraints), iff there exists a solution trajectory starting with the pre-valuation and ending with the post-valuation after a duration equal to the temporal length of the flow, as provided by the value of a special variable `delta_time`. Flow invariants can be used to further constrain the set of admissible solutions: a valuation σ satisfies the conjunction of a definitionally closed system of ODE constraints and flow invariants over the same variables, iff there exists a solution trajectory (as above), which consists only of points that satisfy the flow invariant constraints, i.e., lie within the (potentially partially open) box described by the flow invariants.

More formally, with $\mathbb{Q}$ the set of rational numbers, for given $\vec{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$, a set of flow invariants over $\vec{x}$:

$$\{x_i(\text{time}) \sim c \mid i \in \{1, \ldots, n\}, \sim \in \{\leq, \geq\}, c \in \mathbb{Q}\}, \tag{1}$$

interpreted as the conjunction of its elements, describes either the empty set $\emptyset$ or an interval box

$$\boldsymbol{I} = \big([l_1, u_1], \ldots, [l_n, u_n]\big)^{\mathrm{T}},$$

with $l_i \leq u_i$ and $l_i, u_i \in \mathbb{Q} \cup \{-\infty, +\infty\}$ for all $i \in \{1 \ldots n\}$ (slightly abusing notation in case one of the bounds is left open to $-\infty$ or $+\infty$). With ODE constraints defining $\vec{x}$:

$$\dot{\vec{x}} = \big(f_1(\vec{x}), \ldots, f_n(\vec{x})\big)^{\mathrm{T}} = \vec{f}(\vec{x}), \tag{2}$$

for two BMC unwinding depths i and $i+1$, the instantiations of $\vec{x}$ are given by $\vec{x}[i]$ and $\vec{x}[i+1]$ and their valuations $\sigma(\vec{x}[i])$ and $\sigma(\vec{x}[i+1])$[1] together with $\tau := \sigma(\texttt{delta\_time}[i])$ satisfy (1) and (2), iff there exists a solution function $\vec{y} : [0, \tau] \to \mathrm{dom}(\vec{x})$ such that $\vec{y}(0) = \sigma(\vec{x}[i])$, $\dot{\vec{y}}(t) = \vec{f}(\vec{y}(t))$ for all $t \in [0, \tau]$, and $\vec{y}(\tau) = \sigma(\vec{x}[i+1])$, while $\vec{v}(t) \in \boldsymbol{I}$ for all $t \in [0, \tau]$, thus satisfying also all of the constraints from (1).

Flow invariants We have implemented support for axis-aligned flow invariants as introduced above, which allow restricting solution trajectories to partially open boxes. In practice, the iSAT algorithm requires all variables to have closed domains. While we do not enforce these domain

[1] For simplicity, the valuation of a vector shall be the vector of its valuations.

bounds on the continuous evolution, it will most likely always make sense to translate them directly to flow invariants, thereby also restricting the trajectories to closed boxes.

Solving The task of the solver is to find a valuation satisfying the formula or proving its unsatisfiability. Starting from an input formula like the one depicted in Fig. 1, a preprocessing step (see [8] for more details) introduces auxiliary variables to split complex arithmetic expressions into the format described above and to simplify the Boolean structure into a conjunction of clauses, which are themselves disjunctions of arithmetic atoms, simple bounds, and trigger variables representing ODE constraints and flow invariants. ODE and flow invariant constraints are stored separately and are activated whenever their respective trigger variable becomes true.

Instead of point-valued valuations, the iSAT algorithm interprets the constraints over intervals. Initially, each variable receives its whole domain as an interval valuation. Akin to DPLL-based SAT solving [3,4], the three main ingredients of the solver are deduction, decision, and conflict resolution. However, constraints cannot only be satisfied or unsatisfied for all valuations from the interval box, but also contain a mixture of points satisfying or violating a constraint. For example, consider constraint $C: x = 2 \cdot y$ under the interval valuation $x \in [0, 10]$, $y \in [3, 6]$. No point with $x \in [0, 6)$ or $y \in (5, 6]$ satisfies C, while $x \in [6, 10]$, $y \in [3, 5]$ contains points (x, y) like $(6, 3)$ satisfying and points like $(6.1, 3)$ violating C.

Clauses (disjunctions of constraints) that contain only one constraint that is potentially satisfiable under the current valuation are called *unit* and give rise to *unit propagation*. The last satisfiable constraint in a clause else containing only violated constraints must be propagated to retain a chance for satisfiability of the conjunction of all clauses. If the above constraint C were for example such a last remaining atom of a clause, then *interval constraint propagation* would allow to prune away those ranges above identified as not containing any solutions, yielding a new valuation $x \in [6, 10]$, $y \in [3, 5]$, and thus a reduced search space.

When no more propagations are possible, or all newly deduced bounds have negligible progress with respect to the old ones, a *decision* is performed by selecting heuristically a variable and splitting its interval, i.e., introducing a new upper or lower bound at its midpoint. This bound may give rise to new deductions. If all of a clause's constraints are violated under the current valuation, e.g., due to a prior propagation step, a *conflict* is encountered, which is resolved by analyzing the reasons that caused it and generating a *conflict clause* that is a disjunction of the negated reasons. This clause is added to the formula and forces at least one of the offending bounds to be chosen differently in the future, effectively removing this part of the search space for the remainder of the search.

Termination. If the solver encounters a conflict from which it cannot recover, because no undoing of decisions would resolve it, it has successfully proven *unsatisfiability*. Due to the safe overapproximations used in all propagations (e.g., outward rounding for arithmetic evaluations) and always pruning non-solutions only, this unsatisfiability result is safe. The solver terminates with *unknown*, if it encounters a box whose maximum width is below a small, user-defined threshold, and for which deduction cannot show inconsistency. This small box is a candidate solution box, which merits practical attention when encountered as a potential counter example to the safety of an engineered system. As the reported candidate solution boxes are very small, interval Newton methods may be able to verify that they contain an actual solution. While our algorithm currently does not contain such a check, Ishii et al. [11] have implemented it.

Deduction for ODE constraints Having interval valuations for the variable instances occurring in ODEs again requires lifting their original point-valued interpretation to intervals. For arithmetic constraints, we prune away only parts not containing any solutions. The very same idea applied to ODEs means that we may prune away all those points from the post-valuation that are not (forward) reachable, when starting a trajectory from any point in the pre-valuation and staying on it for any duration contained in the interval valuation of the respective `delta_time` variable. Analogously, we can safely prune away those parts of the pre-valuation for which no trajectory can reach any point in the post-valuation with any of the possible durations (backward propagation). In addition, time points t from `delta_time` can be pruned when no trajectory starting from the pre-valuation reaches any point from the post-valuation at t (cf. Fig. 2). Flow invariants represent additional constraints: only those trajectories that never leave the box $\boldsymbol{I}$ spanned by the active flow invariant constraints can be solutions of the constraint problem. This allows further pruning of the pre- and post-valuations by removing points that cannot be reached without leaving $\boldsymbol{I}$. Additionally, it permits lowering the upper bound of the `delta_time` valuation, by detecting when all trajectories have left $\boldsymbol{I}$.

The essential ingredient in the deduction for ODE constraints is thus a method to safely enclose over a temporal interval all trajectories emerging from the pre-valuation, which is an interval box. While our original integration of such an ODE enclosure mechanism into the iSAT algorithm [5] was confined to embedding a relatively weak own implementation of a Taylor-series-based safe integrator, we base our current approach on VNODE-LP [17].

Learning and caching ODE deductions are performed in strict alternation with the other deduction mechanisms. After completing Boolean and interval constraint propagation as

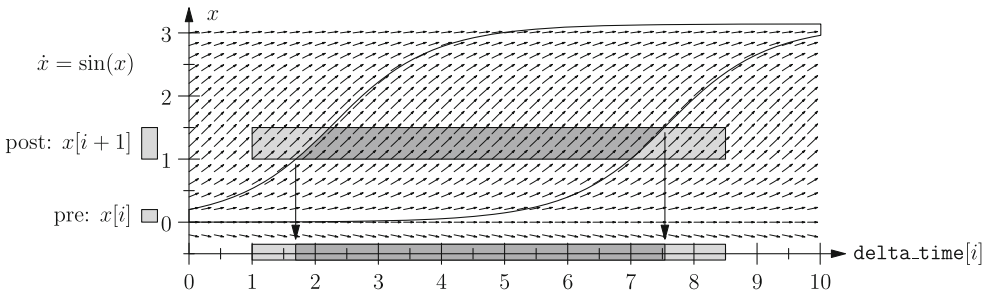


Fig. 2 An ODE deduction which allows to propagate tighter bounds for `delta_time`$[i]$

described above, iSAT's ODE solving layer uses the current valuation of the trigger variables for each instance of the transition system to select the active ODE constraints and flow invariants. This signature of activated ODEs, flow invariants, and the current interval valuation for the occurring variables together suffices to generate an enclosure. In contrast to normal deductions, whose results are stored only temporarily until they may be undone later by a backjump when recovering from a conflict, the results of ODE deductions are stored in clauses. This technique, similar to conflict clause learning, ensures that the same deduction does not have to be repeated since its results have been added persistently to the formula. Similarly to constraints replication [23], we add copies of the learned clauses for all isomorphic variable instances arising from the k-fold unwinding of the transition relation.

Before performing an ODE deduction, the algorithm checks whether the same query has been encountered before and rejects all duplicate queries. In our previous implementation (as used in [6]), the core of this check was based on projecting the current valuation to its individual dimensions and comparing the resulting intervals with the stored ones from previously answered queries. Thus, the algorithm collected per dimension those boxes that fully covered the interval to finally intersect these sets of boxes to generate a set of all previous valuations that have been computed for a superset of the current valuation. If one of these retrieved valuations was sufficiently close to the current query, the current request could be discarded since a clause has already been learned for it. We detected by profiling that this check could become the dominant part of the CPU time spent in the ODE solver (in the order of 90 %), when large numbers of valuations were stored, and attributed this to the fact that, by investigating each dimension individually, large amounts of boxes had to be considered only before being finally discarded.

Our new implementation of this coverage check is therefore based on a tree structure, which takes all dimensions into account at the same time. Given two distinct boxes $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$, we pick a point P that lies inside $\boldsymbol{B}_1$ and outside $\boldsymbol{B}_2$. This point P is thereafter associated with an inner node of the tree and used to decide for each newly added box $\boldsymbol{B}$ whether it falls into the same child tree of the node as $\boldsymbol{B}_1$ (if P is covered by $\boldsymbol{B}$) or $\boldsymbol{B}_2$ otherwise. Leaves can contain multiple boxes, thus reducing the depth of the tree. If a leaf contains more than a predefined number of boxes, it is split again by introducing of a new inner node in order to reduce search time. While we have experimented with balancing the tree and different numbers of boxes under each leaf node, we have found that using an unbalanced tree and only two boxes per leaf node to yield good runtime results for the amount of stored boxes in our examples. We provide some experimental evaluation of the caching performance in Sect. 7.

A second level of caching holds a limited number of intermediate results, which can be reused when enclosures for a subbox of the original box are requested, since interval arithmetic's monotonicity property w.r.t. set inclusion guarantees then that they are still valid (yet coarse) enclosures for the current valuation. Using a stored solver run, whenever the currently examined valuation is only slightly smaller than the original box, partially avoids re-computations. Since the bounds deduced by the ODE solver are subsequently used in interval propagations, it is very likely to encounter this kind of slightly changed query, providing this caching layer with a significant role in avoiding wasted computations.

Soundness The correctness of the core algorithm has been detailed in [8]. Since our extension to deductions for ODE constraints is restricted to the pruning of non-solutions and storing all reasons involved in these deductions explicitly in the learned clauses, the same arguments hold here, too. An essential ingredient to soundness is the use of *validated computations*, i.e., outward rounding for interval computations, interval evaluation of remainder terms to capture truncation errors for the numerical enclosure method detailed in the following section, and detection of overflows during these computations. Technically, many of these issues are delegated to libraries, in our case the MPFR [7] and filib++ [13] libraries.

3 Overview of VNODE-LP

In this section, we present an overview of VNODE-LP, Validated Numerical ODE through Literate Programming. More details can be found in [17,18].

Consider the initial-value problem (IVP)

$$\dot{\vec{x}}(t) = \vec{f}(t, \vec{x}), \quad \vec{x}(t_0) = \vec{x}_0, \quad t \in \mathbb{R}, \ \vec{x} \in \mathbb{R}^n, \tag{3}$$

where $\vec{f} : \mathbb{R} \times \mathbb{R}^n$ is sufficiently smooth (as a consequence, the code list of $\vec{f}$ should not contain, e.g., branches, abs, or min).

Denote the set of n-dimensional interval vectors by $\mathbb{IR}^n$. Given $\vec{\boldsymbol{x}}_0 \in \mathbb{IR}^n$ and $t_{\text{end}} \neq t_0 (t_{\text{end}} \in \mathbb{R})$, VNODE-LP tries to compute an $\vec{\boldsymbol{x}}_{\text{end}} \in \mathbb{IR}^n$ at t_{end} that contains the solution to (3) at t_{end} for all $\vec{x}_0 \in \vec{\boldsymbol{x}}_0$.

This solver proceeds in a one-step manner from t_0 to t_{end}, where it computes bounds at (adaptively) selected points $t_j \in (t_0, t_{\text{end}}]$. To explain an integration step, denote by $\vec{x}(t_j; t_0, \vec{x}_0)$ the solution to (3) with an initial condition $\vec{x}_0$ at t_0 and denote by $\vec{\boldsymbol{x}}_j$ an enclosure of this solution at t_j. That is,

$$\vec{x}(t_j; t_0, \vec{x}_0) \in \vec{\boldsymbol{x}}_j \quad \text{for all } \vec{x}_0 \in \vec{\boldsymbol{x}}_0.$$

On a step from t_j, VNODE-LP first tries to compute a stepsize $h_j = t_{j+1} - t_j$ and a priori bounds $\vec{\tilde{\boldsymbol{x}}}_j$ that contain the solutions originating from $\boldsymbol{x}_j$, that is,

$$\vec{x}(t; t_j, \vec{x}_j) \in \vec{\tilde{\boldsymbol{x}}}_j \quad \text{for all t} \in [\text{t}_\text{j}, \text{t}_{\text{j+1}}] \text{ and all } \vec{\text{x}}_\text{j} \in \vec{\boldsymbol{x}}_\text{j}.$$

If this stepsize becomes too small (see [18] for details), the integration stops at t_j. This usually happens if the computed bounds become too wide.

Then it finds tight bounds $\vec{\boldsymbol{x}}_{j+1}$ at t_{j+1} such that $\vec{x}(t_{j+1}; t_0, \vec{x}_0) \in \vec{\boldsymbol{x}}_{j+1}$ for all $\vec{x}_0 \in \vec{\boldsymbol{x}}_0$. For an illustration of a priori and tight bounds, see Fig. 3.

To compute these bounds, we use interval arithmetic, Taylor series expansion of the solution to (3) at each integration point, and various interval techniques; for more details, see [16,17].

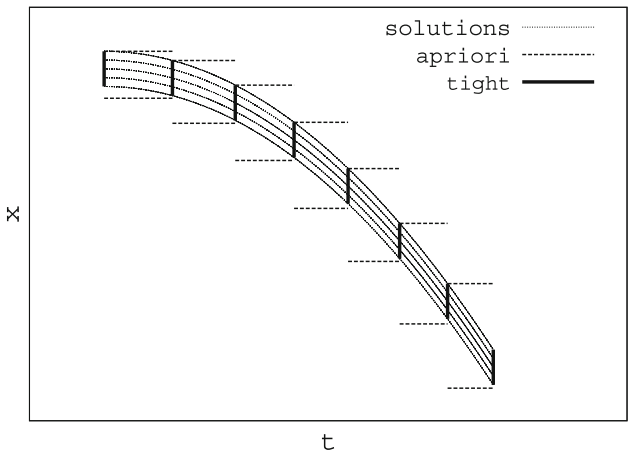


Fig. 3 A priori and tight bounds

VNODE-LP is based on interval Taylor series and the Hermite–Obreschkoff [16] methods. It is a fixed-order, variable-stepsize solver. The stepsize is varied such that an estimate of the *local excess* per unit step is below a user-specified tolerance. The default order of the series expansion is set to 20, and it can be changed by the user.

In general, VNODE-LP is suitable for computing bounds on the solution of an IVP ODE with point initial conditions or interval initial conditions with a sufficiently small width. If the initial condition set is not small enough and/or long time integration is desired, the COSY package of Berz and Makino [1] can produce tighter bounds than VNODE-LP. Alternatively, one can subdivide the initial interval vector (box) $\mathbf{y}_0$ into smaller boxes, perform integrations with them as initial conditions, and build an enclosure of the solution at t_{end}.

For the results reported in [6], on each integration step from t_j to t_{j+1}, iSAT-ODE used the a priori bounds and also computed bounds over selected subintervals of $[t_j, t_{j+1}]$, so it could use tighter than the a priori bounds over such subintervals. This was done by calling VNODE-LP to perform an integration step from t_j to $\mathbf{t}_{\text{end}} \subset [t_j, t_{j+1}]$. Although simple, this scheme is expensive as it leads to repeated integrations, whereas we could use the stored Taylor coefficients at t_j to compute such bounds at $\mathbf{t}_{\text{end}}$.

In the present work, we have built a facility to compute such tight bounds on the solution over any point in $[t_j, t_{j+1}]$ or subinterval of $[t_j, t_{j+1}]$. For this purpose, we extract the computed Taylor coefficients after each step and store them along with all representations of the enclosure, which allows us to later evaluate the Taylor series expansion of the solution (along with its variational equation and remainder enclosure) with stepsize $\mathbf{t}_{\text{end}} - t$.

This approach has three advantages: (1) it does not invalidate the solver's state, i.e., it can be done between solving steps; (2) since all solution representations are stored, the evaluation will not suffer from the additional wrapping effect incurred when reinitializing the solver from an axis-aligned box; and (3) the computational cost is significantly lower, since the saved cost for computing the a priori enclosure and the Hermite–Obreschkoff corrector step will be far higher than the additional cost of storing the coefficients and performing one scaling operation per coefficient.

When employed within iSAT-ODE, the overall acceleration depends on several additional factors, foremost the number of refinements required, which also varies with the traversal of the search space (itself being influenced by the computed enclosures). On the two-tank example, we have therefore encountered significant speedups for most instances (even of more than one order of magnitude for some), as well as slightly negative effects for few instances (see Sect. 7).

4 Using bracketing systems as enclosures

When the starting point of the IVP (3) is a wide interval vector, the enclosures returned by VNODE-LP may diverge after a few computation steps. One way to address this shortcoming, while deriving guaranteed results, is to use the bracketing approach introduced in [20,21], which relies on the classical Müller's existence theorem [12,15].

Given the IVP (3), the bracketing method analyzes the signs of the partial derivatives $\partial f_i/\partial x_l$, evaluated over the enclosure for all $t \in [t_j,\ t_{j+1}]$.

(i) Over each time interval $[t_j, t_{j+1}]$, where these signs remain constant, the method builds two dynamical systems that enclose the original uncertain dynamical system and thus bound the flow pipe between a minimal solution, i.e., a flow that is always smaller than the solution flow pipe, and a maximal solution that is always larger. Since this bracketing system involves no more uncertainty, VNODE-LP can be efficiently used for the guaranteed computation of the minimal and maximal solutions, which start as points instead of intervals. Hence, the solution enclosure of the actual IVP is enclosed between a minimal and a maximal solution, obtained as the solution of a new system of coupled ODEs.

(ii) Over each time interval $[t_j, t_{j+1}]$, where the sign of at least one partial derivative changes, we merely use VNODE-LP on the original IVP.

In our implementation of the bracketing system generation, the signs of the partial derivatives need not be analyzed over the enclosure set for all $t \in [t_j,\ t_{j+1}]$, but are only analyzed over $\vec{x}_j$, the tight enclosure at t_j. Once the bracketing systems are built, and the solution set computed over the whole time interval, these signs are then checked a posteriori: if they remain constant for all $t \in [t_j,\ t_{j+1}]$, then it is proven that the bracketing systems are valid [20]; if not, then the bracketing systems are not valid over the whole time interval. In this case the solution is enclosed using VNODE-LP on the original IVP with interval initial conditions.

Furthermore, our implementation of the bracketing approach is new. Indeed, the bracketing systems are built automatically on the fly inside iSAT-ODE. This is done through the FADBAD++ [24] automatic differentiation package, whereas previously they were built manually or using external symbolic algebra.

Figure 4 compares enclosures obtained using our implementation of the bracketing approach and the direct application of VNODE-LP. In the left graph, the dense enclosure (gray background) can be considered to represent the actual set of trajectories emerging from the prebox. VNODE-LP computes these enclosures at certain points of time, whose separation is determined by automatic stepsize control. Also shown are the a priori enclosures computed by VNODE-LP that together yield a very rough enclosure of the trajectories over the entire time span. The tight enclosures at time points for the upper bracketing variable, denoted by $\times$ and the tight enclosures for the lower bracketing variable $(+)$, while initially having exactly the same distance as the prebox's width, leave the shown range after just three integration steps. When considering the convex hull of the a priori enclosures for the upper and lower bracketing variable, we receive an enclosure over the entire time span which is far coarser than the enclosure obtained directly. In the right graph, the

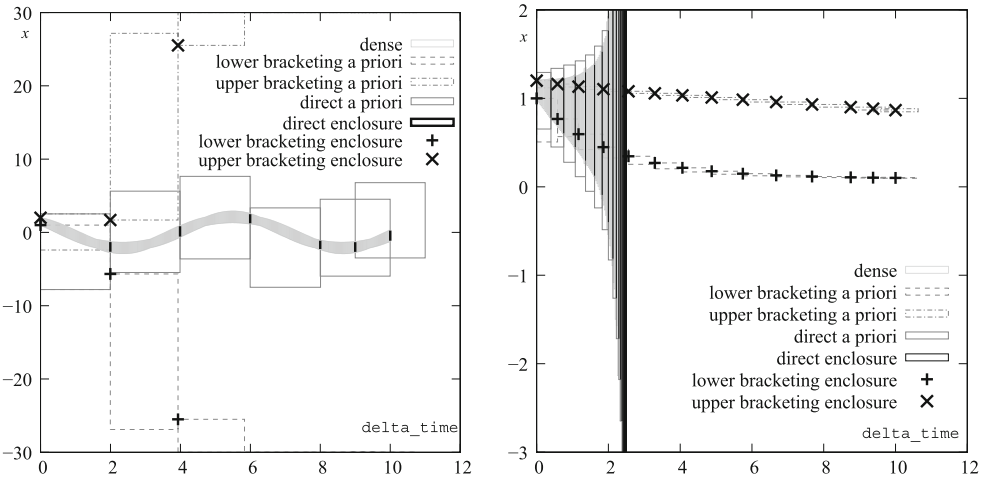


Fig. 4 Comparison of direct and bracketing enclosure. *Left* x dimension of a harmonic oscillator $\dot{x} = y, \dot{y} = -x, x(0), y(0) \in [1, 2]$. *Right* x dimension of $\dot{x} = -p_4 x - (p_1 x)/(1 + p_2 y) + p_3 y + 0.1)$, $\dot{y} = p_4 x - p_3 y$, all $\dot{p}_i = 0$, for $x(0) \in [1, 1.2]$, $y(0) \in [0.8, 1]$, $p_1 \in [0.8, 1]$, $p_2 \in [1.0, 1.2]l$, $p_3 \in [0.3, 0.5]$, and $p_4 \in [0.20, 0.25]$. Dense enclosures have been obtained by direct application of VNODE-LP with small fixed stepsize

same symbols are used for the bracketing enclosure and for the direct application of VNODE-LP. Here, the direct enclosure, including the dense one obtained by very small steps, diverges quickly, while the upper and lower bracketing variables yield an enclosure over a far longer time frame that could probably be continued for many more steps.

Clearly, both methods should be combined, as their actual performances depend on the analyzed ODE. The performance of the bracketing approach, that is, how tight the computed enclosures are, when used with a given system, may be known a priori. For monotone dynamical systems, those whose flows preserve a suitable partial ordering on states, hence on initial conditions, the computed bracketing systems are feasible instantiations of the dynamical system under study, and hence exhibit the same convergence and stability properties as the original system [21]. If the latter is convergent and stable, then so should the bracketing systems be. However, when the dynamical system is not a monotone one, the bracketing systems usually suffer from a hidden wrapping effect that provokes the derived enclosures to blow up. In spite of that, both experimental and theoretical evaluation show that, when the original system exhibits very strong convergence properties, the latter property can overrule the wrapping effect, making the bracketing approach effective [20]. Finally, the bracketing approach performs badly when the system exhibits stable orbits or oscillatory behavior. Nevertheless, we expect our implementation of bracketing systems within iSAT to simplify the thorough practical assessment of its actual performance in the future.

The new evaluation scheme using stored Taylor coefficients applies to the bracketing system enclosures as well, i.e., our solver can recompute the bracketing enclosure for arbitrary subranges of each step $[t_j, t_{j+1}]$ without reinitializing the solver.

5 Combination and flow invariant handling

Whereas we previously [6] first computed the direct and the bracketing enclosures independently of each other up to the temporal horizon (given by the upper bound of the `delta_time` valuation) and only subsequently intersected the resulting enclosures, we have now changed our implementation to interleave these two computations. The major benefit is that the enclosure generation can be stopped after detecting that one of the enclosures has an empty intersection with the box admissible by the active flow invariant constraints. Additionally, in the future, this interleaving may also allow to generate hybrid evaluation schemes of the bracketing system as described in [20], overcoming the restriction to constant signs in the Jacobian over the entire duration of the enclosure, as detailed in the previous section.

Since step sizes of each enclosure are calculated independently, direct and bracketing enclosures are not generated in strict alternation, but instead, we always perform the next enclosure step using the method whose current temporal upper bound is behind—as long as both enclosure methods are still capable of producing enclosures, i.e., they have not yet failed.

After each computation step, we check whether the generated enclosure $\vec{\boldsymbol{x}}_j$ at the nearly point-valued time t_j lies outside $\boldsymbol{I}$. If $\vec{\boldsymbol{x}}_j \cap \boldsymbol{I} = \emptyset$, the time t_j can be safely assumed to be a sufficiently large new horizon up to which all trajectories need to be enclosed.

When an enclosure $\vec{\boldsymbol{x}}[T]$ for a time span $T = [t_a, t_b]$ has been generated by both methods (or only by the remaining one if the other has reached a point from which it cannot be continued), we intersect the valid enclosures and refine this enclosure $\vec{\boldsymbol{x}}[T]$ to detect a potential earlier point of time when all trajectories have left $\boldsymbol{I}$, thus generating a tighter pruning of the `delta_time` valuation. If $\vec{\boldsymbol{x}}[T] \subseteq \boldsymbol{I}$, all trajectories lie inside the flow invariant for the entire duration T, and hence no refinement is necessary. Similarly, if $\vec{\boldsymbol{x}}[T] \cap \boldsymbol{I} = \emptyset$, no refinement is required since *all* trajectories have been safely enclosed outside the flow invariant set. If, however, $\vec{\boldsymbol{x}}[T]$ has a non-empty intersection with both $\boldsymbol{I}$ and its complement, i.e., contains points inside and outside of $\boldsymbol{I}$, we have to investigate whether potentially all trajectories have already left $\boldsymbol{I}$. Since the evaluation relies on interval computations, the enclosure may contain a considerable amount of overapproximation that can be reduced by evaluating over subranges of T.

Figure 5 illustrates the algorithm for avoiding unnecessarily thin refinements by combining a classical linear overapproximation with the tight enclosures that can be generated using the interpolant described in Sect. 3. The goal is to avoid the computational cost of computing refined enclosures for subranges, for which we can already exclude by a simple check that at least one trajectory is still inside $\boldsymbol{I}$. We first compute $\vec{\boldsymbol{x}}(t_a)$, i.e., an enclosure at the beginning of T that is the tightest possible enclosure that can be obtained using the stored Taylor coefficients of the interpolant. Additionally, we compute an interval evaluation of the right-hand side of the ODE system (2) over $\vec{\boldsymbol{x}}[T] \cap \boldsymbol{I}$ which yields an overapproximation of all possible slopes for trajectories starting in $\vec{\boldsymbol{x}}(t_a)$ at t_a before (if ever) they reach the border of $\boldsymbol{I}$. For each dimension $i \in \{1, \ldots, n\}$, we then check whether the interval $(\vec{\boldsymbol{x}}[T])_i$ covers the upper and/or lower bound of $\boldsymbol{I}_i$—the flow invariant interval in this dimension. If, e.g.. only its lower bound is covered, we pick $\max(\vec{\boldsymbol{x}}(t_a)_i)$ as the starting point of a witness trajectory with the lowest possible slope $\min(f_i(\vec{\boldsymbol{x}}[T] \cap \boldsymbol{I}))$. Even this worst-case trajectory can leave $\boldsymbol{I}$ no earlier than

$$t_{\text{leave}}, i = t_a + \frac{\inf(\boldsymbol{I}_i) - \max(\vec{\boldsymbol{x}}(t_a))}{\min(f_i(\vec{\boldsymbol{x}}[T] \cap \boldsymbol{I}))} = t_a + \frac{\text{distance}}{\text{slope}}.$$

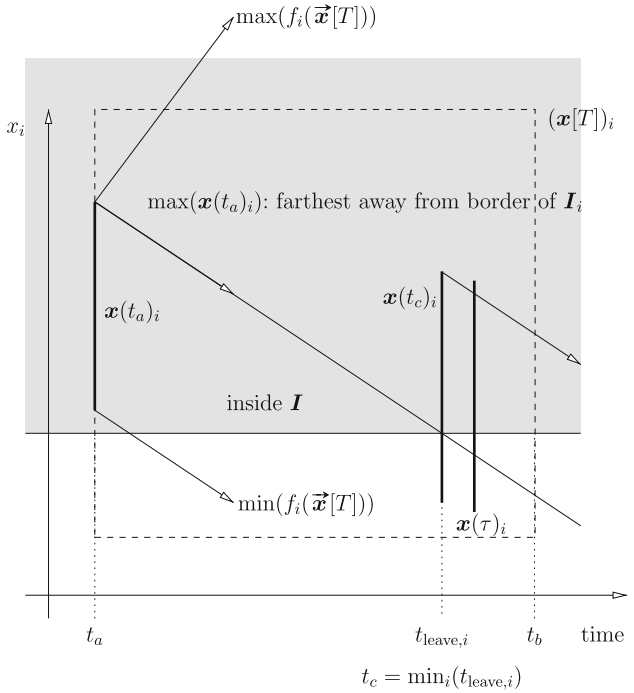


Fig. 5 Combination of the linear overapproximation with tight enclosures computed using the continuous interpolant

Analogously, we can compute t_{leave}, i if $(\vec{\boldsymbol{x}}[T])_i$ covers only the upper bound of $\boldsymbol{I}_i$ by picking the lower bound $\min(\vec{\boldsymbol{x}}(t_a))$ as a starting point and using the maximum slope. If both bounds are covered, we compute an optimal starting point from the relation between largest and smallest slope and pick the starting point from $\vec{\boldsymbol{x}}(t_a)$ that is closest to this point, thereby maximizing the value $t_{\text{leave},i}$. If $(\vec{\boldsymbol{x}}[T])_i \subseteq \boldsymbol{I}_i$, the flow invariant is not violated in this dimension, and hence $t_{\text{leave},i}$ can be safely set to T's upper bound t_b.

By computing the minimum of all these time points $t_{\text{leave,i}}$, we identify a time point $t_c = \min_{i\in\{1,\ldots,n\}}(t_{\text{leave},i})$ before which at least one witness trajectory starting in $\vec{\boldsymbol{x}}(t_a)$ stays inside the flow invariant. Therefore, we do not need to refine the enclosure for $[t_a, t_c]$. If $t_c \geq t_b$, the entire range of T has been covered. If $t_c < t_b$, the trajectories may still leave the flow invariant in the remaining time between t_c and t_b. We therefore select a lookahead point $\tau \in [t_c, t_b]$ and evaluate $\vec{\boldsymbol{x}}(\tau)$. If $\vec{\boldsymbol{x}}(\tau) \cap \boldsymbol{I} = \emptyset$, a refinement for the upper bound of `delta_time` has been found, since the enclosure of all trajectories has been shown to lie outside the flow invariant at τ. If this check fails, we try to continue the linear overapproximation that has already established that not all trajectories leave $\boldsymbol{I}$ over $[t_a, t_c]$. By computing a new starting enclosure $\vec{\boldsymbol{x}}(t_c)$, we iterate the method.

If the iteration progresses slower than a predefined minimum temporal resolution, which is set dynamically by the solver, we replace the enclosure $\vec{\boldsymbol{x}}[T]$ with two interval enclosures and recursively refine these down to the given resolution—again also trying to avoid fully recursive refinement by applying this linear overapproximation and its combined check for either having one trajectory inside $\boldsymbol{I}$ or having found all trajectories outside $\boldsymbol{I}$.

6 Deducing trajectory directions

In the case study shown in the following section, we encounter the problem of showing that a trajectory cannot stay at the point of its origin when at least an infinitesimal amount of time (`delta_time` > 0) has been spent. The enclosure schemes presented so far—powerful as they are—are unable to prove this. One reason for this, illustrated in Fig. 6, is that even for point-valued initial conditions $\boldsymbol{x}_0$, the very first enclosure for an interval $t \in (0, t_1]$ must also contain the enclosure x_0 itself since the solution trajectory is a continuous function.

By using the already computed enclosures and the obvious fact that the ODE's right-hand side evaluated over these enclosures yields an overapproximation of the slopes that the trajectory can take, we extend iSAT-ODE to learn the direction of a trajectory at its "beginning". For each component of the ODE system, a *direction deduction* starts by performing an interval evaluation of the ODE's right-hand side over the first enclosure step. If this result yields a strictly positive interval, we know that the trajectory will move in positive direction in this dimension. Analogously, if the interval evaluation over the first enclosure is negative, we know that for the duration of that enclosure step, the trajectory will evolve in negative direction. In both cases, we store the end of this enclosure step as t_p and continue with the next stored enclosure box. As long as the sign information does not change, the direction of the trajectory cannot change either, and hence we increase t_p to the end of this enclosure box, thereby storing in t_p the

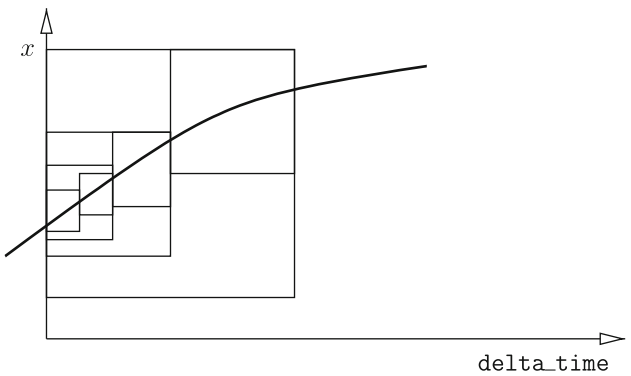
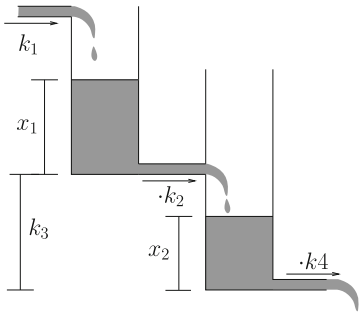


Fig. 6 Motivation for direction deduction: no matter how strong the very first enclosure box is refined, it still contains the starting point of the trajectory and hence the enclosure cannot be used to rule out that the post-value x' stays the same as the pre-value x even for `delta_time` > 0



For $x_2 > k_3$:

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\end{pmatrix} = \begin{pmatrix}k_1 - k_2\sqrt{x_1 - x_2 + k_3}\\ k_2\sqrt{x_1 - x_2 + k_3} - k_4\sqrt{x_2}\end{pmatrix}$$

For $x_2 \leq k_3$:

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\end{pmatrix} = \begin{pmatrix}k_1 - k_2\sqrt{x_1}\\ k_2\sqrt{x_1} - k_4\sqrt{x_2}\end{pmatrix}$$

Fig. 7 Structure and dynamics of the two-tank hybrid system (from [25])

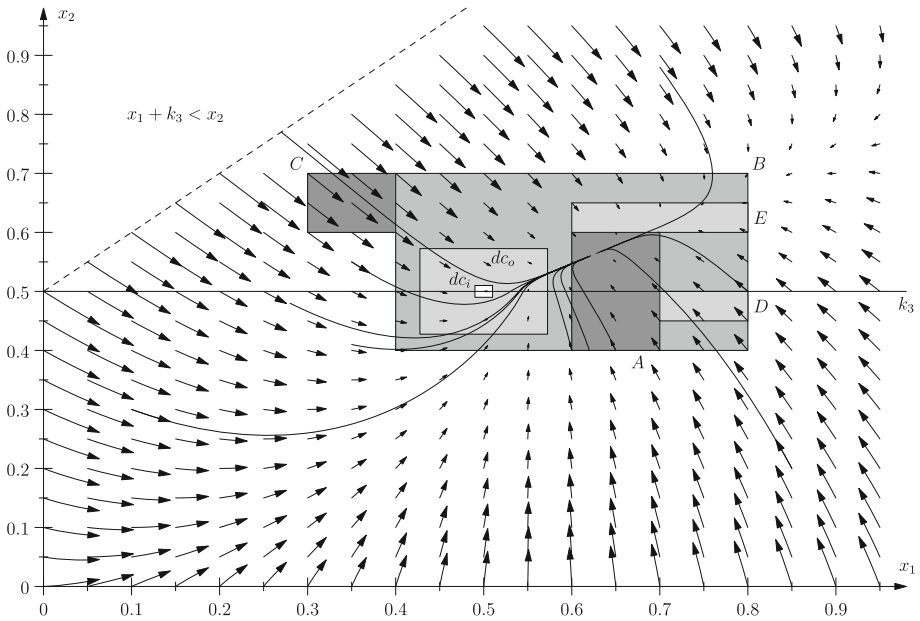


Fig. 8 Simulated trajectories for the two tanks system, inner and outer bounds of the don't care mode, and regions *A*–*E* used in the different verification conditions

duration of the prefix for which the direction is constant. When reaching the end of the enclosure or when encountering a sign change (including zero in the evaluation result), t_p is no longer changed and the iteration is stopped. If a prefix length $t_p > 0$ has been achieved, the solver can safely deduce $\texttt{delta\_time} \in (0, t_p] \Rightarrow x' > x$, i.e., that the post-value is strictly greater than the pre-value for this prefix. Analogously, we can deduce $\texttt{delta\_time} \in (0, t_p] \Rightarrow x' < x$, if the evaluation yields only values strictly less than zero.

7 Experiments

7.1 Two-tank system

To evaluate the integrated tool and the influence of the different enclosure methods, we apply our solver to the two-tank model from [25], which has been frequently used as a case study for verification tools cf., e.g., [10,22]. This system comprises two tanks connected by a tube. The first tank has an inflow of constantly $k_1 = 0.75$ volume units, and its base is $k_3 = 0.5$ length units above the base of the second tank. The connecting tube is characterized by a constant factor $k_2 = 1$, which also characterizes the outflow of the system as $k_4 = 1$.

Figure 7 illustrates this setting and formalizes the dynamic behavior of the liquid's height x_1 and x_2 in the two tanks. The system's behavior switches between two dynamics, when x_2 reaches the outlet from tank 1 and therefore exerts a counter pressure against the incoming flow. Note that the model is implicitly bounded to the case that $x_2 \leq x_1 + k_3$, since it does not provide the dynamics for the inverse direction. To understand better the dynamics of this system and the proof obligations we encoded, Fig. 8 depicts simulated trajectories.

Similar to the introductory example in Fig. 1, we encode this model predicatively using the above description directly as ODE constraints.[2]

Bounded reachability To validate the model, we first check bounded reachability properties. As can be assumed from Fig. 8, there should not be any trajectory leading from region $D = [0.70, 0.80] \times [0.45, 0.50]$ to $E = [0.60, 0.80] \times$

[2] Models and raw results from [6]: http://www.avacs.org/fileadmin/Benchmarks/Open/iSAT_ODE_SEFM_2011_models.tar.gz. Updated models and raw results: http://www.avacs.org/fileadmin/Benchmarks/Open/iSAT_ODE_SoSyM_2012_models.tar.gz.

[0.60, 0.65]. This property has been verified by Henzinger et al. using HyperTech [10].

We restrict the global time $\leq$100 and each step duration `delta_time` $\leq$10. To avoid unnecessary non-determinism in the model, all steps are explicitly enforced in the transition relation to take the maximum possible duration. They may be shorter only if they reach the switching surface at $x_2 = k_3$, if the time = 100, or if (x_1, x_2) reaches E.

In [6], we reported that our solver could prove unsatisfiability of this bounded property for up to 300 unwindings of the transition system within 3,109.1 s on a 2.4 GHz AMD Opteron machine, which is also used for the runtime measurements for this benchmark in this article. As can be seen from the runtime graphs in Fig. 9, for the same model, this runtime can no longer be achieved by the current version of our tool, unless flow invariants provide additional pruning. For the version devoid of flow invariants (graph "w/o flow invariants, all encl. methods"), the runtime has increased to 17,098 s on the same machine, a 5.5-fold increase. Also, the previously reported flat development of the cumulative runtimes is no longer observable.A likely reason for this is that our previous implementation contained a subtle bug in the collection of reasons for direction deductions in that it did not add all active ODE constraints to the reason set and hence generated conflict clauses that sometimes were too general. Such clauses can potentially prune off too large parts of the search space (in the worst case including solutions) and may hence accelerate the search in an unjustified way.

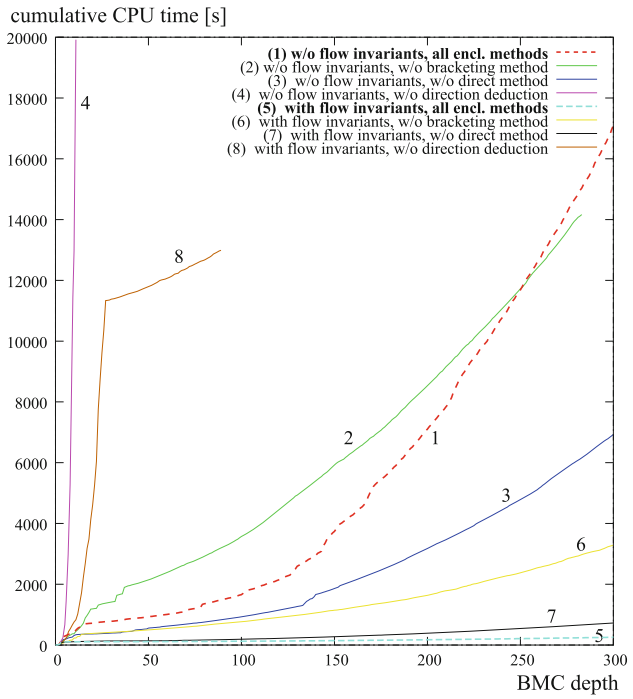


Fig. 9 Cumulative CPU times for checking (un)reachability from region D to region E for the original variant of the two-tank system without flow invariants and for a new variant which contains flow invariants for each mode. Comparison of the different solver settings, disabling one enclosure method at a time, i.e., comparing: all enclosure methods together, without bracketing, without direct enclosures, and without direction deduction

Among the other possible explanations for the slow-down, we can safely rule out detrimental effects of our additional deduction mechanisms and optimized data structures. A negative impact from the caching behavior, which has been measured to take up only 13.2 s, and also of other changes in the ODE solver (e.g., the evaluation of stored Taylor coefficients for refinement) is unlikely, since our profiling indicates that only 2,326.8 s are spent in total within functions of the ODE layer, indicating that the majority of the runtime was consumed by the iSAT core, i.e., for splitting, deductions, and conflict analysis.

We have also extended the benchmark to make use of the new flow invariant feature by adding flow invariants for the variable domains and for capturing the mode invariants of $x_2(\text{time}) \leq k_3$ or $x_2(\text{time}) \geq k_3$. For this more elaborate benchmark version, iSAT-ODE is able to prove unsatisfiability for 300 unwinding depths within just 255.7 s, a more than 12-fold improvement over the originally reported runtime.

Caching To assess the quality of the new cache implementation, we have additionally measured the runtime spent in the ODE caching layer. Over all solver runs shown in Fig. 9, the highest measured percentage is 2.5 % of the total runtime spent within the ODE layer (measured for the original version of the model without flow invariants, and the solver set to not use direction deduction). In that case, the solver (before running into the 20,000 s timeout) accumulates 544, 425 calls to the ODE solver, of which over 95 % have been detected to be cached. We therefore conclude that the cache implementation has been adequately prevented from becoming a dominant runtime factor even for large numbers of accesses.

Unbounded trajectory containment Although the formula structure is a bounded unwinding of the transition system, inductive arguments may be used to prove unbounded properties. One can easily see that region $A = [0.6, 0.7] \times [0.4, 0.6]$ contains an equilibrium point. However, the simulation also shows that there are trajectories leaving this region. We extend our model to show that trajectories can leave region A only on a bounded prefix, but thereafter stay in A forever.

First, we guess a $\tau > 0$ (supported by looking at some simulated trajectories). With

$$\mathcal{M}_l := \{\text{all trajectories of length} \geq l\},$$

from showing that

$$\forall \vec{x} \in \mathcal{M}_{2\tau} : [0, 2\tau] \to \mathbb{R}^2 : \\ (\vec{x}(0) \in A \Rightarrow \forall t \in [\tau, 2\tau] : \vec{x}(t) \in A) \tag{4}$$

follows by inductive application of (4), as facilitated by time invariance,

$$\Rightarrow \forall \vec{x} \in \mathcal{M}_\infty : [0, \infty) \to \mathbb{R}^2 :$$
$$(\vec{x}(0) \in A \Rightarrow \forall t \in [\tau, \infty) : \vec{x}(t) \in A)$$

Intuitively, we show that all trajectories of length 2τ stay in A for `delta_time` $\in [\tau, 2\tau]$ (ignoring their behavior for $[0, \tau)$). All unbounded trajectories must have these trajectories of length 2τ as prefix. At τ, they are thus (again) in A. Due to time invariance, we can consider $(x_1, x_2)(\tau)$ as a new starting point. Since it lies in A, we have already proven that for $[\tau + \tau, \tau + 2\tau]$, the trajectory will lie in A again. For the time in between, we already know that it is in A. By repeating this process ad infinitum, we know that the trajectory can never leave A again.

Note that this proof is related to the idea of *region stability* [19] and can be thought of as a stabilization proof for an unknown (and maybe hard to characterize) sub-region $A_{\text{inv}} \subseteq A$ into which all trajectories from A stabilize, and which is an invariant region for the system.

Table 1 summarizes runtimes for this proof using iSAT and the different enclosure methods. It also compares the current implementation with the older one reported in [6]. Our model encodes the above proof scheme in the following way: if a trajectory exists that is shorter than 2τ or that reaches a point outside A in time $\in [\tau, 2\tau]$, this trajectory satisfies the model. The proof is successful when the solver finds an unwinding depth k of the transition system upon which the model becomes *unsatisfiable*. Here, an unwinding depth of 3 suffices to prove the desired property. Without the direction deduction presented in Sect. 6, the solver fails to prove unsatisfiability, because it always finds counter examples that stay on the switching surface, spending there only tiny amounts of time. These trajectories satisfy the target condition of having time $\leq 2\tau$ and do not allow proving (4). Direction deduction hence enables proving the property.

The runtimes show that the approach without the direct enclosure (using only bracketing enclosures and direction deductions) outperforms both, the restriction to the direct usage of VNODE-LP with direction deduction and the combination of all enclosure methods together on this benchmark in nearly all cases. The table also shows that the changes we made to our implementation have significantly accelerated the solver in nearly all cases for this benchmark instance, with slowdowns only occurring for unwinding depth one. Note that also for this and the following instances, the results reported for the old implementation from [6] may have been influenced by the incomplete set of reasons generated for direction deductions. We assume that those too-general deductions have caused at most an undue acceleration, since they prune off parts of the search space for which they should not have been valid.

Table 1 Comparison of the old [6] and the new implementations on the two-tank system for checking unbounded containment in region A

Depth	All			No bracketing			No direct			No direction		
	Old	New	o/n	Old	New	o/n	Old	New	o/n	Old	New	o/n
1	Unknown, 111.9	Unknown,149.2	0.75	Unknown,42.0	Unknown,4.7	9.01	Unknown, 61.5	Unknown, 94.0	0.65	Unknown, 111.5	Unknown, 146.7	0.76
2	Unknown, 467.5	Unknown, 157.9	2.96	Unknown, 981.0	Unknown, 451.0	2.18	Unknown, 346.3	Unknown, 102.9	3.36	Unknown, 342.0	Unknown, 39.7	8.61
3	**UNSAT**, 674.0	**UNSAT**, 147.8	4.56	**UNSAT**, 5011.6	**UNSAT**, 196.9	25.45	**UNSAT**, 404.2	**UNSAT**, 96.5	4.19	Unknown, 478.8	Unknown, 126.0	3.80
4	**UNSAT**, 812.1	**UNSAT**, 237.2	3.42	**UNSAT**, 1995.1	**UNSAT**, 706.4	2.82	**UNSAT**, 499.1	**UNSAT**, 92.4	5.40	Unknown, 547.5	Unknown, 196.0	2.79
5	**UNSAT**, 986.0	**UNSAT**, 270.3	3.65	**UNSAT**, 2431.7	**UNSAT**, 276.1	8.81	**UNSAT**, 601.1	**UNSAT**, 125.9	4.77	Unknown, 682.4	Unknown, 243.7	2.80
6	**UNSAT**, 1126.1	**UNSAT**, 227.2	4.96	**UNSAT**, 3303.4	**UNSAT**, 466.7	7.08	**UNSAT**, 705.0	**UNSAT**, 227.3	3.10	Unknown, 834.2	Unknown, 191.7	4.35
7	**UNSAT**, 1277.2	**UNSAT**, 254.8	5.01	**UNSAT**, 2486.8	**UNSAT**, 224.7	11.07	**UNSAT**, 803.6	**UNSAT**, 143.2	5.61	Unknown, 982.5	Unknown, 328.6	2.99
8	**UNSAT**, 1451.4	**UNSAT**, 279.4	5.20	**UNSAT**, 5273.3	**UNSAT**, 406.5	12.97	**UNSAT**, 890.8	**UNSAT**, 159.6	5.58	Unknown, 1115.7	Unknown, 434.1	2.57
9	**UNSAT**, 1584.6	**UNSAT**, 328.2	4.83	**UNSAT**, 4905.2	**UNSAT**, 444.0	11.05	**UNSAT**, 966.5	**UNSAT**, 151.5	6.38	Unknown, 1235.8	Unknown, 1203.0	1.03
10	**UNSAT**, 1706.6	**UNSAT**, 312.2	5.47	**UNSAT**, 6396.1	**UNSAT**, 430.7	14.85	**UNSAT**, 1053.2	**UNSAT**, 152.2	6.92	Unknown, 1356.0	Unknown, 807.6	1.68

Column *all* shows results and CPU times (s) when using all enclosure methods combined. In the subsequent columns, one of the methods is disabled. In the *o/n* columns, the old runtime is shown in multiples of the new runtime

Introducing artificial non-determinism and hysteresis Trying a direct inductive proof for the region $B = [0.4, 0.8] \times [0.4, 0.7]$ (i.e., showing that B cannot be left with one step of the transition system) fails with our tool since B's corner at $(0.4, 0.4)$ cannot be represented exactly by floating-point numbers. To compensate, B is overapproximated to capture rounding errors and thus includes points that lie slightly outside B. Using the same proof scheme as above can be expected to work, as the simulated trajectories point inwards from the border of B. Yet, applying this proof scheme, the solver finds trajectories that can chatter indefinitely at $P = (0.5, 0.5)$, since $\dot{x}_2 = 0$ in P. This chattering is a valid behavior, though irrelevant for the actually intended proof of B's invariance.

We therefore identify intersections of the switching surface with $\dot{x}_2 = 0$ (i.e., solutions to the constraint system $k_2\sqrt{x_1} - k_4\sqrt{x_2} = 0 \wedge x_2 = k_3$) and, finding only this one in P, add a *don't-care mode* around it—depicted in Fig. 8 as $dc_i = [0.49, 0.51] \times [0.49, 0.51]$. Since this region lies well inside B, we allow any trajectory that reaches it to jump immediately or after an arbitrary positive amount of time to the outer border of the don't-care mode, illustrated by dc_o, which is $\varepsilon = 0.0625$ away from dc_i. We also forbid any trajectory to enter dc_i. This modification trades in accuracy by introducing non-determinism for the benefit of an artificial hysteresis: trajectories which could formerly stutter in P can now jump to any point on the border of dc_o, but must then move along the system's dynamics again, consuming time.

With this modification, we can prove that B is left for less than $\tau = 0.0625$ using unwinding depths $k \geq 5$. The results are shown in Table 2. For this instance of the model, the new implementation is not only faster for all unwinding depths, when the result is at least as strong as the result obtained from the old implementation, but is also capable of producing successful proofs, i.e., unsatisfiability results, more often. A potential reason could be that the evaluation of the stored Taylor coefficients using all internal solution representations computed by VNODE-LP can in some cases generate tighter enclosures than the old evaluation scheme, since it does not introduce an additional wrapping of the starting set, which was formerly unavoidable during the re-initialization of the solver.

Evaluation on an unstable instance We also applied this proof scheme to the region $C = [0.3, 0.4] \times [0.6, 0.7]$ again with unwinding depths 1–10. As expected, none of the resulting formulae was proven unsatisfiable. Runtimes were again consistently faster with the new implementation, ranging from 1.6 s for unwinding depth 1 without bracketing system usage to 492.9 s observed for depth 9, using all methods in combination. Speedups were between 1.22 for depth 9 with all methods and 13.12 for depth 5 with disabled direct VNODE-LP usage. Detailed results are shown in Table 3.

Table 2 Comparison of results and CPU times (s) for checking unbounded containment in B

Depth	All			No bracketing			No direct			No direction		
	Old	New	o/n	Old	New	o/n	Old	New	o/n	Old	New	o/n
1	Unknown, 17.7	Unknown, 2.6	6.82	Unknown, 9.4	Unknown, 1.2	8.09	Unknown, 12.9	Unknown, 1.6	7.91	Unknown, 15.4	Unknown, 2.6	5.96
2	Unknown, 163.9	Unknown, 10.2	16.02	Unknown, 57.9	Unknown, 5.2	11.08	Unknown, 81.9	Unknown, 6.9	11.82	Unknown, 157.4	Unknown, 8.7	18.14
3	Unknown, 198.9	Unknown, 16.2	12.24	Unknown, 71.8	Unknown, 8.9	8.03	Unknown, 126.9	Unknown, 10.8	11.78	Unknown, 202.3	Unknown, 12.6	16.02
4	Unknown, 666.6	Unknown, 18.0	37.07	Unknown, 193.6	Unknown, 8.4	22.94	Unknown, 146.7	Unknown, 12.2	11.99	Unknown, 206.9	Unknown, 14.4	14.41
5	**UNSAT**, 2334.2	**UNSAT**, 106.6	21.90	**UNSAT**, 3270.2	**UNSAT**, 62.6	52.20	Unknown, 183.4	**UNSAT**, 67.7	(2.71)	Unknown, 283.6	Unknown, 15.8	17.96
6	**UNSAT**, 4615.6	**UNSAT**, 265.5	17.38	**UNSAT**, 1441.2	**UNSAT**, 59.2	24.36	Unknown, 182.2	**UNSAT**, 146.3	(1.25)	Unknown, 122.0	Unknown, 18.1	6.75
7	**UNSAT**, 2967.1	**UNSAT**, 171.6	17.29	Unknown, 1934.7	**UNSAT**, 75.5	(25.61)	Unknown, 144.1	**UNSAT**, 106.2	(1.36)	Unknown, 123.9	Unknown, 21.2	5.84
8	**UNSAT**, 2559.0	**UNSAT**, 223.7	11.44	**UNSAT**, 2953.0	**UNSAT**, 74.3	39.72	Unknown, 201.6	**UNSAT**, 398.4	(0.51)	Unknown, 123.6	Unknown, 21.2	5.84
9	**UNSAT**, 2184.1	**UNSAT**, 459.4	4.75	**UNSAT**, 4121.2	**UNSAT**, 115.1	35.80	Unknown, 135.2	**UNSAT**, 181.6	(0.74)	Unknown, 127.2	Unknown, 21.2	5.98
10	**UNSAT**, 5541.6	**UNSAT**, 308.9	17.94	**UNSAT**, 7717.3	**UNSAT**, 166.3	46.39	Unknown, 272.5	**UNSAT**, 333.1	(0.82)	Unknown, 127.6	Unknown, 21.8	5.86

The *old* columns again refer to our earlier implementation [6]

Table 3 Comparison of results and CPU times (s) for checking unbounded containment in region C (not containing an equilibrium point) with old [6] and new implementation

Depth	All			No bracketing			No direct			No direction		
	Old	New	o/n	Old	New	o/n	Old	New	o/n	Old	New	o/n
1	Unknown, 55.3	Unknown, 5.9	9.34	Unknown, 20.2	Unknown, 1.6	12.42	Unknown, 34.4	Unknown, 4.3	7.93	Unknown, 54.9	Unknown, 6.2	8.79
2	Unknown, 203.3	Unknown, 24.7	8.22	Unknown, 83.4	Unknown, 6.9	12.09	Unknown, 103.8	Unknown, 17.4	5.96	Unknown, 198.3	Unknown, 25.7	7.71
3	Unknown, 308.1	Unknown, 35.1	8.78	Unknown, 121.1	Unknown, 11.4	10.62	Unknown, 155.8	Unknown, 24.3	6.41	Unknown, 291.8	Unknown, 40.3	7.24
4	Unknown, 419.1	Unknown, 56.1	7.47	Unknown, 151.0	Unknown, 15.2	9.94	Unknown, 199.9	Unknown, 35.7	5.60	Unknown, 386.6	Unknown, 54.3	7.12
5	Unknown, 499.3	Unknown, 60.9	8.20	Unknown, 163.6	Unknown, 71.2	2.30	Unknown, 551.7	Unknown, 42.0	13.12	Unknown, 468.5	Unknown, 103.0	4.55
6	Unknown, 525.6	Unknown, 73.0	7.20	Unknown, 177.8	Unknown, 44.6	3.99	Unknown, 536.6	Unknown, 51.3	10.46	Unknown, 492.9	Unknown, 74.0	6.66
7	Unknown, 555.6	Unknown, 102.7	5.41	Unknown, 196.8	Unknown, 34.2	5.75	Unknown, 449.9	Unknown, 100.2	4.49	Unknown, 524.4	Unknown, 106.6	4.92
8	Unknown, 577.6	Unknown, 94.8	6.10	Unknown, 223.8	Unknown, 49.1	4.56	Unknown, 448.9	Unknown, 62.7	7.15	Unknown, 549.3	Unknown, 98.1	5.60
9	Unknown, 599.6	Unknown, 492.9	1.22	Unknown, 235.0	Unknown, 69.6	3.38	Unknown, 447.4	Unknown, 89.2	5.02	Unknown, 574.5	Unknown, 176.5	3.25
10	Unknown, 617.6	Unknown, 93.8	6.59	Unknown, 279.7	Unknown, 52.1	5.37	Unknown, 448.7	Unknown, 214.2	2.10	Unknown, 592.2	Unknown, 159.8	3.71

Model instances with flow invariants Using the newly introduced feature of flow invariants, we repeated the checks for containment in region A and B on a modified version of the model in which the domain bounds and mode invariants for the $x_2 \geq k_3$ and $x_2 \leq k_3$ modes were added. Table 4 compares the results of the region A containment check for these two different model instances using our current implementation, Table 5 shows the same comparison for the containment check in region B. These results show that adding flow invariants to a model can influence the solving times in both directions, yielding roughly as many speedups as slowdowns between fourfold increases and 4.5-fold reductions in solving times.

7.2 A conveyor belt system

In order to evaluate iSAT-ODE on a benchmark with a larger discrete and continuous state space and more complex nonlinear dynamics, we have modeled a fictitious yet realistic system of a sorting facility in which light packages that are traveling on a conveyor belt can be pushed by an air blast from the primary lane on which they arrive to a secondary lane. Figure 11 shows a schematic drawing of this system. Objects arrive from the left $(x, y) = (0, 0)$ and move in positive x-direction with constant velocity $vx = 1$. Centered at position $x = 5$, an air fan can blow air to exert a force on the objects in its vicinity. The force applied to an object at position (x, y) is then given by $F = F_\alpha \cdot \mathrm{e}^{-((x-5)^2+y^2)}$, where F_α is the maximum force the air fan can exert. The force distribution over the position is shown in Fig. 10 for a fixed $F_\alpha = 1.0$ (in the benchmark models, F_α takes unknown but constant values from different ranges).

The controller of the system evaluates a sensor at $x = 0$, the starting position of `object_1`, and subsequently decides to activate the air flow. To capture the uncertainties in the involved measurement, the signal processing latencies, and the actuator reactiveness, we model the switching to occur at an unknown time point during $[T_{earliest\_act}, T_{latest\_act}]$. Similarly, the controller deactivates the air fan at an unknown time point during $[T_{earliest\_deact}, T_{latest\_deact}]$.

While the objects move with constant velocity $vx = 1$ in x-direction due to the mechanical coupling with the grooves in the conveyor belt, their movement in y-direction obeys a slip-stick friction model. When in sticking mode, the object's y-velocity vy is 0 and does not change. Up to a maximum static friction force, all externally applied forces are counteracted by the friction. However, as soon as the external force from the air fan overcomes this maximum friction force, the object starts to slip and the force that counteracts the acceleration caused by the air blast force is governed by kinetic friction which is substantially lower than the static friction.

Table 4 Comparison solver results and CPU times (s) using our new implementation on two variants of the containment check in region *A*

Depth	All			No bracketing			No direct			No direction		
	a	*b*	*a*/*b*	*a*	*b*	*a*/*b*	*a*	*b*	*a*/*b*	*a*	*b*	*a*/*b*
1	Unknown, 149.2	Unknown, 148.2	1.01	Unknown, 4.7	Unknown, 4.7	1.00	Unknown, 94.0	Unknown, 94.4	1.00	Unknown, 146.7	Unknown, 147.6	0.99
2	Unknown, 157.9	Unknown, 112.7	1.40	Unknown, 451.0	Unknown, 136.3	3.31	Unknown, 102.9	Unknown, 100.3	1.03	Unknown, 39.7	Unknown, 69.7	0.57
3	**UNSAT**, 147.8	**UNSAT**, 108.4	1.36	**UNSAT**, 196.9	**UNSAT**, 243.2	0.81	**UNSAT**, 96.5	**UNSAT**, 69.6	1.39	Unknown, 126.0	Unknown, 95.0	1.33
4	**UNSAT**, 237.2	**UNSAT**, 107.7	2.20	**UNSAT**, 706.4	**UNSAT**, 239.5	2.95	**UNSAT**, 92.4	**UNSAT**, 79.2	1.17	Unknown, 196.0	Unknown, 132.2	1.48
5	**UNSAT**, 270.3	**UNSAT**, 126.7	2.13	**UNSAT**, 276.1	**UNSAT**, 448.2	0.62	**UNSAT**, 125.9	**UNSAT**, 92.2	1.37	Unknown, 243.7	Unknown, 185.9	1.31
6	**UNSAT**, 227.2	**UNSAT**, 142.9	1.59	**UNSAT**, 466.7	**UNSAT**, 1182.5	0.39	**UNSAT**, 227.3	**UNSAT**, 112.2	2.03	Unknown, 191.7	Unknown, 217.1	0.88
7	**UNSAT**, 254.8	**UNSAT**, 160.9	1.58	**UNSAT**, 224.7	**UNSAT**, 582.0	0.39	**UNSAT**, 143.2	**UNSAT**, 131.8	1.09	Unknown, 328.6	Unknown, 240.6	1.37
8	**UNSAT**, 279.4	**UNSAT**, 179.4	1.56	**UNSAT**, 406.5	**UNSAT**, 696.2	0.58	**UNSAT**, 159.6	**UNSAT**, 124.5	1.28	Unknown, 434.1	Unknown, 450.2	0.96
9	**UNSAT**, 328.2	**UNSAT**, 199.4	1.65	**UNSAT**, 444.0	**UNSAT**, 1509.2	0.29	**UNSAT**, 151.5	**UNSAT**, 149.1	1.02	Unknown, 1203.0	Unknown, 266.0	4.52
10	**UNSAT**, 312.2	**UNSAT**, 217.4	1.44	**UNSAT**, 430.7	**UNSAT**, 627.7	0.69	**UNSAT**, 152.2	**UNSAT**, 157.3	0.97	Unknown, 807.6	Unknown, 532.4	1.52

Column *a* contains the results for the original model without flow invariants, column *b* those for the modified version with flow invariants

Table 5 Comparison solver results and CPU times (s) using our new implementation on two variants of the containment check in region *B*

Depth	All			No bracketing			No direct			No direction		
	a	*b*	*a*/*b*	*a*	*b*	*a*/*b*	*a*	*b*	*a*/*b*	*a*	*b*	*a*/*b*
1	Unknown, 2.6	Unknown, 2.6	0.99	Unknown, 1.2	Unknown, 1.2	0.98	Unknown, 1.6	Unknown, 1.6	0.99	Unknown, 2.6	Unknown, 2.6	1.00
2	Unknown, 10.2	Unknown, 11.3	0.91	Unknown, 5.2	Unknown, 5.7	0.92	Unknown, 6.9	Unknown, 7.6	0.91	Unknown, 8.7	Unknown, 11.7	0.74
3	Unknown, 16.2	Unknown, 23.3	0.70	Unknown, 8.9	Unknown, 10.2	0.88	Unknown, 10.8	Unknown, 14.1	0.76	Unknown, 12.6	Unknown, 15.2	0.83
4	Unknown, 18.0	Unknown, 25.2	0.71	Unknown, 8.4	Unknown, 14.4	0.58	Unknown, 12.2	Unknown, 15.4	0.80	Unknown, 14.4	Unknown, 25.3	0.57
5	**UNSAT**, 106.6	**UNSAT**, 155.6	0.69	**UNSAT**, 62.6	**UNSAT**, 93.9	0.67	**UNSAT**, 67.7	**UNSAT**, 93.3	0.73	Unknown, 15.8	Unknown, 31.9	0.50
6	**UNSAT**, 265.5	**UNSAT**, 399.8	0.66	**UNSAT**, 59.2	**UNSAT**, 61.8	0.96	**UNSAT**, 146.3	**UNSAT**, 162.7	0.90	Unknown, 18.1	Unknown, 37.2	0.49
7	**UNSAT**, 171.6	**UNSAT**, 405.2	0.42	**UNSAT**, 75.5	**UNSAT**, 72.9	1.04	**UNSAT**, 106.2	**UNSAT**, 176.7	0.60	Unknown, 21.2	Unknown, 53.4	0.40
8	**UNSAT**, 223.7	**UNSAT**, 297.4	0.75	**UNSAT**, 74.3	**UNSAT**, 147.9	0.50	**UNSAT**, 398.4	**UNSAT**, 356.0	1.12	Unknown, 21.2	Unknown, 65.7	0.32
9	**UNSAT**, 459.4	**UNSAT**, 447.7	1.03	**UNSAT**, 115.1	**UNSAT**, 69.9	1.65	**UNSAT**, 181.6	**UNSAT**, 257.9	0.70	Unknown, 21.2	Unknown, 55.4	0.38
10	**UNSAT**, 308.9	**UNSAT**, 999.8	0.31	**UNSAT**, 166.3	**UNSAT**, 206.9	0.80	**UNSAT**, 333.1	**UNSAT**, 389.0	0.86	Unknown, 21.8	Unknown, 88.2	0.25

Column *a* contains the results for the original model without flow invariants, column *b* those for the modified version with flow invariants

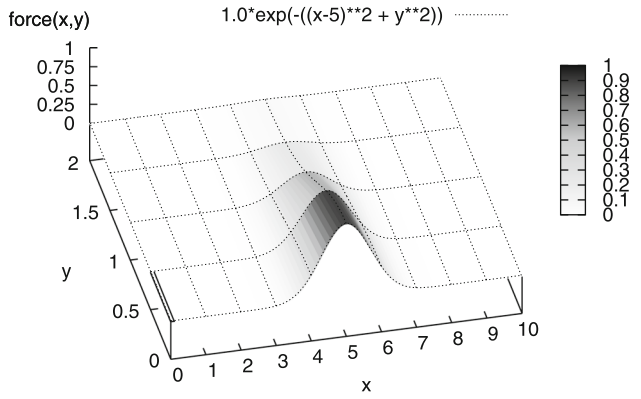


Fig. 10 Air blast force distribution over (x, y) postion

The object goes back to the sticking mode only when its velocity has decreased to zero again.

Flow invariants As described earlier, flow invariants in our formalism can only be given by simple upper or lower bounds on variables that are defined by ODEs. In the above slip-stick model, however, there is a more complex invariant on the x and y variables, representing the constraint that the object sticks as long as the force $F = F_\alpha \cdot \mathrm{e}^{-((x-5)^2+y^2)}$ does not exceed the maximum static friction force $F_{s\_max} = \mu_s \cdot m \cdot g$, where μ_s is the static friction coefficient, m the object's mass, and g the gravitational constant. The direct flow invariant for the mode `sticking` when the air blast is active (symbolized by air_blast_on) is

$$F_\alpha \cdot \mathrm{e}^{-((x-5)^2+y^2)} - \mu_s \cdot m \cdot g \leq 0.$$

In order to formulate this flow invariant in a form compatible with our restriction, we introduce a new variable

$$f := F_\alpha \cdot \mathrm{e}^{-((x-5)^2+y^2)} - \mu_s \cdot m \cdot g \leq 0,$$

and simply calculate its derivative with respect to the time t, i.e.,

$$\begin{aligned}\dot{f} &= \frac{\partial f}{\partial x} \cdot \overbrace{\dot{x}}^{=1} + \frac{\partial f}{\partial y} \cdot \overbrace{\dot{y}}^{=0} \\ &= F_\alpha \cdot \mathrm{e}^{-((x-5)^2+y^2)} \cdot (-2) \cdot (x-5) \cdot \dot{x} \\ &= F_\alpha \cdot \mathrm{e}^{-((x-5)^2+y^2)} \cdot (-2x+10).\end{aligned}$$

By adding this expression for $\dot{f}$ as an additional ODE to the system and adding as initialization of $f = F_\alpha \cdot \mathrm{e}^{-((x-5)^2+y^2)} - \mu_s \cdot m \cdot g$ when entering the mode, we can add $f \leq 0$ as flow invariant and thus model the complex condition by means of a simple upper bound on a newly introduced variable.

Modeling and encoding Figure 12 shows the complete conveyor belt model as a system of parallel hybrid automata with three components for `object_1`, `object_2`, and the controller `ctrl`. Each object automaton consists of four states to capture the different dynamics depending on whether the object is sticking or slipping and whether the air blast is active or inactive. Jumps occur hence when either the air blast is activated or deactivated by the controller (at the bottom of the figure) or when an object satisfies the condition to leave the sticking or slipping regime. Due to a restriction in iSAT-ODE, the parameters for the objects' masses m_1 and m_2 as well as for the maximum force F_α of the air blast have to be modeled explicitly as dimensions of the ODE system since their exact value is unknown. We therefore also show these explicit dimensions in the automaton. For the friction constants μ_k and μ_s and the gravitational constant g, we have assumed known exact values and therefore do not have to model them by additional dimensions. The initial values for x, F_α, m_1, and m_2 are taken from intervals denoted with $[X]$, $[FA]$, $[M_1]$, and $[M_2]$ respectively. Additionally, the controller's behavior depends on choices for $T_{earliest\_act}$, T_{latest_act}, $T_{earliest\_deact}$, and T_{latest_deact}. The instantiation of these intervals allows a significant amount of parameterization, which we exploit when using this model as a benchmark.

The nominal behavior of the system is that after reaching $t = 10$, `object_1` has reached the secondary lane of the conveyor belt, i.e., satisfies $y \geq 1$, while `object_2` has not been pushed off the primary lane, despite its proximity to the first object. A simulated trajectory that satisfies these properties can be seen in Fig. 13.

Model checking The goal of model checking is to find trajectories which violate this property, i.e., at least one of the objects ends up on the wrong lane.

Using simulation, we have identified ten parameter ranges which we will subsequently analyze by model checking using iSAT-ODE. Table 6 gives an overview over these parameters

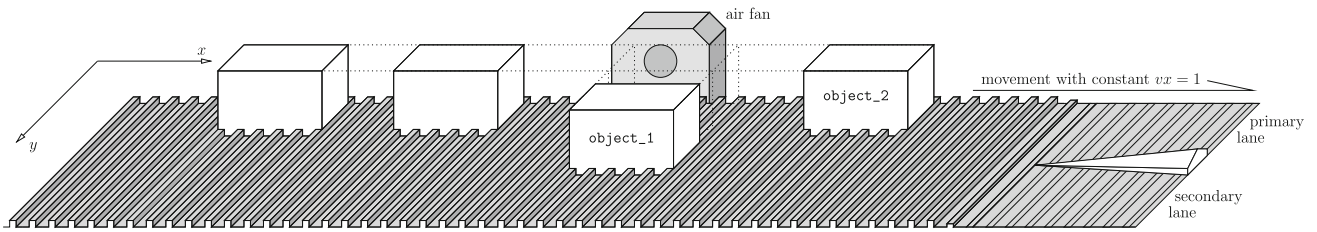


Fig. 11 Schematic drawing of the conveyor belt system

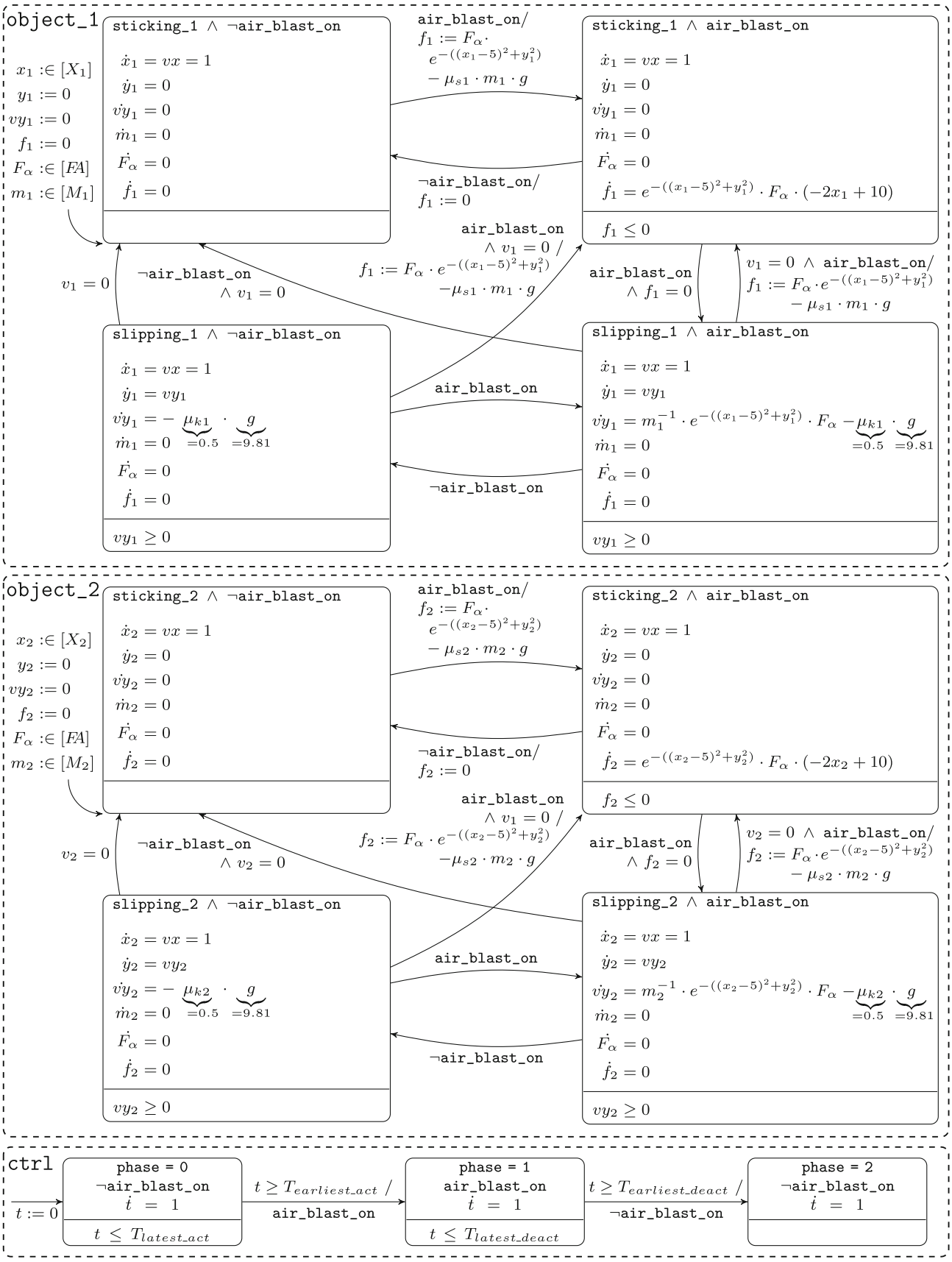


Fig. 12 Conveyor belt system modeled by parallel automata

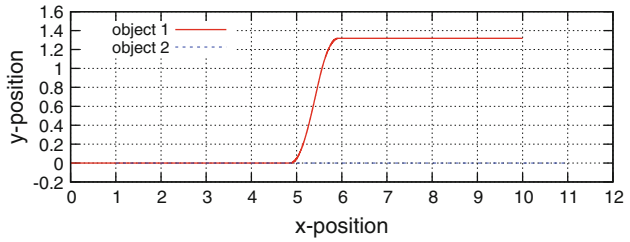


Fig. 13 Numerically simulated nominal trajectory for the conveyor belt system of 10 s length

and also shows aggregated simulation results, which give a hint at the expected outcome for model checking using each of the parameter sets.

Benchmark results Figure 14 shows all results obtained from 120 iSAT-ODE solver runs on a 2.6 GHz AMD Opteron machine (running multiple instances on parallel cores independently) with a memory limit of 8 GiB each and 50,000 s timeout. In the figure, the following abbreviations have been used: *all* (all ODE enclosure methods active), *no-brsys* (all enclosure methods except the bracketing systems), *no-direct* (all except the direct usage of VNODE-LP), *no-direction* (all except direction deduction); Heuristics: *disc-fst* (split down all discrete variables first), *dyn-rel-width* (split variable whose current range is largest relative to its domain width), and *default-heur* (default heuristic: no sorting, split round robin).

Observations and evaluation The data obtained from the simulation runs (see last row of Table 6) suggest that sets 01, 03, and 03_wider01 are unsafe, i.e., lead to error trajectories, while sets 02_point to 02_wider06 are safe, i.e., the system does not have error trajectories for parameter choices from these ranges. Consistent with this expectation, iSAT-ODE finds error trajectories in the form of candidate solution boxes for sets 01 and 03_wider01 and successfully proves unsatisfiability for up to the requested limit of 14 unwindings for sets 02_point to 02_wider05 with a varying number of solver settings. For the set 02_wider06, the solver is unable to perform this proof for BMC depths above 6, indicating that the widening of the uncertainty of the controller's phase switching time points makes the problem significantly harder to solve. Similarly, the solver runs into memory or time limits for set03 at depth 7.

The most striking outliers in Fig. 14 are caused by the disabling of the direct enclosure method. Restricted only to the bracketing system enclosure and the direction deduction, iSAT-ODE terminates with candidate solution boxes for early unwinding depths on all parameter sets from 02_point to 03, always in contradiction to solver runs in which the direct method is not disabled. This clearly indicates that for this benchmark, enclosures obtained from the bracketing system are insufficient to rule out those boxes that are finally reported as candidate solution boxes, whereas the direct enclosure is able to refute these spurious candidate solution boxes.

Equally noticeable is the runtime advantage of the *disc-fst* splitting heuristics on this benchmark over the other two heuristics. Using *disc-fst*, the solver does not split any real-valued variable, as long as there is still a Boolean or integer variable, whose width is above the minimum splitting width. The minimum splitting width was kept at its default of 0.01 for this benchmark; hence the *disc-fst* heuristic means that first all discrete variables are split down to point values (since their ranges can obviously either have a width of above 1 or of exactly 0). The effect of this heuristic is that the solver—earlier than with other heuristics—examines abstract paths of the system in which for each step only one mode or one jump can be active. This seems to guard against some unnecessary search and the costly ODE deductions it causes.

Compatible with the observation that the bracketing systems alone lead to spurious candidate solution boxes is the runtime advantage when the bracketing system is disabled. However, the difference between the *all* and *no-brsys* run-

Table 6 Parameter sets used for the instantiation of the conveyor belt benchmark

	01	02_point	02_wider01	02_wider02	02_wider03	02_wider04	02_wider05	02_wider06	03	03_wider01
X_1	$[-0.5, 0.5]$	$[0, 0]$	$[-0.1, 0.1]$	$[-0.1, 0.1]$	$[-0.1, 0.1]$	$[-0.1, 0.1]$	$[-0.2, 0.2]$	$[-0.2, 0.2]$	$[-0.2, 0.2]$	$[-0.2, 0.2]$
M_1	$[0.08, 0.12]$	$[0.1, 0.1]$	$[0.1, 0.1]$	$[0.09, 0.11]$	$[0.09, 0.11]$	$[0.09, 0.11]$	$[0.09, 0.11]$	$[0.09, 0.11]$	$[0.09, 0.11]$	$[0.09, 0.11]$
X_2	$[0.0, 2.5]$	$[1, 1]$	$[1, 1]$	$[1, 1]$	$[1, 1]$	$[1, 3]$	$[1, 3]$	$[1, 3]$	$[1, 3]$	$[1, 3]$
M_2	$[0.08, 0.12]$	$[0.1, 0.1]$	$[0.1, 0.1]$	$[0.1, 0.1]$	$[0.09, 0.11]$	$[0.09, 0.11]$	$[0.09, 0.11]$	$[0.09, 0.11]$	$[0.09, 0.11]$	$[0.09, 0.11]$
FA	$[0.7, 1.5]$	$[1, 1]$	$[1, 1]$	$[1.2, 1.3]$	$[1.2, 1.3]$	$[1.2, 1.3]$	$[1.2, 1.3]$	$[1.2, 1.3]$	$[1.1, 1.2]$	$[1.0, 1.3]$
TA	$[4.0, 4.99]$	$[4.5, 4.5]$	$[4.5, 4.5]$	$[4.5, 4.5]$	$[4.5, 4.5]$	$[4.5, 4.5]$	$[4.5, 4.5]$	$[4.4, 4.6]$	$[4.3, 4.7]$	$[4.3, 4.7]$
TD	$[5.00, 6.00]$	$[5.5, 5.5]$	$[5.5, 5.5]$	$[5.5, 5.5]$	$[5.5, 5.5]$	$[5.5, 5.5]$	$[5.5, 5.5]$	$[5.4, 5.6]$	$[5.3, 5.7]$	$[5.3, 5.7]$
#	645	0	0	0	0	0	0	0	4	75

Each column shows the intervals used for the system parameters using the same names as in the automaton in Fig. 12, except for $TA := [t_{earliest\_act}, t_{latest\_act}]$ and $TD := [t_{earliest\_deact}, t_{latest\_deact}]$. The very last row of the table shows how many out of 1,000 simulation runs using randomly chosen points from the column's parameter set were violating at least one property, i.e., one of the objects ended up on the wrong lane. Based on this estimate, parameters chosen from sets 02_point to 02_wider06 are expected not to cause any error trajectories

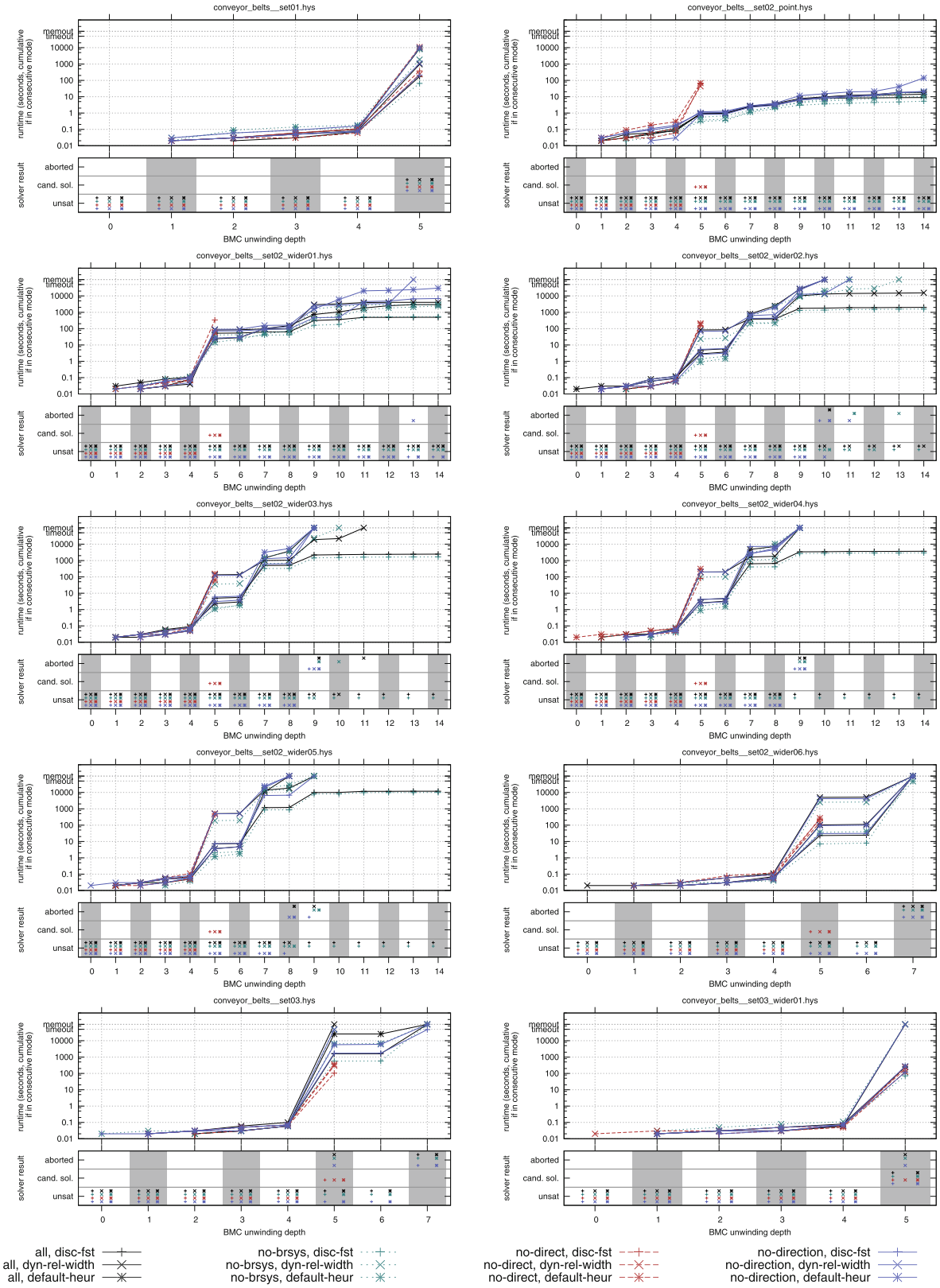


Fig. 14 Runtimes and results for the conveyor belt benchmark. *All* all ODE enclosure methods active, *no-brsys* all enclosure methods except the bracketing systems, *no-direct* all except the direct usage of VNODE-LP, *no-direction* all except direction deduction, *disc-fst* split down all discrete variables first, *dyn-rel-width* split variable whose current range is largest relative to its domain width, *default-heur* default heuristic: no sorting, split round robin

times is not very large, indicating that the computational cost of generating the bracketing enclosures is not high in this example and that they do not significantly influence the search process either. Without knowing in advance whether the bracketing system's enclosures work on a given problem, this benchmark's results suggest that even if they do not contribute enough to the deduction to solve the benchmark successfully on their own, their computational cost is so low in such a case that it is a good default choice to have them activated.

7.3 Comparison with hydlogic

To conclude our evaluation, we compare our tool with the results published in [11] for the `hydlogic` tool, which is the technologically most closely related competitive approach, being also based on a satisfiability modulo ODE scheme and having a VNODE-LP core for handling of non-linear ODEs. In [11], Ishii et al. present several case studies and the results they obtained for them. Where appropriate, they also compare the results with PHAVer and HSolver. Our comparison with the `hydlogic` results therefore also yields an indirect comparison with these other tools, which we hence do not repeat.

Based on the description as hybrid automata in [11], we have remodeled some of these systems in our predicative encoding for iSAT-ODE. As even small changes in the modeling approach or subtle variations in the encoding can lead to dramatically different results (especially runtimes), we want to emphasize that such a comparison can only give a limited snapshot of the actual relation of the tools.

7.3.1 Car steering problem

The first benchmark from [11] is based on a car steering controller originally investigated by [2]. We depict our version of the automaton for this system in Fig. 15. The car's movement is modeled by its position p and its heading γ. Its initial position and heading are unknown, but bounded by intervals. When the car reaches one of the borders of the street, its heading is changed continuously and the time measured until the car reaches the border again (now heading inwards). The heading is now changed in the opposite direction for half the time that the car has spent outside of the road boundaries. The maneuver ends in the unsafe `in_canal` state when the position reaches $p \leq 1.5$.

Modeling details The cited versions of this automaton have an additional sink mode `straight_ahead` which is reached from `correct_left` and `correct_right` when the counter reaches zero (before the obverse border is reached). If flows are allowed to take zero time or jumps to immediately follow one another, an instantaneous sequence of mode changes arises which terminates in an inappropriate

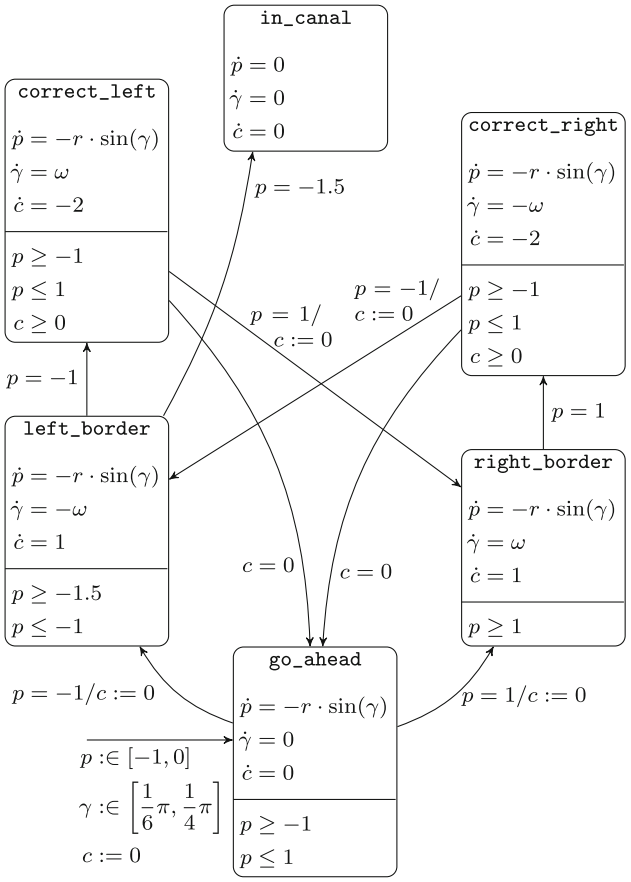


Fig. 15 Car steering system based on [2,11].

mode. Under this semantics, traces may proceed immediately into `correct_left` after entering `left_border` with $p = 1$. As the counter remains $c = 0$ under these circumstances, the trace races through to and then stays forever in `straight_ahead`, although the car has actually never changed its direction when crossing the border and will definitely reach $p \leq 1.5$. Since neither of [2,11] detect this race condition in the model they present, we have changed the model in two ways[3]: (a) we have collapsed the `straight_ahead` mode with `go_ahead`, such that this zero-time trace would not be able to hide the eventual reaching of the `in_canal` state, and (b) have added a condition that, when entering the modes `left_border` and `right_border`, there must follow a flow, and it must take strictly more than zero time.

For the simple constant ODE components $\dot{p} = 0$, $\dot{\gamma} = \pm\omega$, and $\dot{c} = \{-2, 0, 1\}$, we added the closed-form linear solutions as redundant encodings, since they are easily obtained and may help with deduction. This step could be

[3] Note that we found this trace when validating our encoding of the original model with iSAT-ODE and were surprised to find this obviously unintended trajectory which is compatible with this often assumed semantics of hybrid systems (e.g., in [14]).

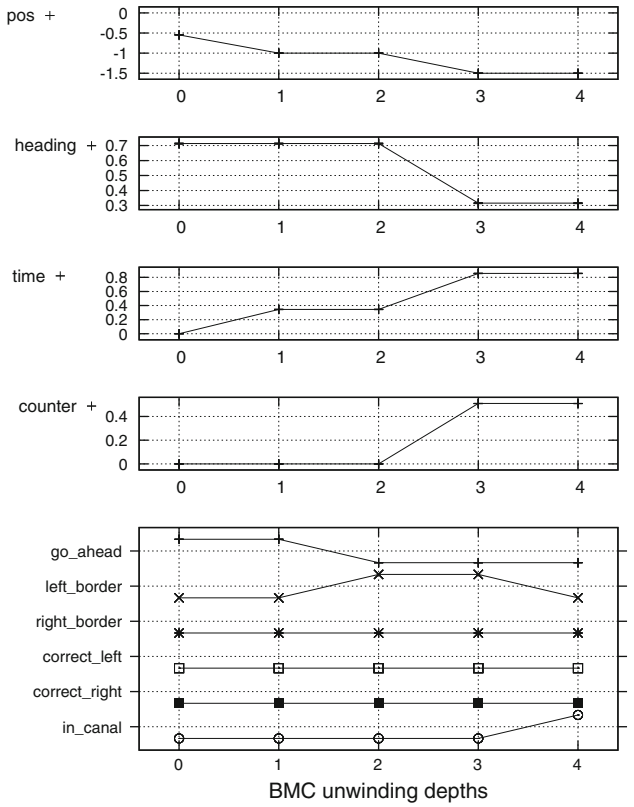


Fig. 16 An iSAT-ODE trace for the *steering-1* benchmark instance. Value of variables at the BMC unwinding depths

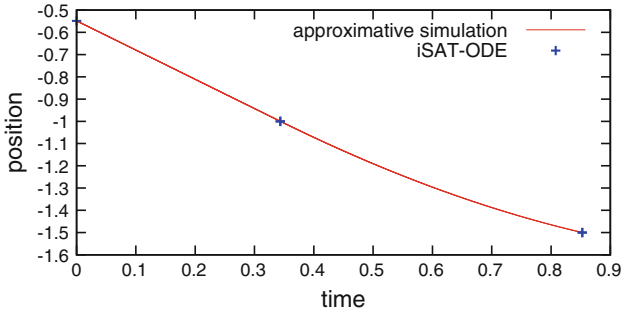


Fig. 17 Car positions for the *steering-1* benchmark obtained from approximative numerical simulation compared with the candidate solution box from the iSAT-ODE trace

automated, e.g., as a preprocessing step, but has currently not been implemented in our tool. The bounds for the initial values of the car's heading γ have been overapproximated by representable interval boundaries. For the angular velocity ω, we approximate by $\omega = 0.78539816$ the value $\pi/4$ that is given in [2]. From there, we also take the value for the radius $r = 2$. In order to reduce unnecessary non-determinism, we also disallow any flows that end before they satisfy a guard condition. This stuttering in one mode is otherwise very costly for the search, since there are infinitely many points to interrupt a flow that all have to be examined, if the system is modeled in a naïve way.

Results. In [11], Ishii et al. analyze the following four scenarios. We summarize their results from their paper and compare them with the iSAT-ODE results as shown in detail in Fig. 18.

For the *steering-1* scenario, which is using the model as described above and in Fig. 15, `hydlogic` finds a trajectory leading to `in_canal` in two or three steps and proves its existence. With iSAT-ODE, we also find a trajectory with all tried settings for four BMC unwindings. As can be seen from the trace in Fig. 16, these four steps amount to a sequence of one flow in `go_ahead`, a mode switch to `left_border`, a flow in that mode, and a final switch to `in_canal`. Figure 17 shows that this trace is consistent with a numerically approximated trajectory emerging from the starting values identified by iSAT-ODE. Our result is weaker only in the lack of a proof that the identified trace really exists. The runtime reported in [11] is 5.31s on 2.4 GHz Intel Core 2 Duo processor. Our runtimes on a newer AMD Opteron 2.6 GHz processor are spread out, but the best can be considered competitive with this number.

In the *steering-2* instance of this benchmark, the canal is moved to the left by 0.5 units, such that the guard for entering `in_canal` becomes $p = -2$. We also change the invariant of mode `left_border` to cover this widened range. The `hydlogic` result for this safe instance of the system is reported in [11] as *unknown* after three steps and 198.30 s of runtime. With iSAT-ODE, the graph in the upper right corner of Fig. 18 shows clearly that for the large majority of settings, we can prove unsatisfiability of the formula up to much larger numbers of unwindings. The solver runtimes diverge depending on the chosen heuristics. While the default heuristic leads to timeouts after depth 17, the other two heuristics allow successful refutation up to unwinding depth 30, before running into the 50,000 s timeout limit. Again, one notable observation is that disabling the direct method leads to candidate solution boxes as has been observed already in the conveyor belt benchmark.

The *steering-3* instance is the same as *steering-2*, except that the initial range for the position is restricted to $p :\in [-0.9, 0]$. This restriction helps `hydlogic` to prove unsatisfiability for three steps within 22.16s. The results for iSAT-ODE are very similar to the *steering-2* example. If we compare the iSAT-ODE runtimes for six unwindings of the formula with `hydlogic`'s reported runtime for three steps, we consider our runtimes to be competitive even when factoring in the newer CPU architecture.

Finally, *steering-4* is the same as *steering-3*, except that the initial ranges are set to $p :\in [-1, 1]$ and $\gamma :\in [-\pi/4, \pi/4]$. For `hydlogic` [11], reports a timeout at 1,200 s of runtime for checking satisfiability for three steps of the system. The lower right graph in Fig. 18 shows that iSAT-ODE can prove

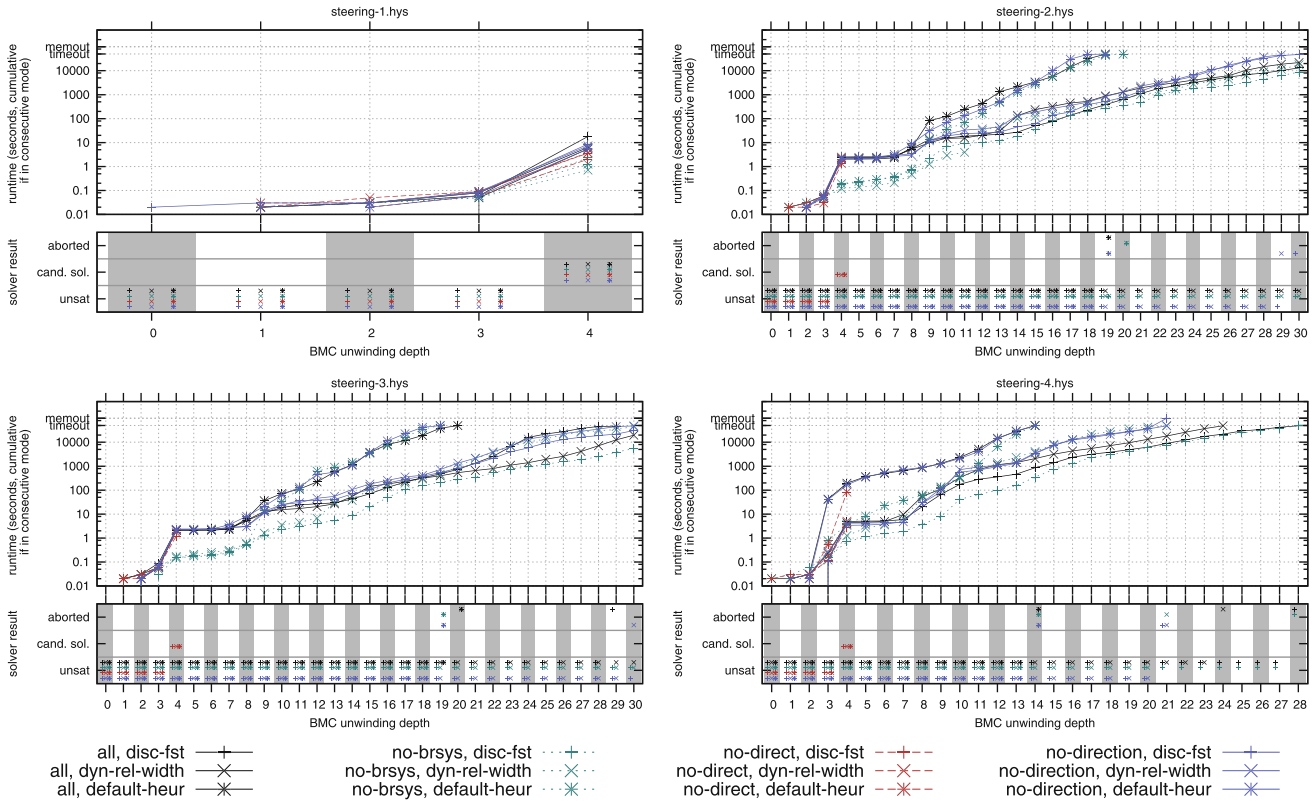


Fig. 18 Results and runtimes for four instances of the car steering benchmark

unsatisfiability up to unwinding depths of 27 with the *disc-fst* and disabled bracketing system based enclosures. Using these settings, iSAT-ODE can, e.g., prove unsatisfiability for 12 unwinding depths in less than 100 s.

7.3.2 Two-tank system

Structurally the same model as has already been shown in Sect. 7.1, the version of the two-tank system from [11] uses different parameters (all $k_i = 1$), additional invariants, and different initial ranges. The model checking goal is then classical reachability rather than the stabilization properties we examined in Sect. 7.1. The property to be checked by BMC is the reachability of an unsafe mode, which is characterized by a circular region in the (x_1, x_2) state space. For the sake of clarity, we repeat the automaton from [11] in Fig. 19 and—since the brevity of this model allows it—also show the full encoding of it in the iSAT-ODE language, to give a better practical impression about the encoding step.

Modeling details. Ishii et al. instantiate the model with $A = [5.25, 5.75] \times [0, 0.5]$. Due to the $\sqrt{x_2}$ term that occurs in the ODE system, this initial range for x_2 is problematic since it leaves no margin for numerical overapproximation where x_2 grows into the negative range. While there is no report of problems in [11] for `hydlogic` with this issue, we have observed the VNODE-LP layer in iSAT-ODE running into long sequences of deduction failures caused by the underlying interval library's reports of encountered numerical errors. Without deduction, the solver is obviously free to split down the box in the proximity of $x_2 = 0$, consequently finding spurious traces there, whose refutation would have required successful generation of ODE enclosures. In order to be able to compare our results, we have opted for a modification of the benchmark such that the initial value of x_2 is positively separated from 0 and have therefore chosen $A = [5.25, 5.75] \times [0.01, 0.5]$.

In Fig. 20, we show the complete encoding of the system. The first part contains the variable declaration, including the ranges for all variables and the special `time` and `delta_time` variables. The next section defines the initial states of the system: starting at time point 0, with any value for $(x_1, x_2) \in A$, being in mode `s1` and starting with a continuous flow instead of a jump. The transition predicate contains explicitly the semantic knowledge that time progresses in each step exactly by the amount of the duration `delta_time` and that the system cannot be in both modes at the same time (only the guard condition for entering the `unsafe` mode has been modeled by a predicate in line 48 that can be true independently of the current mode, as long as (x_1, x_2) is in the unsafe region). Lines 20–27 and 29–36 contain the continuous dynamics and flow invariants for modes

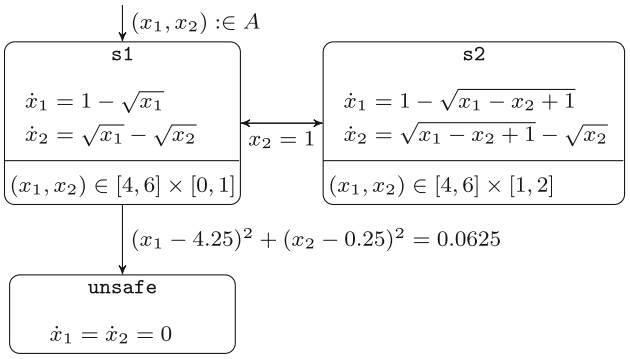


Fig. 19 Model of the two-tank system from [11]

```
1  DECL
2      float [−10, 10] x1, x2;
3      float [0, 1000] time;
4      float [0, 1000] delta_time;
5      boole s1, s2;
6      boole flow;
7      boole unsafe;
8  INIT
9      time = 0;
10     x1 >= 5.25; x1 <= 5.75;
11     x2 >= 0.01; x2 <= 0.5;
12      s1;
13     !s2;
14     !unsafe;
15     flow;
16 TRANS
17     time' = time + delta_time;
18     s1' + s2' = 1;
19
20     flow and s1 ->
21         (d.x1 / d.time = 1 − nrt(x1, 2));
22     flow and s1 ->
23         (d.x2 / d.time = nrt(x1, 2) − nrt(x2, 2));
24     flow and s1 -> (x1(time) >= 4);
25     flow and s1 -> (x1(time) <= 6);
26     flow and s1 -> (x2(time) >= 0);
27     flow and s1 -> (x2(time) <= 1);
28
29     flow and s2 ->
30         (d.x1 / d.time = 1 − nrt(x1 − x2 + 1, 2));
31     flow and s2 ->
32         (d.x2 / d.time = nrt(x1 − x2 + 1, 2) − nrt(x2, 2));
33     flow and s2 -> (x1(time) >= 4);
34     flow and s2 -> (x1(time) <= 6);
35     flow and s2 -> (x2(time) >= 1);
36     flow and s2 -> (x2(time) <= 2);
37     flow -> ((s1 and s1') or (s2 and s2'));
38     flow -> delta_time > 0;
39
40     flow -> (!flow' or unsafe');
41
42     !flow -> x2 = 1.0;
43     !flow -> ((s1 and s2') or (s2 and s1'));
44     !flow -> flow';
45     !flow -> delta_time = 0;
46     !flow -> (x1' = x1 and x2' = x2);
47
48     unsafe' <-> (x1' − 4.25)^2 + (x2' − 0.25)^2 = 0.0625;
49 TARGET
50     s1;
51     unsafe;
```

Fig. 20 Two-tank model encoded in the iSAT-ODE input language

`s1` and `s2`, respectively. The `nrt` symbol stands for the *n*th root. Line 37 makes it explicit that flows do not change the current mode and line 38 requires that flows actually have positive duration (which is a design choice that is suitable for this model). Line 40 requests that after a flow either a jump occurs or the unsafe state is reached—enforcing that flows are not interrupted without having reached a guard condition. Line 42 encodes the guard condition: if the mode is changed (line 43), the guard $x_2 = 1$ must hold. The remainder of the section encodes that there are no two jumps following directly after each other (again a design choice for this model) and that jumps do not take time and do not change the continuous variables (since the automaton has no actions). In lines 50–51, the property to be checked is whether the system can reach the guard condition for entering the mode `unsafe` while being in state `s1`.

Results For the *twotanks-1* instance, which is exactly as shown in Fig. 19 and Fig. 20, `hydlogic` (again all results quoted from [11]) reports unsatisfiability for two steps within 33.49 s. The left part of Fig. 21 shows that all iSAT-ODE instances are capable of proving unsatisfiability for up to 40 unwindings of the formula. The runtimes for checking the 40 unwindings consecutively are spread between 1.25 and 20.5 s. At least the fastest settings can therefore be considered to be significantly faster than `hydlogic` even when taking into account the differences in the CPUs on which the benchmarking was performed.

In the *twotanks-2* instance, the region of unsafety is moved such that it becomes reachable. The new guard condition for entering the mode `unsafe` is $(x_1 - 4.5)^2 + (x_2 - 0.75)^2 = 0.0625$. For this instance, `hydlogic` finds a trajectory of two steps length and proves its existence within 36.34 s. As has been detailed earlier, in our model the target property is to find a valuation that satisfies the entrance guard for the `unsafe` mode while in `s1`. Therefore, the solution trajectory in our model is reached already within one step (omitting the final jump to `unsafe`. The trace consists of just a direct flow from a state admissible in the initial condition and following the dynamics of `s1` to the unsafety region. As can be seen from the right part of Fig. 21, iSAT-ODE finds this one-step trajectory within just a few seconds. The shortest runtime result for the depth 1 unwinding is 3.63 s with the default heuristic and disabled bracketing, the longest runtime for depth 1 is 24.56s with the *dyn-rel-width* heuristic and all ODE enclosure methods enabled.

Again, the iSAT-ODE result is weaker than the result from `hydlogic` since there is no guarantee that the identified candidate solution box contains a solution.[4]

[4] Note that in this special case where the solution consists of only one flow, using just the ODE enclosure and showing that all points from its prebox satisfy the initial condition and all points from the last enclosure lie within the unsafe region, would yield an equally strong proof.

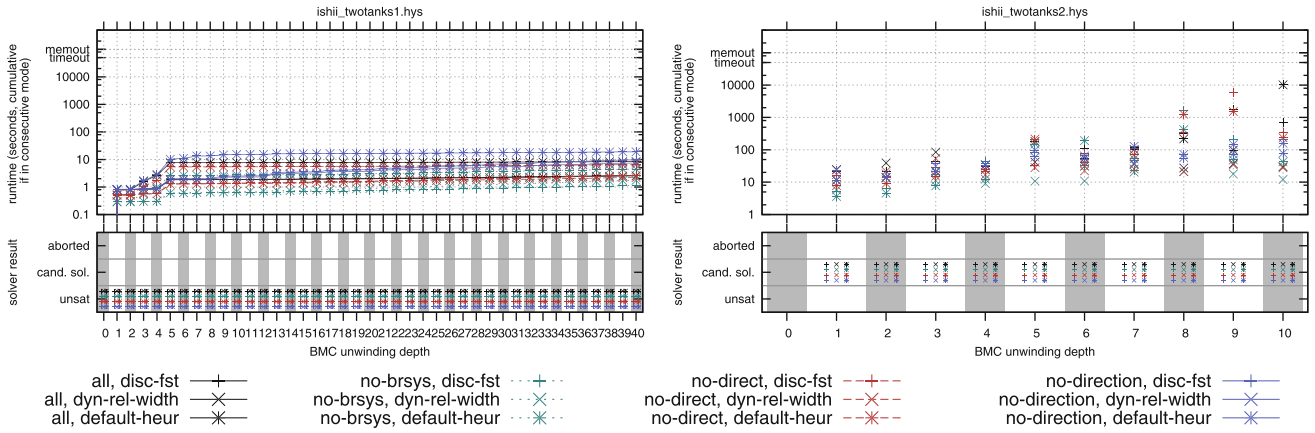


Fig. 21 Results and runtimes for the instances of the two-tank benchmark as parameterized in [11]

To do a more extensive analysis of solver runtimes, we forced the solver to check unwinding depths individually instead of using the consecutive mode results that can be seen, e.g., in the left part of Fig. 21. The first observation is that runtimes for larger unwindings spread out significantly. These instances become harder to solve since the solver needs to find a trajectory that reaches the unsafe region, but still has more than the one flow step that is actually required. As can be seen in line 40 of Fig. 20, our model requires alternating jumps and flows except when the unsafe predicate $(x_1 - 4.5)^2 + (x_2 - 0.75)^2 = 0.0625$ is true. Since this is satisfied already by the endpoint of the trajectory after the first step, the solver needs to find a valuation for the remaining variable instances (e.g., 9 remaining steps for 10 unwindings) such that this predicate still holds or find an alternating sequence of jumps and flows to satisfy line 40.

A solution to this is based on exploiting the overapproximation that occurs in the interval-based ODE enclosures. Although the constraint in line 38 enforces that flows have a duration strictly larger than 0, even the tightest enclosure will still contain the starting points of that flow (which was the argument needed to motivate the direction deduction presented in Sect. 6). Those solution traces for larger unwindings that we investigated further therefore lead directly to the satisfied unsafe predicate in the first step and thereafter contained steps of very short duration, e.g., `delta_time` $\in (0, 0.00011517)$, which is strictly greater than zero, but still small enough such that the equality constraint for the unsafe predicate was still satisfied due to the enclosure still containing the original prebox. In a way, these results could hence be considered spurious and a stronger form of the direction deduction might have been able to rule out satisfiability for larger unwindings of the formula unless there also exist paths that really perform some alternation between the two modes before reaching the unsafe region.

7.3.3 Bouncing ball on sine-waved surface.

The last example from [11] that we use for our comparison is the model of a ball that bounces off from a sine-wave surface, called *bouncing3* in that paper.

Modeling details. Like the classical bouncing ball hybrid automaton, this behavior can be modeled by one mode and a self-loop that is triggered when the height of the ball reaches the ground, which in this case is not flat, but satisfies the constraint $p_y = \sin(p_x)$. Ishii et al. use `hydlogic` to "simulate" the system for ten steps and "assumed that the ball bounces at the earliest crossing point between the ball and the ground". While this refers probably to an algorithmic assumption to search and use the first intersection of the ODE enclosure with a guard condition and pruning after all parts of the enclosure are past the guard, we think that this can be formulated explicitly inside the automaton. From our perspective this means that there should be a flow invariant,

$$p_y \geq \sin(p_x) \Leftrightarrow \underbrace{\sin(p_x) - p_y}_{=:g} \leq 0$$

such that the ball cannot reach a point below the sine curve. Using the same modeling trick that we have detailed earlier, we add a new flow invariant $g \leq 0$ and the ODE $\dot{g} = \cos(p_x) \cdot v_x - v_y$, which is the derivative of the original flow invariant with respect to time. Additionally, the value of g must be initialized correctly to $g := \sin(p_x) - p_y$ (Fig. 22).

Results. For the "simulation" [11], reports that `hydlogic` proves unsatisfiability for unwinding depth 10 within 29.15s. The longest trajectory, that could be found by iSAT-ODE, was for unwinding depth 15 and took 19,060 s of CPU time, using the *disc-fst* heuristic with disabled bracketing systems. We were able to validate that all intermediate elements of this candidate solution box contained points on the $p_y = \sin(p_x)$ surface.

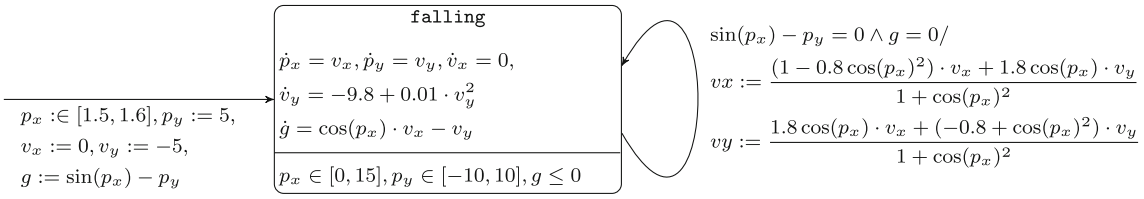


Fig. 22 Our version of the hybrid automaton for the bouncing ball on a sine-waved surface extending the original from [11] by an extra dimension g that captures the flow invariant

Assuming that in the `hydlogic` results, again, one "step" consists of a flow and a jump, we would need to solve unwinding depth 20 to analyze the same 10 steps instance for which `hydlogic` could report unsatisfiability. For this unwinding depth, iSAT-ODE was not able to find a candidate solution or prove unsatisfiability within 50,000 s. We therefore consider iSAT-ODE to be clearly slower on this benchmark.

8 Conclusion

After exploring the feasibility of using ODE enclosures to solve SAT modulo ODE problems in [5], this paper and its previous version [6] extend and improve the abilities of the resulting solver by combining enclosure methods. We have shown that the techniques presented in this paper have complementary strengths and that our integrated approach is capable of handling different types of proof obligations for nonlinear hybrid systems. Our improvements are orthogonal to the application of interval Newton contractors in [9,11] and could be extended in the same way to gain the ability to prove existence of solutions.

In this extended version of [6] we have addressed the formerly missing capability of handling flow invariants. Our approach now allows the direct encoding of axis-aligned flow invariant constraints and uses them during deductions to prune off parts of the step duration `delta_time` for which all trajectories have at least once left the region admitted by the active flow invariant constraints. A technique reducing more general invariants, e.g., those defined by nonlinear inequalities, to simple ones at the expense of increased dimensionality of the ODE system has been demonstrated by example. A comprehensive analysis of the expressiveness of this technique is subject to future work. A possible extension of our tool may be based on using the same interval narrowing functions from the iSAT core also on pruning inconsistent parts of the ODE solution enclosures.

Finally, we have improved the use of VNODE-LP in our solver by extracting the Taylor coefficients and internal solution representations, which allow faster and—depending on the interval widths—also potentially more accurate evaluations of the enclosure set.

Our experiments have shown that the enhanced ODE evaluations and technical improvements of the caching layer, used to avoid unnecessary re-computations for previously encountered queries to the ODE evaluations, have led to significant performance gains over the results we reported in [6], especially on larger instances of the previously examined case study. Furthermore, we have substantially extended our experimental evaluation by a new case study and by comparisons with the `hydlogic` tool [11] on its own benchmarks. While these comparisons clearly show that our approach is competitive with the state of the art, the complexity of the newly introduced conveyor belt case study shows that SAT modulo ODE solving has gotten at least a small step closer to the direct analysis of real-world hybrid systems.

In our future work, we will explore ways to automatically build bracketing systems when off-diagonal elements of the Jacobian change signs during the continuous evolution.

Acknowledgments We would like to thank Stefan Ratschan, Christian Herde, Tino Teige, Jens Oehlerking, and Corina Mitrohin for discussions on the region-stability-related proof scheme utilized for the experiments in this paper and all colleagues from the transregional research center AVACS, project H1/2 "Constraint-based Verification for Hybrid Systems" for the joint development of the iSAT core. Additionally, we are grateful to the reviewers of [6] for their detailed comments. Especially by insisting on a more thorough experimental evaluation and by pointing out shortcomings in our presentation, the SoSyM reviewers have helped tremendously to improve the quality of this paper. Thank you!

References

1. Berz, M.: COSY INFINITY version 8 reference manual. Tech. Rep. MSUCL-1088, National Superconducting Cyclotron Laboratory, Michigan State University, USA (1997)
2. Clarke, E.M., Fehnker, A., Han, Z., Krogh, B.H., Stursberg, O., Theobald, M.: Verification of hybrid systems based on counterexample-guided abstraction refinement. In: Gravel, H., Hatcliff, J. (eds.) TACAS, Lecture Notes in Computer Science vol 2619, pp. 192–207. Springer, Berlin (2003)
3. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
4. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Commun. ACM **5**, 394–397 (1962)
5. Eggers, A., Fränzle, M., Herde, C.: SAT modulo ODE: a direct SAT approach to hybrid systems. In: ATVA, LNCS, vol. 5311, pp. 171–185. Springer, New York (2008)

6. Eggers, A., Ramdani, N., Nedialkov, NS., Fränzle, M.: Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In: Barthe, G., Pardo, A., Schneider, G. (eds.) Proceedings of the Ninth International Conference on Software Engineering and Formal Methods (SEFM), LNCS, vol. 7041, pp. 172–187. Springer, Berlin (2011). doi:10.1007/978-3-642-24690-6-13
7. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: a multiple-precision binary floating-point library with correct rounding. ACM Trans. Math. Softw. **33**(2) (2007). doi:10.1145/1236463.1236468, MPFR is available at http://www.mpfr.org/
8. Fränzle, M., Herde, C., Ratschan, S., Schubert, T., Teige, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. JSAT **1**(3–4), 209–236 (2007)
9. Goldsztejn, A., Mullier, O., Eveillard, D., Hosobe, H.: Including ordinary differential equations based constraints in the standard CP framework. In: Cohen, D. (ed.) Principles and Practice of Constraint Programming—CP 2010, LNCS, vol. 6308, pp. 221–235. Springer, Berlin (2010)
10. Henzinger, T., Horowitz, B., Majumdar, R., Wong-Toi, H.: Beyond HyTech: hybrid systems analysis using interval numerical methods. In: Lynch, N., Krogh, B. (eds.) Hybrid Systems: Computation and Control, LNCS, vol. 1790, pp. 130–144. Springer, New York (2000)
11. Ishii, D., Ueda, K., Hosobe, H.: An interval-based SAT modulo ODE solver for model checking nonlinear hybrid systems. Int. J. Softw. Tools Technol. Transf. (STTT), 1–13 (2011). doi:10.1007/s10009-011-0193-y
12. Kieffer, M., Walter, E., Simeonov, I.: Guaranteed nonlinear parameter estimation for continuous-time dynamical models. In: Proceedings 14th IFAC Symposium on System Identification, Newcastle, pp. 843–848 (2006)
13. Lerch, M., Tischler, G., Gudenberg, J.W.V., Hofschuster, W., Krämer, W. Filib++, a fast interval library supporting containment computations. ACM Trans. Math. Softw. **32**(2):299–324 (2006). doi:10.1145/1141885.1141893, FILIB++ is available at http://www2.math.uni-wuppertal.de/~xsc/software/filib.html
14. Lygeros, J., Johansson, K., Simic, S., Zhang, J., Sastry, S.: Dynamical properties of hybrid automata. IEEE Trans. Autom. Control **48**(1), 2–17 (2003). doi:10.1109/TAC.2002.806650
15. Müller, M.: Über das Fundamentaltheorem in der Theorie der gewöhnlichen Differentialgleichungen. Mathematische Zeitschrift **26**, 619–645 (1927)
16. Nedialkov, N.S.: Computing rigorous bounds on the solution of an initial value problem for an ordinary differential equation. PhD thesis, Department of Computer Science, University of Toronto, Toronto, M5S 3G4 (1999)
17. Nedialkov, N.S.: VNODE-LP—a validated solver for initial value problems in ordinary differential equations. Tech. Rep. CAS-06-06-NN. Department of Computing and Software, McMaster University, Hamilton, L8S 4K1, VNODE-LP is available at http://www.cas.mcmaster.ca/~nedialk/vnodelp (2006)
18. Nedialkov, N.S.: Implementing a rigorous ODE solver through literate programming. In: Rauh, A., Auer, E. (eds.) Modeling, Design, and Simulation of Systems with Uncertainties. Mathematical Engineering, vol. 3, pp. 3–19. Springer, New York (2011). doi:10.1007/978-3-642-15956-5_1
19. Podelski, A., Wagner, S.: Region stability proofs for hybrid systems. In: Raskin, J.F., Thiagarajan, P.S. (eds.) FORMATS, LNCS, vol. 4763, pp. 320–335. Springer, Berlin (2007)
20. Ramdani, N., Meslem, N., Candau, Y.: A hybrid bounding method for computing an over-approximation for the reachable space of uncertain nonlinear systems. IEEE Trans. Autom. Control **54**(10), 2352–2364 (2009)
21. Ramdani, N., Meslem, N., Candau, Y.: Computing reachable sets for uncertain nonlinear monotone systems. Nonlinear Anal. Hybrid Syst. **4**(2), 263–278 (2010)
22. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation based abstraction refinement. ACM Trans. Embed. Comput. Syst. **6**(1), (2007)
23. Shtrichman, O.: Tuning SAT checkers for bounded model checking. In: Emerson, E., Sistla, A. (eds.) Computer Aided Verification, LNCS, vol. 1855, pp. 480–494. Springer, Berlin (2000). doi:10.1007/10722167_36
24. Stauning, O.: Automatic validation of numerical solutions. PhD thesis, Technical University of Denmark, Lyngby, (1997). http://www2.imm.dtu.dk/documents/ftp/phdliste/phd36_97.ps, FADBAD++ is available at http://www.fadbad.com
25. Stursberg, O., Kowalewski, S., Hoffmann, I., Preußig, J.: Comparing timed and hybrid automata as approximations of continuous systems. In: Antsakalis, P., Kohn, W., Nerode, A., Sastry, S. (eds.) Hybrid Systems IV, LNCS, vol. 1273, pp. 361–377. Springer, Berlin (1997). doi:10.1007/bfb0031569

Author Biographies

Andreas Eggers obtained his BSc degree (2005) in Computer Science and his M.Sc. degree (2006) in Embedded Systems and Microrobotics from the Carl von Ossietzky University in Oldenburg, Germany. Since 2007, he has been working in the AVACS Collaborative Research Center on constraint-based analysis of hybrid systems. His research focus is the combination of satisfiability checking with enclosure methods for Ordinary Differential Equations.

Nacim Ramdani received the Engineer degree from Ecole Centrale de Paris, France, in 1990, the Ph.D. degree from the University of Paris-Est Créteil, France, in 1994 and the Habilitation in 2005. Since September 2010, he has been a full Professor at the Université of Orléans (IUT de Bourges) and member of the Laboratoire PRISME. From 1996 to 2010, he was Maître de Conférences with the University of Paris-Est Créteil. He was affiliated with the Robotics Department of LIRMM CNRS Montpellier during 2005–2010 and also on secondment with INRIA during 2007–2009. His current research interests include modeling and analysis of cyber-physical and hybrid systems in presence of uncertainty, and set membership estimation, with applications to robotics and human-robot interaction.

Nedialko S. Nedialkov was born and raised in Bulgaria. He received M.Sc. (1995) and Ph.D. (1999) degrees in Computer Science at the University of Toronto, and has been with the Department of Computing and Software at McMaster University since 1999. His research is in the general area of scientific computing and mathematical software with emphasis on interval numerical methods for differential equations and numerical methods for differential-algebraic equations. He is the author of the VNODE and VNODE-LP packages for computing rigorous bounds on solutions in initial-value problems for ordinary differential equations, and the DAETS package for solving high-index differential algebraic equations.

Martin Fränzle is Professor for Hybrid Discrete-Continuous Systems at the University of Oldenburg since 2004, where he also is member of the board of the Interdisciplinary Research Center for Safety Critical Systems and of the research division Transportation of the affiliated research institute OFFIS. He furthermore is member of the board as well as head of research area hybrid systems of the Transregional Research Center SFB/TR 14 AVACS. His research focuses on effective methods for representing the dynamics and analysing the safety and stability of hybrid-state systems, with a special focus on systems in the transportation domain.