Foundations of the Automatic Analysis of Cyber-Physical Systems

by

Stefan Ratschan

A habilitation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague.

Prague, December 2014
Abstract

This habilitation thesis presents work that addresses foundational questions of the automatic analysis of cyber-physical systems. Here we understand cyber-physical systems as systems that tightly integrate computation with physical processes. As a basis, we use the formalism of hybrid dynamical systems, and present several results on the formal verification of such systems. The according algorithms reduce the verification task to a constraint solving task for predicate logical formulas over the real numbers. Hence, the thesis also contains several contributions to real number constraint solving. Those are useful for the automatic analysis of cyber-physical systems but are relevant in many other areas, and so they are presented independently, as results on their own.
Contents

1 Introduction and Overview 2
   1.1 Cyber-Physical Systems .................................................. 2
   1.2 Hybrid Dynamical Systems ................................................. 3
   1.3 System Design, Analysis, and Verification ............................ 6

2 Discussion of Contributions 8
   2.1 Formal Verification .......................................................... 10
   2.2 Constraint Solving ......................................................... 11

3 References 13
   3.1 List of Author’s Publications Included in this Thesis ................ 13
   3.2 List of Selected Author’s Publications not in this Thesis .............. 14
   3.3 List of References by Other Authors .................................... 16

4 Safety Verification of Hybrid Systems by Constraint Propagation Based
   Abstraction Refinement ....................................................... 21

5 Safety Verification of Non-linear Hybrid Systems is Quasi-Decidable 45

6 Guaranteed Termination in the Verification of LTL Properties of Non-
   linear Robust Discrete Time Hybrid Systems ............................. 66

7 Safety Verification for Probabilistic Hybrid Systems 91

8 Providing a Basin of Attraction to a Target Region of Polynomial Sys-
   tems by Computation of Lyapunov-like Functions ....................... 108

9 Constraints for Continuous Reachability in the Verification of Hybrid
   Systems ................................................................. 127

10 Efficient Solving of Quantified Inequality Constraints over the Real Num-
    bers ................................................................. 143
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Efficient Solution of a Class of Quantified Constraints with Quantifier Prefix Exists-Forall</td>
<td>170</td>
</tr>
<tr>
<td>12</td>
<td>Satisfiability of Systems of Equations of Real Analytic Functions is Quasi-decidable</td>
<td>183</td>
</tr>
</tbody>
</table>
Preface

This thesis includes several publications of the author (Chapters 4 to 12), preceded by an enframing discussion (Chapters 1 to 3). Throughout, references starting with the letters “AI” refer to publications of the thesis author included as Chapter 4 to 12 of this document, references starting with the letters “AE” (E for external) refer to other publications of the thesis author, and references starting with the letter “O” refer to publications by other authors. Those publications are listed, separated into these categories, as sections of Chapter 3.
Chapter 1

Introduction and Overview

In this chapter, we give a general introduction and overview of the subject of this thesis.

1.1 Cyber-Physical Systems

In this section, we review the notion of a cyber-physical system and the role of models in the design of such systems. We present a specific class of models, hybrid dynamical systems, that will serve as a basis for the questions addressed in this thesis and discuss the automatic analysis of hybrid dynamical systems.

Citing Edward A. Lee [O34 page 1], “a cyber-physical system (CPS) is an integration of computation with physical processes.” Among others, Lee states the following examples: information technological support for heart surgery, a non-centralized traffic control system based on inter-car interaction, electronic pilot-support systems in airplanes, and multiple autonomous quadrotors cooperating on a common task.

In current industrial practice, systems involving both computation and physical processes are more and more designed based on models, that is, a formal description of the given system that is abstract in the sense that it describes only those aspects of the system that are important in the current stage of the design process and ignores other aspects. To this end, certain modeling languages and tools are used (e.g., Matlab/Simulink, Modelica) that partially automatize the design process using methods for the automatic simulation and analysis of the built models. The current research area of model based design [O44, O29, O14] tries to advance the usage of models in systems design.

Modeling languages used in industrial practice are often very rich, allowing a broad range of expressive features and containing extensive libraries of modeling components. The downside of this expressivity is the complexity of those formalisms and the lack of a precise formal semantics [O9]. Since this thesis concentrates on foundational aspects, where a clean and elegant mathematical formalization is essential, we use a small and

1The notion of a cyber-physical system is often also used in a broader sense, or even as a keyword summarizing a certain vision and resulting research agenda (NSF, EU). Instead of this, in this thesis we will restrict ourselves to the simple technical definition given above.
1.2 Hybrid Dynamical Systems

The Turing Award winner Joseph Sifakis cites as one of the three grand challenges for rigorous system design [O50]: “We need theory and models encompassing continuous and discrete dynamics to predict the global behavior of a [computational] system interacting with its physical environment.”

The classical way of modeling computation are discrete formalisms such as finite automata. The classical way of modeling physical systems is based on continuous mathematics, especially differential equations. Hence, a natural way of modeling the integration of computation with physical processes is based on formalisms that integrate differential equations with finite automata. This gives rise to the notion of a hybrid dynamical system, often just called hybrid system, that combines continuous with discrete state and evolution. This ignores several aspects of cyber-physical systems (e.g., components [O46], physical distribution [O22, O16], process scheduling [O35], network communication [O40], and aspects resulting from specific application areas) but serves its role as a clean basis for studying foundational questions arising from the interaction between physical and computational components.

In the literature, definitions of the notion of hybrid system come in several flavors, that we roughly classify as follows:

- automata based [O4, O48, O31] (very often called “hybrid automaton”)
- set/function based [O23, O51]
- constraint/logic based [O45]

In this thesis we mainly use the set and function based approach, sometimes using constraints/logical formulas to describes those sets. More specifically, we use variants of the following definition:

**Definition 1.2.1** A hybrid system is a tuple \((M, \Phi, \text{Init}, \text{Flow}, \text{Jump})\) where

- \(M\) is a finite set, whose elements are usually called modes,
- \(\Phi \subseteq M \times \mathbb{R}^n\),
- \(\text{Init} \subseteq \Phi\),
- \(\text{Flow} \subseteq \Phi \times \mathbb{R}^n\),
- \(\text{Jump} \subseteq \Phi \times \Phi\).

\(^2\)The word in brackets was added by the author of this thesis for clarification purposes.
CHAPTER 1. INTRODUCTION AND OVERVIEW

The intuition is that $\Phi$ describes the state space of the system, with discrete part $M$ and continuous part $\mathbb{R}^n$. Init describes the set of initial states, Flow describes the continuous behavior of the hybrid system, and Jump describes the discrete behavior of the hybrid system.

Before making precise, how those elements determine the dynamical behavior of a hybrid system, we illustrate this definition based on the widely used example of a thermostat controlling the heating of a room. Here, the elements of the tuple $(M, \Phi, \text{Init, Flow, Jump})$ are as follows:

- $M = \{ \text{on, off} \}$
  (the heating can either be on or off)

- $\Phi = M \times \mathbb{R}$, i.e. $n = 1$
  (we model the continuous behavior using one real variable, the temperature in the room)

- $\text{Init} = \{(m, x) \mid 10 \leq x \wedge x \leq 25\}$
  (initially, the temperature is between 10 and 25 degrees, independently of whether the heating is on or off)

- $\text{Flow} = \{(m, x, \dot{x}) \mid [m = \text{on} \Rightarrow x \leq 21 \wedge \dot{x} = 30 - x] \wedge [m = \text{off} \Rightarrow x \geq 19 \wedge \dot{x} = -x] \}$
  (evolution of the temperature follows two differential equations, depending on whether the heating is on or off; in addition, continuous evolution is only possible if the temperature is not higher than 21 when the heating is on, or less then 19, when the heating is off, respectively)

- $\text{Jump} = \{(m, x, m', x') \mid [m = \text{on} \wedge x \geq 21] \Rightarrow [m' = \text{off} \wedge x' = x] \vee [m = \text{off} \wedge x \leq 19] \Rightarrow [m' = \text{on} \wedge x' = x]\}
  (the heating switches off if the temperature is at least 21 degrees and on if the temperature is 19 degrees or less)

Often such hybrid systems are visualized using graphs such as the one in Figure 1.1.

![Figure 1.1: Visualization of Heating Example](image-url)
The sets $\Phi, \text{Init}, \text{Flow}, \text{Jump}$ are usually infinite. So, we cannot directly represent them on computers using their elements. Instead, we work with a finite, symbolic representation of these sets. The constraints used for describing those sets in the above example (e.g., $10 \leq x \land x \leq 25$), form such a representation. In the thesis we will often call the result a hybrid system description which then falls into the class of constraint/logic based descriptions of hybrid systems mentioned above.

Widespread special cases of hybrid systems include switched systems [37, 36], differential equations with dis-continuous right-hand side [18], differential inclusions [5], timed automata [3], and rectangular automata [27, 28].

One can define the evolution of a hybrid system as follows:

**Definition 1.2.2** A flow of length $l$ of a hybrid system $H = (M, \Phi, \text{Init}, \text{Flow}, \text{Jump})$ is a pair $(m, \varphi)$ where $m \in M$ and $\varphi : [0, l] \to \mathbb{R}^n$. By abuse of notation, for a flow $\phi = (m, \varphi)$ of length $l$, and $t \in [0, l]$, we denote by $\phi(t)$ the pair $(m, \varphi(t))$. A trajectory of $H$ is a sequence of flows $\phi_0, \ldots, \phi_p$ of lengths $l_0, \ldots, l_p$ such that

- $r_0(0) \in \text{Init}$,
- if $i > 0$ then $(r_{i-1}(l_{i-1}), r_i(0)) \in \text{Jump}$,
- if $l_i > 0$ then for all $t \in [0, l_i]$, $(r(t), \dot{r}(t)) \in \text{Flow}$.

For example, Figure 1.2 illustrates a trajectory of the thermostat example given above, where each element of the sequence of flows is drawn into one and the same diagram. Note also, that each flow has a local notion of time starting from zero that the diagram joins into a global time axis.

![Figure 1.2: Example Trajectory](image)

The above definitions are powerful enough to allow the modeling of common features of hybrid systems such as invariants or forced/non-forced jumps. The fact that Flow is
based on a relation instead of a function $\Phi \to \mathbb{R}^n$ allows us to define non-deterministic hybrid systems, that is, hybrid systems for which a given initial point does not determine a unique trajectory starting from that point, but that allows many different trajectories to start in a given initial point. This permits the modeling of uncertainty or leaving open behavior that one does not want to specify in more detail at a given point of time in the design process. Constraints defining this relation $\text{Flow}$ can model such non-deterministic behavior using differential inequalities, instead of differential equations. For example, the constraint $1 \leq \dot{x} \wedge \dot{x} \leq 2$, restricts the derivative to the interval $[1, 2]$, but allows any derivative in this interval.

Note that we use the term “system” in “hybrid system” in an essentially different way from its usage in “cyber-physical system”: A hybrid system is an abstract mathematical formalism whereas a cyber-physical system is a concrete object from reality. Especially, it is not necessarily the case that the continuous part of a hybrid system models the physical and the discrete part models the computational part of a cyber-physical system. This is just a design choice that often turns out to be convenient. Still, there are many exceptions. For example, the gears of a motor or physical impact (bouncing ball) are often modeled discretely. For details, we refer to several textbooks [O38, O45, O48, O51, O23] and surveys [O10, O47, O24, O25, O49, O31] in the area.

### 1.3 System Design, Analysis, and Verification

When designing cyber-physical systems, two activities play a major role:

- **Modeling**: creating a model of the desired system.
- **Analysis**: analyzing the behavior of the created model.

These two activities are repeatedly iterated. One of the advantages of the usage of models is the fact that already early in the design process models are available whose analysis can be supported by computer tools. The most widely used tool for analysis is simulation, that is, the computation of (approximations of) possible behavior of the model. For hybrid systems, this means the computation of (approximations of) possible trajectories. Today, the simulation of hybrid systems is mature technology [O43, O17, O2].

The research area of model checking [O13, O6, O1] tries to automatize the analysis of models at a higher level: Instead of computing a single behavior of the model, it tries to design algorithms that formally verify a given model in the following sense:

- **Given**: a model $\mathcal{M}$ and formal specification $S$.
- **Either prove that all behaviors of the model fulfill the formal specification, or compute some example behavior (usually called counter-example) that shows that the model does not fulfill its specification.**
1.3. SYSTEM DESIGN, ANALYSIS, AND VERIFICATION

Up to recent years, the model \( \mathcal{M} \) was usually required to have finitely many states. Only recently, there has been major progress in the model checking of systems with infinitely many states, allowing the modeling and model checking of data structures [O30, O42], time [O7], probability [O32], and continuous/hybrid dynamics [O26, AI1, O19, O8, O15].

There are various ways of providing the formal specification \( S \). In this thesis we will mostly concentrate on safety verification, where the specification \( S \) requires that the model \( \mathcal{M} \) always stays in a given set of states that we consider to be safe, that is, it never reaches a state that we consider to be not safe. In the case of hybrid systems, this can be formalized by extending the definition of a hybrid system with a set of unsafe states \( \text{Unsafe} \subseteq \Phi \) and calling a trajectory \( \phi_0, \ldots, \phi_p \) of lengths \( l_0, \ldots, l_p \) an error trajectory if and only if there is an \( i \in \{0, \ldots, p\} \) and \( t \in [0, l_i] \) such that \( \phi_i(t) \in \text{Unsafe} \). Then, for a given hybrid system \( H \), we either prove that \( H \) is safe, that is, that it does not have an error trajectory, or compute a counter-example in the form of an error trajectory of \( H \).

A major question in the formal safety verification of hybrid systems is, how to analyze their continuous behavior. A widespread technique to do so, is to reduce the analysis of this continuous behavior, including the analysis of ordinary differential equations, to a constraint solving problem in a first-order theory of real numbers. We also follow this approach in this thesis, classifying the contributions of this thesis into formal verification and constraint solving.
Chapter 2

Discussion of Contributions

On the next page, the reader will find a map structuring the publications included in this thesis. As already mentioned, we classify them into formal verification (upper part of the figure, discussion in Section 2.1) and constraint solving (lower part of the figure discussion in Section 2.2). The order of both the discussion and the included articles also follows the map from top to bottom, with exception of the first part that corresponds to the upper third of the map: In this part, we start with an article solving the central problem of this thesis (safety verification of hybrid systems [AI1]), then, theoretical problem analysis [AI2, AI3], and extension [AI4].
constraint solving

solving quantified constraints [AI7, AI8]

theoretical problem analysis

quasi-decidability [AI9]

usage

usage

usage

analogous problem

basin of attraction [AI5]

usage

safety verification algorithm [AI1]

extension

constraint solving

verification of probabilistic systems [AI4]

quasi-decidability [AI2, AI3]

theoretical problem analysis

constraints for verification [AI6]
2.1 Formal Verification

The first contribution is described in an article [AI1] that solves the central problem of this thesis: safety verification of hybrid systems. Its advantage over earlier algorithms solving this problem lies in the fact that it can handle non-linear continuous dynamics, including differential equations and inequalities whose right-hand side contains function symbols such as sin or exp. Moreover, while the algorithm is based on floating-point arithmetic for efficient computation, it always handles rounding errors in a conservative way. The implementation of the algorithm is available online at [http://hsolver.sourceforge.net](http://hsolver.sourceforge.net).

Most formal verification tasks for hybrid systems are undecidable [O27]. Decidability results exist only for very special cases [O33, e.g.,]. Algorithms for more general cases, including the algorithm discussed in the previous paragraph, are usually based on their ability to verify a wide class of benchmark problems efficiently. Our contributions described in two further articles [AI2, AI3] show that this problem of undecidability can be circumvented by providing a procedure for safety verification that may run forever, but that terminates successfully in all cases that are in a certain sense practically relevant. More specifically, the algorithms terminate in all cases that are robust, that is, where small changes to the hybrid system do not change its safety. The fact that engineers usually design systems to be robust implies practical relevance of this class. One of the contributions handles the discrete-time case [AI3], the other one the continuous-time case [AI2]. In both cases, the goal of the provided algorithm is the proof of termination in the robust case, but not practical efficiency. An algorithm that both terminates in all robust cases and is practically efficient is left for future work.

The classical hybrid systems formalism includes non-determinism only in its set-based form, where the system may evolve according to an arbitrary element of a set. For example, the constraint \( \dot{x} \in [1,2] \) says that the derivative of \( x \) can be an arbitrary element of the interval \([1,2]\). Here, no element of the set \([1,2]\) has priority over any other element. Especially, no probability is assigned to any element. In our contribution [AI4], we design a safety verification algorithm for probabilistic hybrid systems, that is hybrid systems that allow the modeling of probabilistic non-determinism. The algorithm reduces the probabilistic verification problem to a verification problem without probabilities, allowing the usage of tools for non-probabilistic safety verification such as the one included in this thesis [AI1].

Most of the contributions on formal verification included in this thesis pertain to safety verification. We also include an article [AI5] that verifies that a given non-linear ordinary differential equation eventually reaches an element from a set of target states. The article follows the overall structure of the thesis: It shows how the given verification problem can be reduced to a constraint solving problem, and then gives an algorithm for efficiently solving the resulting constraints. The constraints have the quantifier prefix \( \exists \forall \). Constraints of this form play an important role in the field of verification [O12].

A further contribution [AI6] introduces several possibilities of reducing reachability problems for ordinary differential equations to a constraint solving problems in the first-order predicate logical theory of the real numbers, connecting the results on formal verifi-
2.2 CONSTRAINT SOLVING

We have seen that efficient constraint solving algorithms are key to many algorithms in formal verification of hybrid systems. The first article on constraint solving in this thesis describes an algorithm for efficiently solving constraints \[AI7\] with quantifiers. Such constraints also have many other applications \[AE23\]. In the application described in this thesis \[AI1\], the time variable is quantified in order to describe trajectories over an arbitrarily long time span.

For efficiency reasons, it is essential to exploit the structure of constraints occurring in a specific application problem. In the previous chapter, we have already observed that in verification, constraints with the specific quantifier prefix \(\exists \forall\) play an important role \[AI5, O12\]. Here, the universal quantifier is used to characterize the global behavior of the system over the whole state space and the existential quantifier ensures the existence of an object that functions as a certificate for the global property of the given system. We include a contribution providing an efficient algorithm for solving constraints with such a structure \[AI8\].

Due to a classical result by A. Tarski \[O52\], the first-order predicate logical theory of real-closed fields, that formalizes the real numbers with addition and multiplication, allows quantifier elimination, and hence is decidable. However, adding a function symbol representing the sine function, makes the theory undecidable, since the sine function is periodic, and hence it allows encoding the theory of integers that is undecidable \[O41\]. Still, in a similar way as described above for the safety verification of hybrid systems \[AI2, AI3\], it is possible to find a procedure that takes as input formulas with such function symbols, returns a correct result, if it terminates, and terminates in all cases, when the input formula is in a certain, precisely defined sense, robust.

The constraint solving algorithms in this thesis fulfill this property \[AI7, AI8\]. So, in contrast to hybrid systems verification, in this case, the gap between theoretical quasi-decidability and practical efficiency is already closed. However, here, equalities of the form \(f = 0\) are handled as a short-cut for two inequalities of the form \(-f \leq 0 \land f \leq 0\). This may destroy robustness, since a solution of \(f = 0\) may vanish if the two occurrences of \(f\) in \(-f \leq 0 \land f \leq 0\) are slightly changed. Hence the algorithms do not provide any termination guarantees when proving satisfiability of equalities. In many applications this is not a problem, since either this case does not occur, or relaxation of equalities to \(-f \leq \epsilon \land f \leq \epsilon\), for a small positive real number \(\epsilon\), is fine. Still, another included contribution can handle systems of equalities for which the number of variables coincides with the number of equalities directly \[AI9\] without rewriting to inequalities, again terminating in all robust cases.

Since the publication of the articles discusses above, the above results have been partially re-invented, partially extended under the keyword \(\delta\)-decidability \[O20, O21\]. Also those articles cannot prove satisfiability of equalities, and hence our contribution \[AI9\] (to-
gether with its extended and revised version [AE24] is still the strongest in this direction.
Chapter 3

References

In this chapter, we provide three lists of references, depending on whether they refer to publications included into this thesis (prefix “AI”), to other publications by the thesis author (prefix “AE”), or to publications by other authors (prefix “O”).

3.1 List of Author’s Publications Included in this Thesis


Chapter 3. References

3.2 List of Selected Author’s Publications not in this Thesis


3.2. LIST OF SELECTED AUTHOR’S PUBLICATIONS NOT IN THIS THESIS


3.3 List of References by Other Authors


Chapter 4

Safety Verification of Hybrid Systems by Constraint Propagation Based Abstraction Refinement

This chapter appeared as:

This paper deals with the problem of safety verification of nonlinear hybrid systems. We start from a classical method that uses interval arithmetic to check whether trajectories can move over the boundaries in a rectangular grid. We put this method into an abstraction refinement framework and improve it by developing an additional refinement step that employs interval-constraint propagation to add information to the abstraction without introducing new grid elements. Moreover, the resulting method allows switching conditions, initial states, and unsafe states to be described by complex constraints, instead of sets that correspond to grid elements. Nevertheless, the method can be easily implemented, since it is based on a well-defined set of constraints, on which one can run any constraint propagation-based solver. Tests of such an implementation are promising.

Categories and Subject Descriptors: I.6.4 [Simulation and Modeling]: Model Validation and Analysis; C.4 [Performance of Systems]: Reliability, Availability, and Serviceability

General Terms: Algorithms, Reliability, Verification

Additional Key Words and Phrases: Hybrid systems, intervals, constraint propagation

ACM Reference Format:

This is a revised and extended version of an earlier paper [Ratschan and She 2005]. This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS). See www.avacs.org for more information.

Authors’ address: Stefan Ratschan, Institute of Computer Science, Czech Academy of Sciences, Prague, Czech Republic; email: stefan.ratschan@cs.cas.cz; Zhikun She, School of Science, Beihang University, Beijing, China; email: zkshe77@hotmail.com.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org. © 2007 ACM 1539-9087/2007/02-ART8 $5.00 DOI 10.1145/1210268.1210276 http://doi.acm.org/10.1145/1210268.1210276
1. INTRODUCTION

The environment in which embedded computing systems operate is, in many cases, described by physical laws that are formulated using differential equations. When these are taken into account in verification, the resulting model is a hybrid system. In this paper, we provide a method for verifying that a given nonlinear hybrid system has no trajectory that starts from an initial state and reaches an unsafe state.

Our approach builds upon a known method that decomposes the state space according to a rectangular grid and that uses interval arithmetic to check the flow on the boundary between neighboring grid elements. The reasons for choosing this method as a starting point are: it can do verification instead of verification modulo rounding errors, it can deal with constants that are only known up to intervals, and it uses a check that is less costly than explicit computation of continuous reach sets, or checks based on quantifier elimination. However, this method has the drawback that it may require a very fine grid to provide an affirmative answer. In this paper we provide a remedy to this problem.

In our solution we put the classical interval method into an abstraction refinement framework where the abstract states represent hyperrectangles (boxes) in the continuous part of the state space. Here, refinement corresponds to splitting boxes into pieces and recomputing the possible transitions. In order to avoid splitting into too many boxes, we employ an idea that is at the core of the field of constraint programming: instead of a splitting process that is potentially exponential in the dimension of the problem, we try to deduce information without splitting, in an efficient, but possibly incomplete, constraint propagation step. Here we use conditions on the motion of the trajectories within these boxes to construct a constraint without a differentiation operator, whose solution contains the reach set. We then employ an interval constraint-propagation algorithm to remove elements from the boxes that do not fulfill this constraint.

Many algorithms for checking the safety of hybrid systems are based on floating-point computation that involve rounding errors and on algorithms whose correctness can be hampered because of the fact that local instead of global optima are used for intermediate computation. In many cases, this is a perfectly valid approach. However, in some safety critical applications one would like to verify safety. Experience shows that just replacing floating-point computation by faithfully rounded interval operations either results in too wide intervals, or—in combination with splitting—in inefficient algorithms. Thus, we tried to develop a genuine interval-based approach here.

Our implementation of the algorithms is publically available [Ratschan and She 2004].

The structure of the paper is as follows: in Section 2, we formalize our safety verification problem; in Section 3, we put the classical interval-based method into an abstraction refinement framework; in Section 4, we improve the method, using interval constraint-propagation techniques; in Section 5, we give a further improvement of the method that employs incremental computation; in Section 6, we discuss our implementation; in Section 7, we illustrate the behavior of the implementation using some benchmarks problems; in Section 8, we
discuss the benchmarking results; in Section 9, we cover related work; and in Section 10, we conclude the paper.

2. PROBLEM DEFINITION

We fix a variable \( s \) ranging over a finite set of discrete modes \([s_1, \ldots, s_n]\) and variables \( x_1, \ldots, x_k \) ranging over closed real intervals \( I_1, \ldots, I_k \). We denote by \( \Phi \) the resulting state space \([s_1, \ldots, s_n] \times I_1 \times \cdots \times I_k \). In addition, for denoting the derivatives of \( x_1, \ldots, x_k \), we assume variables \( \dot{x}_1, \ldots, \dot{x}_k \), ranging over \( \mathbb{R} \) each,\(^1\) and for denoting the targets of jumps, variables \( s', x_1', \ldots, x_k' \) ranging over \([s_1, \ldots, s_n]\) and \( I_1, \ldots, I_k \), correspondingly.

In order to describe hybrid systems, we use constraints that are arbitrary Boolean combinations of equalities and inequalities over terms that may contain function symbols like +, ×, exp, sin, and cos (which further function symbols might be allowed will become clear in Section 6). These constraints are used, on the one hand, to describe the possible flow and jumps and, on the other, to mark certain parts of the state space (e.g., the set of initial states).

**Definition 1.** A state space constraint is a constraint over the variables \( s, x_1, \ldots, x_k \). A flow constraint is a constraint over the variables \( s, x_1, \ldots, x_k, \dot{x}_1, \ldots, \dot{x}_k \). A jump constraint is a constraint over the variables \( s, x_1, \ldots, x_k \) and \( s', x_1', \ldots, x_k' \). A hybrid system description (or short: description) is a tuple consisting of a flow constraint, a jump constraint, a state space constraint describing the set of initial states, and a state space constraint, describing the set of unsafe states.

Example of a flow constraint for the case \( k = 2 \):

\[
\dot{x}_1 = x_1 - x_2 + 3 \land \dot{x}_2 = -x_1 - x_2 \land x_1 > 1 \land x_1 < 3 \land x_2 > 1 \land x_2 < 3
\]

Example of a flow constraint for the case \( n = 2 \) and \( k = 1 \):

\[
((s = s_1 \rightarrow \dot{x} = x) \land (s = s_2 \rightarrow \dot{x} = -x))
\]

Example of a jump constraint for the case \( n = 2 \) and \( k = 1 \):

\[
((s = s_1 \land x \geq 10) \rightarrow (s' = s_2 \land x' = 0))
\]

We use these constraints to describe the following:

**Definition 2.** A hybrid system is a tuple \((Flow, Jump, Init, UnSafe)\) where \(Flow \subseteq \Phi \times \mathbb{R}^k\), \(Jump \subseteq \Phi \times \Phi\), \(Init \subseteq \Phi\), and \(UnSafe \subseteq \Phi\).

A hybrid system description gives rise to the hybrid system, for which each constituting set is the solution set of the corresponding constraint of the hybrid system description.

**Definition 3.** A flow of length \( l \) in a mode \( s \) is a function \( r : [0, l] \mapsto \Phi\) such that the projection of \( r \) to its continuous part is differentiable, and for all \( t \in [0, l] \), the mode of \( r(t) \) is \( s \). A trajectory of a hybrid system

\(^1\)The dot does not have any special meaning here, and is just used for distinguishing dotted from undotted variables.
(\textit{Flow}, \textit{Jump}, \textit{Init}, \textit{UnSafe}) is a sequence of flows \(r_0, \ldots, r_p\) of lengths \(l_0, \ldots, l_p\) such that for all \(i \in \{0, \ldots, p\},\)

- if \(i > 0\) then \((r_{i-1}(l_{i-1}), r_i(0)) \in \textit{Jump},\)
- if \(l_i > 0\) then for all \(t \in [0, l_i], (r(t), \dot{r}(t)) \in \textit{Flow},\)

where \(\dot{r}\) is the derivative of the projection of \(r\) to its continuous part.

Note that this definition allows us to enforce jumps by formulating a flow constraint that does not allow continuous evolution in a certain region. Note also that our definition of trajectory prohibits the use of flow constraints to describe nondifferentiable evolution. For example, it is not possible to encode if-then-else constructions into the flow constraint as \((\phi \rightarrow \dot{x} = f(x)) \land (\neg \phi \rightarrow \dot{x} = g(x))\), because such a constraint can, in general, only be satisfied by nondifferentiable evolutions. However, one can encode the desired behavior using two different modes, for example, \(m_1\) and \(m_2\), a flow constraint \((s = m_1 \rightarrow \dot{x} = f(x)) \land (s = m_2 \rightarrow \dot{x} = g(x)),\) and a jump constraint \((\phi \land s = m_2 \rightarrow s' = m_1) \land (\neg \phi \land s = m_1 \rightarrow s' = m_2)\) for switching between the two modes based on the condition \(\phi.\)

**Definition 4.** A hybrid system \(H = (\textit{Flow}, \textit{Jump}, \textit{Init}, \textit{UnSafe})\) is safe if, and only if, there is no trajectory \(r_0, \ldots, r_p\) of \(H\) such that \(r_0(0)\) is in \(\textit{Init}\) and \(r_p(l)\) is in \(\textit{UnSafe},\) where \(l\) is the length of \(r_p.\)

We would like to have an algorithm that, given a hybrid system description, decides whether the corresponding system is safe. However, this is an undecidable problem [Henzinger et al. 1998]. Thus, we aim at an algorithm for which we know that, if it terminates, the hybrid system described by the input is safe. Moreover, we want it to terminate efficiently for problems of practical relevance.

### 3. AN INTERVAL-BASED METHOD

In this section, we describe an algorithm for verifying safety of hybrid systems. Basically, it is the result of taking a classical method for safety verification and putting it in an abstraction refinement framework. It seems that this classical method is in the folklore of the hybrid systems community and appears for the first time in the literature as a basis for a method that abstracts to timed automata [Stursberg et al. 1997]. It checks the flow at the boundary of boxes using interval arithmetic and requires that switching conditions, initial states, and unsafe states be aligned to the box grid. In contrast, our resulting algorithm allows these sets to be described by complex constraints, as introduced by Definition 1. We assume that we have an algorithm that can test such constraints for falsehood, that is, an algorithm that either returns “false” or “unknown.” For details on how to arrive at such an algorithm, see Section 6.

We abstract to systems of the following form:

**Definition 5.** A discrete system over a finite set \(S\) is a tuple \((\text{Trans}, \text{Init}, \text{UnSafe})\) where \(\text{Trans} \subseteq S \times S\) and \(\text{Init} \subseteq S, \text{UnSafe} \subseteq S.\) We call the set \(S\) the state space of the system.

In contrast to Definition 2, here the state space is a parameter. This will allow us to add new states to the state space during abstraction refinement.
Algorithm 1. Abstraction Refinement

let $A$ be a discrete abstraction of the hybrid system represented by a description $D$

while $A$ is not safe
    refine the abstraction $A$
end while

Definition 6. A trajectory of a discrete system $(Trans, Init, UnSafe)$ over a set $S$ is a function $r : \{0, \ldots, p\} \mapsto S$ such that for all $t \in \{1, \ldots, p\}$, $(r(t - 1), r(t)) \in Trans$. The system is safe iff there is no trajectory from an element of $Init$, to an element of $UnSafe$.

We will abstract the given hybrid system to a discrete system (the abstraction) in such a way that if the abstract system is safe, then the original (the concrete) system is also safe. If the current abstraction is not yet safe, we refine the abstraction, that is, we include more information about the concrete system into it. This results in Algorithm 1

In order to implement the above algorithm, we need to fix the state space of the abstract system. Here we use as abstract states pairs $(s, B)$, where $s$ is one of the modes $(s_1, \ldots, s_k)$ and $B$ is a hyperrectangle (box), representing subsets of the concrete state space $\Phi$. For the initial abstraction, we use the state space $\{(s_i, x) | (s_i, x) \in \Phi) | 1 \leq i \leq n\}$. When refining the abstraction, we split a box into two parts, creating two abstract states $(s, B_1)$ and $(s, B_2)$ with $B_1 \cup B_2 = B$, from an abstract state $(s, B)$.

In order to compute a discrete abstraction over this state space, we have to show how to compute the transitions of the resulting abstraction and its set of initial and unsafe states. Here we assume that the input consists of a hybrid system description with flow constraint $Flow(s, x, x')$, jump constraint $Jump(s, x, s', x')$, initial constraint $Init(s, x)$, and unsafety constraint $UnSafe(s, x)$. Now

- we mark an abstract state $(s, B)$ as initial iff we cannot disprove the constraint $\exists x \in B \ Init(s, x)$, and
- we mark an abstract state $(s, B)$ as unsafe iff we cannot disprove the constraint $\exists x \in B \ UnSafe(s, x)$.

In order to compute the possible transitions between two neighboring boxes in the same mode, we first consider the flow on common boundary points. For a box $B = [x_1, \ldots, x_k] \times \cdots \times [x_j, \ldots, x_k]$, we let its $j$th lower face be $[x_1, \ldots, x_{j-1}, x_j, \ldots, x_k]$ and its $j$th upper face be $[x_1, \ldots, x_{j-1}, x_j, \ldots, x_k]$. Two boxes are nonoverlapping if their interiors are disjoint.

Moreover, we say that a flow $r$ in a mode $s$ enters the box $B$ at a point $x \in B$ if, and only if, there exist a $t_0$ and a $\delta > 0$ such that $r(t_0) = (s, x)$ and for all $t \in (t_0, t_0 + \delta)$, $r(t) \in (s, B)$.

Lemma 1. For a mode $s$, and a box $B \subseteq \mathbb{R}^k$, if a flow in $s$ enters the box $B$ at a point $x$, then for all faces $F$ of $B$ such that $x \in F$,
∃\dot{x}_1, \ldots, \dot{x}_k [\text{Flow}(s, x, (\dot{x}_1, \ldots, \dot{x}_k)) \land \dot{x}_j \geq 0], \text{ if } F \text{ is the } j\text{th lower face of } B,

and

∃\dot{x}_1, \ldots, \dot{x}_k [\text{Flow}(s, x, (\dot{x}_1, \ldots, \dot{x}_k)) \land \dot{x}_j \leq 0], \text{ if } F \text{ is the } j\text{th upper face of } B

We denote the above constraint by \text{incoming}_{s, B}(x). Using this constraint, we can now construct a constraint for checking the possible transition between two boxes in the same mode.

**Lemma 2.** For a mode \(s\), two nonoverlapping boxes \(B, B' \subseteq \mathbb{R}^k\), if there is a flow in mode \(s\) that enters \(B'\) at a common point of \(B\) and \(B'\), then

\[
\exists x \in B \land x \in B' \land [\forall \text{faces } F \text{ of } B'[x \in F \Rightarrow \text{incoming}_{s, B}(x)]]
\]

This holds since Lemma 1 can be applied to all the faces of \(B\) that contain \(x\). We denote the corresponding constraint by transition_{s, B, B'}.

Now we introduce a transition from \((s, B)\) to \((s', B')\) iff

- \(s = s'\) and \(B = B'\), or
- \(s = s', B \neq B'\), and we cannot disprove transition_{s, B, B'} of Lemma 2, or
- we cannot disprove the constraint \(\exists x \in B \exists x' \in B' \text{Jump}(s, x, s', x')\)

Thus, given a hybrid system description \(D\) and a set \(B\) of abstract states (i.e., mode/box pairs) such that all boxes corresponding to the same mode are nonoverlapping, we have a method for computing the set of initial states, the set of unsafe states, and the transitions of a corresponding abstraction. We denote the resulting discrete system by Abstract_{D}(B).

**Theorem 1.** For all hybrid system descriptions \(D\) and sets of abstract states \(B\) covering the whole state space such that all boxes corresponding to the same mode are nonoverlapping, the safety of Abstract_{D}(B) implies the safety of the hybrid system described by \(D\).

This theorem will easily follow from a more general theorem, that we will prove in the next section.

If the differential equations in the flow constraint are in explicit form \(\dot{x} = f(x)\) then one can disprove the above constraints using interval arithmetic. According to Lemma 2 one can take all faces \(F\) of the common boundary of two boxes \(B\) and \(B'\), evaluate \(f\) on \(F\) using interval arithmetic, and check whether the resulting intervals have a sign that does not allow flows over the boundary—as described by Lemma 1. In Section 6, a method will be described that allows the flow constraints also to be in implicit form.

Now a concrete instantiation of Algorithm 1 can maintain the abstract state space \(B\) as described earlier, compute a corresponding abstract system Abstract_{D}(B), and (since this abstract system is finite) check its safety—either by a brute force algorithm or using a more sophisticated model-checking technology. We can either recompute the abstract system Abstract_{D}(B) each time we want to check its safety, or we can do this incrementally, just recomputing the elements corresponding to a changed element of the abstract state space (i.e., a box resulting from splitting).
4. A CONSTRAINT PROPAGATION-BASED IMPROVEMENT

The method introduced in the previous section has the problem that splitting can result in a huge number of boxes. This problem is especially virulent for high dimensions, since one needs a number of boxes that is exponential in the number of variables, to arrive at a box with a certain, small side length. In this section, we will try to find a remedy to this problem. The idea is to refine the abstraction without creating more boxes by splitting. Here we can use the observation that the unreachable state space is uninteresting and there is no need to include it in the abstraction.

Thus, we can exclude parts of the state space from the abstraction process, for which we can show that they are not reachable. In order to do this, we observe that a point in a box $B$ is reachable only if it is reachable either from the initial set via a flow in $B$, from a jump via a flow in $B$, or from a neighboring box via a flow in $B$.

We will now formulate constraints corresponding to each of these conditions. We then can remove points from boxes that do not fulfill at least one of these constraints. For this, we first give a constraint describing flows within boxes:

**Lemma 3.** For a box $B \subseteq \mathbb{R}^k$ and a mode $s$, if a point $y = (y_1, \ldots, y_k) \in B$ is reachable from a point $x = (x_1, \ldots, x_k) \in B$ via a flow in $B$ and $s$, then

$$\bigwedge_{1 \leq m < n \leq k} \exists a_1, \ldots, a_k, a_1, \ldots, a_k \in B \land \text{Flow}(s, (a_1, \ldots, a_k), (a_1, \ldots, a_k)) \land a_n \cdot (y_m - x_m) = a_m \cdot (y_n - x_n)$$

**Proof.** Assume that $r(t) = (r_1(t), \ldots, r_k(t))$ is a flow in $B$ from $(s, x)$ to $(s, y)$. So $r(0) = (s, x)$ and for a certain $t \in \mathbb{R}_{\geq 0}$, $r(t) = (s, y)$. Then, for $i, j \in \{1, \ldots, k\}$ arbitrary, but fixed, by the extended mean value theorem, we have:

$$\exists t' \in [0, t] \exists r_j(t') = r_j(t')(y_j - x_j)$$

Now choose such a $t'$ and let $(s, (a_1, \ldots, a_k)) = r(t')$ and $r(t') = (a_1, \ldots, a_k)$ and, hence, the whole constraint holds.

The intuition behind the above Lemma is shown in Figure 1, which shows that whenever we have a flow from a two-dimensional (2D) point $(x_n, x_m)$ to a 2-dimensional point $(y_n, y_m)$, then there must be a point on the flow where the vector field points exactly in the direction $(y_n - x_n, y_m - x_m)$. Therefore, the box must contain such a point.

We denote the above constraint by $flow_B(s, x, y)$. Now we can write down a constraint describing the first condition—reachability from the initial set:

**Lemma 4.** For a mode $s$ and a box $B \subseteq \mathbb{R}^k$, if $z \in B$ is reachable from the initial set via a flow in $s$ and $B$, then

$$\exists y \in B \text{Init}(s, y) \land flow_B(s, y, z)$$

The proof is trivial since it is an immediate consequence of Lemma 3. We denote the above constraint by $initflow_B(s, z)$.
We also have a constraint describing the second condition—reachability from a jump:

**Lemma 5.** For modes $s$ and $s'$, boxes $B, B' \subseteq \mathbb{R}^k$, and $z \in B'$, if $(s', z)$ is reachable from a jump from $(s, B)$ via a flow in $B'$ and $s'$, then

$$\exists x \in B \exists x' \in B'[Jump(s, x, s', x') \land \text{flow}_{B'}(s', x', z)]$$

The proof is trivial since it is also a consequence of Lemma 3. We denote the above constraint by $\text{jumpflow}_{B, B'}(s, s', z)$.

Finally, we strengthen the condition mentioned in Lemma 2 to a constraint describing the third condition—reachability from a neighboring box.

**Lemma 6.** For a mode $s$ and nonoverlapping boxes $B, B' \subseteq \mathbb{R}^k$, if $z \in B' \setminus B$ is reachable via a flow in $s$ that enters $B'$ at a common point of $B$ and $B'$, then

$$\exists x \in B \land x \in B' \land [\forall \text{ faces } F \text{ of } B[x \in F \Rightarrow \text{incoming}_{s, B}(x)] \land \text{flow}_{B}(s, x, z)]$$

This is a consequence of Lemmas 3 and 2. We denote the above constraint by $\text{boundaryflow}_{B, B'}(s, z)$.

Now a point that is only covered by one abstract state is reachable only if it is reachable according to Lemmas 4, 5, or 6:

**Theorem 2.** For a set of abstract states $B$, a pair $(s', B') \in B$ and a point $z \in B'$, if $(s', z)$ is reachable and $z$ is not an element of the box of any other abstract state in $B$, then

$$\text{initflow}_{B'}(s', z) \lor \bigvee_{(s, B) \in B} \text{jumpflow}_{B, B'}(s, s', z)$$

$$\lor \bigvee_{(s, B) \in B, s = s', B \neq B'} \text{boundaryflow}_{B, B'}(s', z)$$

We denote this constraint by $\text{reachable}_{B, B'}(s', z)$. Now, if we can prove that a certain point $z$ in $B'$ does not fulfill this constraint, we can remove it from the box $B'$ (if the point is also covered by another abstract state, we can remove it in any case). For now we assume that we have an algorithm (a pruning algorithm) that takes such a constraint, and an abstract state $(s', B')$ and returns a sub-box.
of $B'$ that still contains all the solutions of the constraint in $B'$. (See Section 6 for details on such algorithms.)

Since the constraint $\text{reachable}_{B,B'}(s',z)$ depends on all current abstract states, a change of $B'$ might allow further pruning of other abstract states. Thus, we can repeat pruning until a fixpoint is reached. This terminates since we use floating-point computation and there are only finitely many floating point numbers. Given a set of abstract states $B$, we denote the resulting fixpoint by $\text{Prune}_D(B)$.

Now, since we do not need to consider unreachable parts of the state space in the abstraction, we can do the operation $B \leftarrow \text{Prune}_D(B)$ anywhere in Algorithm 1. We do this at the beginning, and each time $B$ is refined by splitting a box.

Hence, our method can, in some cases, refine the abstraction without splitting, which is a remedy for the problem that splitting may produce a large number of boxes, especially for high dimensions (i.e., for the curse of dimensionality). For doing so, the method considers the flow not only on the boundary but also inside of the boxes.

Now observe that in the computation of $\text{Abstract}_D(B)$, we check whether one abstract state is reachable from another one. This information has already been computed by $\text{Prune}_D(B)$. More precisely, we get this information from the individual disjuncts of Theorem 2 and we do not need to recompute it.

We get the correctness of our method from the following modification of Theorem 1:

**Theorem 3.** For all hybrid system descriptions $D$ and sets of abstract states $B$, containing all elements of the state space reachable from the initial set such that all boxes corresponding to the same mode are nonoverlapping, the safety of $\text{Abstract}_D(B)$ implies the safety of the hybrid system described by $D$.

**Proof.** Let $D$ be an arbitrary, but fixed hybrid system description, and let $B$ be an arbitrary, but fixed set of abstract states containing all elements of the state space reachable from the initial set such that all boxes corresponding to the same mode are nonoverlapping. Let $H = (\text{Flow}_1, \text{Jump}_1, \text{Init}_1, \text{Unsafe}_1)$ be the hybrid system described by $D$ and let $(\text{Trans}_2, \text{Init}_2, \text{Unsafe}_2) = \text{Abstract}_D(B)$. We will prove that every concrete trajectory from an element of $\text{Init}_1$ to $\text{Unsafe}_1$ has a corresponding abstract trajectory from $\text{Init}_2$ to $\text{Unsafe}_2$.

Now let $r_0, \ldots, r_p$ be a concrete trajectory from an element in $\text{Init}_1$ to $\text{Unsafe}_1$. Let $l_0, \ldots, l_p$ be the corresponding lengths of the flows $r_0, \ldots, r_p$. Let $r_0^a, \ldots, r_p^a$ be such that for all $i \in \{0, \ldots, p\}$, $r_i^a : [0, l_i] \rightarrow B$ and for all $t \in [0, l_i]$, $r_i^a(t)$ is an abstract state $(s, B)$ such that $s$ is equal to the mode of $r_i(t)$ and $B$ contains the continuous part of $r_i^a(t)$. Such an $(s, B)$ always exists, since $B$ contains the states reachable from the initial set. Moreover, if for a certain $t$, there is a $\delta > 0$ such that for all $t' \in [t, t + \delta]$, $r_i(t')$ is covered by the same abstract state, then we assign the same abstract state to $r_i^a(t')$.

We construct for every $i \in \{0, \ldots, p\}$ a discrete abstract trajectory from $r_i^a(0)$ to $r_i^a(l_i)$. For this, for a function $f : [0, l_i] \rightarrow B$ and abstract state $a$, let $F^a$ be such that $F^a(f)$ (the flattening of $f$ wrt. $a$) is again a function in $[0, l_i] \rightarrow B$
and such that for all $t \in [0, l_i]$, $F^a(\rho)(t) =$
- $a$, if there exist $t^-$ and $t^+$ in $[0, l_i]$ such that $t^- < t < t^+$ and such that $f(t^-) = a$ and $f(t^+) = a$;
- $f(t)$, otherwise.

Let $\rho^a_i$ be the result of taking the flattening of $r^a_i$ wrt. every abstract state $a \in B$, in an arbitrary order, that is, for $B = \{a_1, \ldots, a_p\}$, $\rho^a_i = (F^{a_1} \circ \cdots \circ F^{a_p})(r^a_i)$.

Then $\rho^a_i$ changes its value only finitely often (if it would change its value infinitely often, there would be $t, t^-, t^+ \in [0, l]$ such that $t^- < t < t^+$ and $\rho^a_i(t^-) = \rho^a_i(t^+) \neq \rho^a_i(t)$, which contradicts the fact that $\rho^a_i$ is flattened).

Hence, every time $\rho^a_i$ changes its value from an abstract state $(s, B)$, to an abstract state $(s, B')$, there is a corresponding flow that enters $B'$ at a common point of $B$ and $B'$, and because of Lemma 2, $\text{trans}_{s, B, B'}$ holds. Hence, by definition of $\text{Abstract}_D(B), ((s, B), (s, B')) \in Trans_2$. This allows the construction of a finite sequence of abstract states that forms a trajectory of $\text{Abstract}_D(B)$ from $r^a_i(0)$ to $r^a_i(l_i)$.

Moreover, for every $i \in \{1, \ldots, p\}$, $(r_{i-1}(l_{i-1}), r_i(0)) \in \text{Jump}_1$. Hence, the jump constraint holds on this pair, and by definition of $\text{Abstract}_D(B)$, $(r_{i-1}(l_{i-1}), r_i(0)) \in Trans_2$. So we can concatenate the discrete trajectories from $r^a_i(0)$ to $r^a_i(l_i)$ into a discrete trajectory from $\text{Init}_2$ to $\text{Unsafe}_2$.

Thus, the safety of $\text{Abstract}_D(B)$ implies the safety of $D$. $\square$

The fact that pruning removes elements from the boxes forming the abstract states has the consequence that in certain cases these boxes do not contain any unsafe states any more. This will result in an abstraction that does not have any unsafe abstract states left. In such a case, the abstract system is trivially safe which immediately implies safety of the concrete system. However, pruning will not be able to remove unsafe states from boxes that intersect with a cycle of the hybrid system. In order to eventually be able to remove the corresponding cycle in the abstraction, we always remove abstract states that are not reachable in the abstraction.

5. INCREMENTAL COMPUTATION

The approach as described up to now applies the algorithm $\text{Prune}_D$, from scratch, in each turn of the main loop. In this section, we describe how this can be avoided by reusing information from one call to the next.

First, observe that for a given abstract state $(s, B)$ our solver might prove that one of the disjuncts of the constraint $\text{reachable}_{s, B}$ of Theorem 2 has no solution in $(s, B)$. For example, this is trivially the case for $\text{boundaryflow}$ and nonneighboring boxes. In such a case we can remove the corresponding disjunct from the disjunction and we do not have to check it again for this abstract state or any abstract state that contains a box that results from $B$ by splitting.

We call the resulting method $\text{weakly incremental}$. Since our implementation does not keep separate copies of the constraint $\text{reachable}_{s, B}(s, z)$ for each abstract state $(s, B)$, we cannot physically remove the disjuncts. Instead we store two types of transitions in the abstraction: boundary transitions and jump transitions. If there is no such transition, we know that the corresponding disjunct is empty.

Algorithm 2. Strongly Incremental Pruning

Input: \( B_n, \) a set of abstract states (i.e., mode/box pairs),
\( Q \) such that
\( Q \subseteq B_n, \)
for all \( (s, B) \in B_n \setminus Q, \) \( \text{prune}_{B_n}(s, B) = B \)

Output: \( B_n \) such that
\( B \) contains all reachable elements of \( B_n, \)
for all \( (s, B) \in B, \) \( \text{prune}_{B_n}(s, B) = B. \)

\[
B \leftarrow B_n
\]
while \( Q \neq \emptyset \)
let \((s, B)\) be an element of \( Q \)
\( B' \leftarrow \text{prune}_{B_n}(s, B) \)
if \( B' \neq B \)
\( Q \leftarrow Q \cup \{(s', B') \in B \mid (s', B') \text{ depends on } (s, B)\} \)
\( B \leftarrow (B \setminus (s, B)) \cup \{(s, B')\} \)
end if
\( Q \leftarrow Q \setminus (s, B) \)
end while

Now observe that, after we remove some disjuncts to achieve weak incrementality, the constraint \( \text{reachable}_{B_n}(s, z) \) only depends on some, but not necessarily all, other abstract states in \( B \) and we only have to recompute it, if one of these changed.

The resulting reimplementation of \( \text{Prune}_D \), is shown as Algorithm 2. It uses a subalgorithm \( \text{prune}_{B_n}(s, B) \), which, given a set of abstract states \( B \) and an element \((s, B) \in B\), returns the result of pruning a single abstract state \((s, B)\) using the constraint \( \text{reachable}_{B_n}(s, z) \). We say that \((s', B')\) depends on another abstract state \((s, B)\) if the constraint \( \text{reachable}_{B_n}(s', z)\) still contains a disjunct referring to \((s, B)\). The algorithm maintains a set of abstract states for which pruning might succeed and applies the pruning algorithm only to those. If an abstract state is changed, it reconsider other abstract states that depend on the changed one.

When calling Algorithm 2, we pass as \( Q \) all the abstract states that we might be able to prune. These are the abstract states that contain a box that has been split or that depend on another abstract state whose box has been split.

This algorithm can be viewed as an adaption of the constraint propagation algorithm AC-3 [Mackworth 1977; Apt 1999] to our purpose. The main difference is that AC-3 stores constraints in the set \( Q \), while we store abstract states.

We call the resulting method \textit{strongly incremental}. Currently, we implement the set \( Q \) by marking the abstract states that are in \( Q \) with a Boolean value. This is a simplification from the usual implementations where \( Q \) is implemented as a FIFO-queue. This simplification has the following consequences:

- Searching for the next box to prune is not done in constant time, since we have to search for marked abstract states (but, in practice, this might be more efficient than enqueuing/dequeuing).
- The pruning order is different from the usual one. Especially, this order does not take into account the dependencies between the constraints.
As usual in continuous domains, in order to avoid slow convergence, in the implementation of the algorithm, we do not include all the constraints with changed variables into the set $Q$, but only the ones for which a variable changed more than a certain threshold. Thus, we do not compute a fixpoint, but only an almost fixpoint.

6. CONSTRAINT SOLVING

In this section, we discuss how the algorithm used for pruning the abstract states with respect to constraints (i.e., the subalgorithm $\text{prune}_{D,p}(s, B)$ of Algorithm 2) can be implemented. Such pruning algorithms are one of the main topics of the area of constraint programming (for more information see http://slash.math.unipd.it/cp/). Usually these work on conjunctions of atomic constraints over a certain domain. For the domain of the real numbers, given a constraint $\phi$ and a floating-point box $B$, they compute another floating-point box $N(\phi, B)$, such that $N(\phi, B) \subseteq B$ (contractance) and $N(\phi, B)$ contains all solutions of $\phi$ in $B$ (cf. the notion of narrowing operator [Benhamou et al. 1994; Benhamou 1996], sometimes also called contractor).

There are several methods for implementing such a pruning algorithm. The most basic method [Davis 1987; Cleary 1987; Benhamou and Older 1997] decomposes all atomic constraints (i.e., constraints of the form $t \geq 0$ or $t = 0$, where $t$ is a term) into conjunctions of so-called primitive constraints (i.e., constraints such as $x + y = z$, $xy = z$, $z \in [a, b]$, or $z \geq 0$) by introducing additional auxiliary variables (e.g., decomposing $x + \sin y \geq 0$ to $\sin y = v_1 \land x + v_1 = v_2 \land v_2 \geq 0$). It then applies a pruning algorithm for these primitive constraints [Hickey et al. 1998] until a fixpoint is reached.

We illustrate pruning of primitive constraints using the example of a primitive constraint $x + y = z$ with the intervals $[1, 4]$, $[2, 3]$, and $[0, 5]$ for $x$, $y$, and $z$, respectively. We can solve the primitive constraint for each of the free variables, arriving at $x = z - y$, $y = z - x$, and $z = x + y$. Each of these forms allows us to prune the interval associated with the variable on the left-hand side of the equation: Using the first solved form, we subtract the interval $[2, 3]$ for $y$ from the interval $[0, 5]$ for $z$, concluding that $x$ can only be in $[−3, 3]$. Intersecting this interval with the original interval $[1, 4]$, we know that $x$ can only be in $[1, 3]$. Proceeding in a similar way for the solved form $y = z - x$ does not change any interval, and, finally, using the solved form $z = x + y$, we can conclude that $z$ can only be in $[3, 5]$.

Pruning for other primitive constraints can be based on interval arithmetic in a similar way. There is extensive literature [Neumaier 1990; Hickey et al. 2001] providing precise formulas for interval arithmetic for addition, subtraction, multiplication, division, and the most common transcendental functions. The floating-point results are always rounded outward, such that the result remains correct also under rounding errors.

There are several variants, alternatives, and improvements of the basic approach described above [Jaulin et al. 2001; Benhamou et al. 1994; Lhomme 1993; Lhomme et al. 1998; Hickey 2001; Lebbah et al. 2002].
For applying such an algorithm to the constraints occurring in our method, we first eliminate the defined predicates by substituting the constraints implied by their definitions. Moreover, the constraints contain variables $s$ and $s'$, ranging over a finite set. These can be easily eliminated by a trivial substitution and simplification.

The constraints also contain existential quantifiers. These can be treated by simply dropping them from the constraints and pruning the Cartesian product of the box corresponding to the free variables and the box bounding the quantified variables [Ratschan 2002]. For disjunctions, one can prune the disjuncts and take the union of the result [Ratschan 2002].

Still, our constraints are not first-order, since they also have box-valued arguments (appearing in the subscripts of the names of constraints) for use with the set-theoretic predicate $\in$. One could eliminate these arguments by substituting the corresponding boxes into the constraints. However, this would require reparsing the constraint each time the corresponding box changed. Instead, we eliminate these boxes as follows: We first rewrite a constraint $v \in B$, with $B = I_1 \times \cdots \times I_k$ to a conjunction $v_1 \in I_1 \land \cdots \land v_k \in I_k$. We then introduce the meta-constraint $x \subseteq y$ [Older and Benhamou 1993; Hickey 2000] that models the information that the solution set in the variable $x$ is a subset of the solution set in the variable $y$. Using this constraint, we can now replace a constraint $v_i \in I_i$ by the metaconstraint $v_i \subseteq b_i$, where $b_i$ is a new variable for which we pass the corresponding interval $I_i$. A solver can then use this metaconstraint to prune the variable $v_i$ to this interval. For example, in the case of a two-dimensional state space, the result for $\text{flow}_{B}(s, (x_1, x_2), (y_1, y_2))$ has the form $\exists a_1, a_2, a_3(a_1 \leq b_1 \land a_2 \leq b_2 \land \cdots \land a_3(y_1 - x_1) = a_3(y_2 - x_2))$, such that for $B = I_1 \times I_2$, we pass to the pruning function the interval $I_1$ for the new variable $b_1$, and the interval $I_2$ for the new variable $b_2$.

We have implemented the algorithm on top of our RSOLVER [Ratschan 2004] package that provides pruning and solving of quantified constraints of the real numbers, a graphical user interface, and several other features, and that uses the smathlib library [Hickey] for pruning primitive constraints. The implementation is publically available [Ratschan and She 2004] and we will make the source code open, which will make it easy to extend it or to experiment with changes.

7. COMPUTATION RESULTS

In our opinion, it is essential to test new algorithms on more than just two to three examples. Hence, we devote this section to extensive benchmarking. For this we have to fix a certain splitting strategy in the method. Intuitively—in order to achieve some sort of convergence—one wants to require that the side length of all boxes eventually goes to zero. In fact, this is precisely the condition under which we could prove convergence in the discrete time case in another paper [Damm et al. 2005]. We always split the widest box along the widest side length. However, this strategy has the disadvantage that it depends on the units used for the individual variables. Especially, if the state space is much larger in one variable than in others, then only that variable will be split.
Hence, we also tried a round-robin strategy, where the widest box is split along the variable along, which it has not been split for the longest time. We call these two strategies as Widest and RoundRobin, respectively.

**Example FOCUS:**
Flow: \((x_1, x_2) = (x_1 - x_2, x_1 + x_2)\)
Empty jump relation
Init: \(2.5 \leq x_1 \leq 3 \land x_2 = 0\)
Unsafe: \(x_1 \leq 2\)
The state space: \([0, 4] \times [0, 4]\)

**Example 2-TANKS:** The flow constraint results from setting all the parameters in the two tanks problem [Stursberg et al. 1997] to 1.
Flow: \((s = 1 \rightarrow (x_{1, 1}) = (\frac{1 - \sqrt{2} - \sqrt{2}}{\sqrt{2} - \sqrt{2}})) \land (s = 2 \rightarrow (x_{1, 2}) = (\frac{1 - \sqrt{3} + 3}{\sqrt{3} + 3}))\)
Jump: \((s = 1 \land 0.99 \leq x_2 \leq 1) \rightarrow (s' = 2 \land x_1' = x_1 \land x_2' = 1)\)
Init: \((s = 1 \land (x_1 - 5.5)^2 + (x_2 - 0.25)^2 \leq 0.0625)\)
Unsafe: \((s = 1 \land (x_1 - (x_2)^2) + (x_2 - 0.25)^2 \leq 0.0625)\)
The state space: \((1, [0, 1]) \cup (2, [4, 6] \times [1, 2])\)

**Example ECO:** A predator–prey example of ecosystem problems.
Flow: \((s = 1 \rightarrow (x_{1, 1}) = (-x_1 + x_1 x_2) \land (s = 2 \rightarrow (x_{1, 2}) = (-x_1 + x_1 x_2))\)
Jump: \((s = 1 \land 0.875 \leq x_2 \leq 0.9) \rightarrow (s' = 2 \land (x_1' - 1.2)^2 + (x_2' - 1.8)^2 \leq 0.01)\)
\(\lor ((s = 2 \land 1.1 \leq x_2 \leq 1.125) \rightarrow (s' = 1 \land (x_1' - 0.7)^2 + (x_2' - 0.7)^2 \leq 0.01))\)
Init: \((s = 1 \land (x_1 - 0.8)^2 + (x_2 - 0.2)^2 \leq 0.01)\)
Unsafe: \((s = 1 \land x_1 > 0.8 \land x_2 > 0.8 \land x_1 <= 0.9 \land x_2 \leq 0.9)\)
The state space: \((1, [0, 0.9]) \times [0, 1.9]) \cup (2, [1, 1.9] \times [1, 1.9])\)

**Example 1-FLOW:** An example from a paper by J. Preussig [Preussig et al. 1998].
Flow: \(\dot{x} = y = t = 1\)
Empty jump relation
Init: \(0 \leq x \leq 1 \land y = t = 0\)
Unsafe: \((0 \leq x \leq 2 \land 1 < y < 2 \land 0 \leq t < 1)\)
The state space: \([0, 2] \times [0, 2] \times [0, 4]\)

**Example CLOCK:** A simple example with a clock variable.
Flow: \((\dot{x}, \dot{y}, t) = (-5.5 y + y^2, 6 x - x^2, 1)\)
Empty jump relation
Init: \(4 \leq x \leq 4.5 \land y = 1 \land t = 0\)
Unsafe: \((1 \leq x < 2 \land 2 < y < 3 \land 2 \leq t \leq 4)\)
The state space: \([1, 5] \times [1, 5] \times [0, 4]\)

**Example CAR:** A three-dimensional and nonlinear example about a simple controller that steers a car along a straight road from a paper by Clarke and others [Clarke et al. 2003a].
The three continuous variables are the position $x$, the heading angle $\gamma$, and the internal timer $c$. Since we cannot prove the safety property described in the original paper, in this paper we set the unsafe space to $x \leq -4$.

**Example HEATING:** A linear, three-dimensional example of the room heating problem defined by three rooms and two heaters from a paper by A. Fehnker and F. Ivančić [2004].

In the original paper, the authors require that the temperature in all rooms always be above a given threshold. However, this given threshold is not specified. In this paper, we specify it to be 14. In addition, we specify the state space to be $[14, 22] \times [14, 22] \times [14, 22]$.

It is currently unknown whether this system is safe or not.

**Example CONVOI:** A linear collision avoidance example from a part of the car convoi control from a paper by A. Puri and P. Varaiya [1995].

Let $\text{gap}$, $v_r$, $v_l$ and $a_r$, respectively, represent the distance between the two cars $(d_{i-1} - d_i$ in the original paper), the velocity of the rear car $(d_{i})$, the velocity of the leading car $(d_{i-1})$, and the acceleration of the rear car $(d_{i})$. By using these variables, and restricting $v_l$ by $-2 \leq v_l \leq -0.5$, we transformed the original higher-order differential equation into a four-dimensional differential (in)equation of order one.

We set the state space to $[0, 4] \times [0, 2] \times [0, 2] \times [-2, -0.5]$ and we want to verify that $\text{gap} > 0$ when starting from $\text{gap} = 1, v_r = 2, v_l = 2$, and $a_r = -0.5$.

**Example MIXING:** A four-dimensional and nonlinear example about a mixing-tank-system from a paper by O. Stursberg, S. Kowalewski, and S. Engell [Stursberg et al. 2000].

In the original paper, the system is simplified to a two-dimensional system. In this paper, we keep the differential equations $(V_1, V_2) = (0.008, 0.015)$ in the flow constraint, where $V_1$ and $V_2$ are two inlet streams. Then, initially, $V_1(0) = 1, V_2(0) = 1$, and $(h(0), c(0)) \in [1.32, 1.5] \times [1.2, 1.32]$, where $h$ is liquid height and $c$ is concentration. We want to verify that the state $(V_1, V_2, h, c) : h \in [1.1, 1.3] \land c \in [1.68, 1.80]$ is unreachable.

**Example CIRCUIT:** A two-dimensional and nonlinear example about a tunnel-diode oscillator circuit [Frehse 2005]. It models the voltage drop $V$ and current $I$.

The original problem was to prove that all trajectories eventually reach a certain set and stay there. We transformed it to a reachability problem, using the state space $[-0.1, 0.6] \times [-0.002, 0.002]$ and the unsafety constraint $V < -0.04 \lor V > 0.54 \lor I < -0.0015 \lor I > 0.00175$.

We tried to verify the above examples by applying our improved method from Section 4 and comparing the two splitting techniques mentioned above, and the basic unincremental (Table I), the weakly incremental (Table II), and the strongly incremental versions (Table III) described in Section 5. The computations were performed on a Pentium IV, 2.60-GHz with 1 GB RAM; and they were canceled in cases when computation did not terminate before 10 hr of computation time. We also list the number of splitting steps (which corresponds to the number of loop turns in Algorithm 1) and the number of calls to
the pruning algorithm. In all cases, memory consumption was too small to be of interest.

8. DISCUSSION AND IMPROVEMENT

Note that, to our knowledge, there is no rigorous (i.e., not floating-point error prone) system available that allows similar generality as our method (e.g.,

### Table I. Direct Computation

<table>
<thead>
<tr>
<th>Example</th>
<th>Widest</th>
<th>Round-Robin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU time (s)</td>
<td>Splitting step</td>
</tr>
<tr>
<td>FOCUS</td>
<td>11.58</td>
<td>60</td>
</tr>
<tr>
<td>2-TANKS</td>
<td>1.43</td>
<td>18</td>
</tr>
<tr>
<td>ECO</td>
<td>5233</td>
<td>452</td>
</tr>
<tr>
<td>1-FLOW</td>
<td>0.95</td>
<td>1</td>
</tr>
<tr>
<td>CLOCK</td>
<td>62.25</td>
<td>183</td>
</tr>
<tr>
<td>CAR</td>
<td>0.52</td>
<td>3</td>
</tr>
<tr>
<td>HEATING</td>
<td>&gt;10 hr</td>
<td></td>
</tr>
<tr>
<td>CONVOI</td>
<td>65392</td>
<td>415</td>
</tr>
<tr>
<td>MIXING</td>
<td>7798</td>
<td>468</td>
</tr>
<tr>
<td>CIRCUIT</td>
<td>&gt;10 hr</td>
<td></td>
</tr>
</tbody>
</table>

### Table II. Weakly Incremental

<table>
<thead>
<tr>
<th>Example</th>
<th>Widest</th>
<th>Round-Robin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU time (s)</td>
<td>Splitting step</td>
</tr>
<tr>
<td>FOCUS</td>
<td>1.27</td>
<td>60</td>
</tr>
<tr>
<td>2-TANKS</td>
<td>0.33</td>
<td>18</td>
</tr>
<tr>
<td>ECO</td>
<td>2.32</td>
<td>72</td>
</tr>
<tr>
<td>1-FLOW</td>
<td>0.05</td>
<td>1</td>
</tr>
<tr>
<td>CLOCK</td>
<td>62.25</td>
<td>183</td>
</tr>
<tr>
<td>CAR</td>
<td>0.38</td>
<td>3</td>
</tr>
<tr>
<td>HEATING</td>
<td>&gt;10 hr</td>
<td></td>
</tr>
<tr>
<td>CONVOI</td>
<td>3165</td>
<td>415</td>
</tr>
<tr>
<td>MIXING</td>
<td>2514</td>
<td>468</td>
</tr>
<tr>
<td>CIRCUIT</td>
<td>&gt;10 hr</td>
<td></td>
</tr>
</tbody>
</table>

### Table III. Strongly Incremental

<table>
<thead>
<tr>
<th>Example</th>
<th>Widest</th>
<th>Round-Robin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU time (s)</td>
<td>Splitting step</td>
</tr>
<tr>
<td>FOCUS</td>
<td>0.71</td>
<td>107</td>
</tr>
<tr>
<td>2-TANKS</td>
<td>0.14</td>
<td>18</td>
</tr>
<tr>
<td>ECO</td>
<td>0.44</td>
<td>72</td>
</tr>
<tr>
<td>1-FLOW</td>
<td>0.05</td>
<td>1</td>
</tr>
<tr>
<td>CLOCK</td>
<td>7.96</td>
<td>183</td>
</tr>
<tr>
<td>CAR</td>
<td>0.34</td>
<td>3</td>
</tr>
<tr>
<td>HEATING</td>
<td>&gt;10 hr</td>
<td></td>
</tr>
<tr>
<td>CONVOI</td>
<td>203.74</td>
<td>415</td>
</tr>
<tr>
<td>MIXING</td>
<td>169.88</td>
<td>472</td>
</tr>
<tr>
<td>CIRCUIT</td>
<td>&gt;10 hr</td>
<td></td>
</tr>
</tbody>
</table>
nonlinear system specification, even with transcendental functions). Moreover, most current systems only work reasonably well with human interaction for specifying certain parameters, whereas we run a full benchmark suite without any interaction. The results clearly show that the efficiency of our method is already similar to systems that do not have the same generality and rigor as our method and are often based on years of development effort. For the example HEATING, whose safety is unknown, our method also does not succeed.

One can conclude that incremental computation definitely pays off in terms of number of calls to the pruning function, where it results in an improvement of orders of magnitude. This is also reflected in the amount of CPU time spent. Also, the splitting strategy $\text{Widest}$ behaves as expected—it has problems for examples where the state space of different variables has highly different magnitude, as is the case for the example CIRCUIT. By avoiding this phenomenon, the round-robin splitting strategy is significantly more efficient, on average.

However, the example 1-FLOW shows an anomaly: It can be easily solved by the strategy $\text{Widest}$, but the round-robin strategy fails. Here a phenomenon occurs that can be more easily explained on the following simpler example:

Flow: $\dot{x} = \dot{y} = 1,$
Init: $x = y = 0,$
State space: $[0, 4] \times [0, 4]$

Assume that we have an abstract state given by the white box in Figure 2. The neighboring boxes then continue to consider flows from the whole boundary of the white box (the dotted arrows), although there is only one flow from the upper right corner. Here a well-known inherent problem of abstraction refinement techniques occurs: the introduction of additional transitivity along a series of abstract states, that is, given abstract states $a$, $b$, and $c$, if there are (nonspurious) transitions $a \rightarrow b$, and $b \rightarrow c$, then $c$ is declared to be reachable from $a$, although this is not necessarily the case. Sometimes not even refinement of the abstraction can remove this problem (one would have to check whole counterexample paths to avoid this [Clarke et al. 2003a; Alur et al. 2003]). This is different from the discrete time case, where one can show convergence of a
similar abstraction refinement scheme [Damm et al. 2005]. However, in this example, for boxes that are not square, pruning can remove some parts (the grey part in the figure can be removed when starting from the bigger nonsquare box covering both the white and grey area). In the example, 1-FLOW round-robin splitting happens to choose a sequence of variables for splitting, where pruning does not help to avoid this transitivity problem.

Another limitation of the method can be seen on the following example:

Flow: \((\dot{x}, \dot{y}) = (1, \sqrt{2})\)
Init: \((x, y) = (0, 0)\),
Unsafe: \(y < \sqrt{2}x\),
State space: \([0, 2] \times [0, 2]\)

Obviously, this system is safe. However, if the flow constraint is perturbed to the constraint Flow: \((\dot{x}, \dot{y}) = (1, 1.414)\) then the resulting system is unsafe. This is because the original system is not robustly safe in the sense that under a small perturbation the perturbed system is unsafe. For such systems, the method will not terminate. However, this behavior is desired! We clearly do not want to declare systems as safe that become unsafe under small perturbations, since the implemented system will never be able to exactly match the verified model [Fränzle 1999; Damm et al. 2005; Ratschan 2002].

The fact that for some examples a high number of splitting steps is necessary, motivated us to experiment with an alternative splitting strategy. Here we use the observation that in multimode systems, very often, there is only a small amount of interaction between the modes and, hence, only a small number of abstract transitions between abstract states in different modes. As a consequence, splitting a certain box in one mode will usually allow only a small amount of pruning in the other modes. Thus, in order to encourage pruning in all modes, instead of splitting only one box in total, we also tried to split one box (again the largest one) in each mode. Clearly this is different from the old strategy only for multimode systems. The result can be seen in Tables IV, V, and VI—a clear improvement over the previous strategy.

**9. RELATED WORK**

The idea of using abstraction to compute the reach set of hybrid systems is not new. Here the basic choice is, which data structure to use for representing subsets of the continuous part of the state space.
Kowalewski, Stursberg, and co-workers pioneered the use of box representations [Stursberg et al. 1997; Stursberg and Kowalewski 2000; Preussig et al. 1998; Stursberg and Kowalewski 1999; Preussig et al. 1999; Stursberg et al. 2000]. Also, in their method, interval arithmetic is used to check the flow on the boundaries of a rectangular grid. Timing information is then added by checking the flow within these boxes. As a result, one arrives at rectangular or timed automata. All appearing switching conditions, initial states, and unsafe states have to be aligned to the predefined grid, whereas in this paper, we allow complex constraints. Moreover, their method has been designed for a fixed grid and a refinement of the abstraction requires a complete recomputation, whereas in the present work, this can be done incrementally. Their method also does not include a step for refining the abstraction without splitting and it is harder to implement, since it does not build upon an existing constraint solver. However, they generate additional timing information and use additional information on reachable subsets of faces.

Another frequently used technique for representing parts of the state space are polyhedra [Chutinan and Krogh 1999; Alur et al. 2002; Clarke et al. 2003a; Alur et al. 2003; Asarin et al. 2002; Frehse 2005]. This has the advantage of being flexible, but requires involved algorithms for handling these polyhedra and for approximating reachable sets. In contrast to that, boxes are less flexible, but the corresponding operations are simple to implement efficiently, even with validated handling of floating-point rounding errors. It is not clear how one could adapt the pruning mechanism of this paper to polyhedra.

Another method uses semialgebraic sets for representation [Tiwari and Khanna 2002]. This is even more flexible and can produce symbolic output, but requires highly complex quantifier elimination tools [Collins and Hong 1991]. Again, it is not clear how one could employ a pruning mechanism for such a representation.
Other methods try to compute the reach set explicitly, without abstraction. Also, here one has to decide on the representation for this set, for example, zonotopes [Girard 2005], polytopes [Chutinan and Krogh 1999], orthogonal polyhedra [Asarin et al. 2002], ellipsoids [Kurzhanski and Varaiya 2000], or level sets [Mitchell and Tomlin 2000].

There are also methods that use interval arithmetic to compute the reach set explicitly. In one approach [Henzinger et al. 2000], an interval ODE solver is used. In another approach [Hickey and Wittenberg 2004], a constraint logic programming language [Hickey 2000] that allows constraints with differentiation operators is used.

In general, abstraction refinement methods, such as the one described in this paper, have the advantage that they avoid computation of information that is not necessary for the safety checking problem at hand. On the other hand, for cases where this safety checking problem does require a very precise computation of the reach set, a direct computation for this reach set might be more efficient.

10. CONCLUSION
In this paper, we have put a classical method for verifying safety of hybrid systems into an abstraction refinement framework, and we have provided a constraint propagation based remedy for some of the problems of the method. As a result, we need to split into less boxes, we retain information on the flow within boxes, and we can use complex constraints for specifying the hybrid system. Since the method is based on a clear set of constraints, it can be easily implemented using a pruning algorithm based on interval constraint propagation.

Our long-term goal is to arrive at a method for which one can prove termination for all, except numerically, ill-posed cases, in a similar way as can be done for quantified inequality constraints [Ratschan 2002] and hybrid systems in which all trajectories follow polynomials [Fränzle 1999]. Moreover, we will try to exploit the structure of special, for example, linear systems [Tiwari 2003], and analyze whole sequences of abstract states (instead of just pairs) to avoid the transitivity problem.

Interesting further questions are, whether work on constraint propagation in the discrete domain can be useful in a similar way for pruning the discrete state space, and whether similar pruning of the state space can be done for more complex verification tasks, (i.e., for general ACTL queries).

ACKNOWLEDGMENTS
The authors thank Martin Fränzle for carefully removing some of our initial ignorance about the intricacies of hybrid systems.

REFERENCES

Safety Verification of Hybrid Systems


Received May 2005; revised October 2005; accepted March 2006
Chapter 5

Safety Verification of Non-linear Hybrid Systems is Quasi-Decidable

This chapter appeared as:

Safety verification of non-linear hybrid systems is quasi-decidable

Stefan Ratschan

Published online: 21 September 2013
© Springer Science+Business Media New York 2013

Abstract Safety verification of hybrid systems is undecidable, except for very special cases. In this paper, we circumvent undecidability by providing a verification algorithm that provably terminates for all robust problem instances, but need not necessarily terminate for non-robust problem instances. A problem instance \( x \) is robust iff the given property holds not only for \( x \) itself, but also when \( x \) is perturbed a little bit. Since, in practice, well-designed hybrid systems are usually robust, this implies that the algorithm terminates for the cases occurring in practice. In contrast to earlier work, our result holds for a very general class of hybrid systems, and it uses a continuous time model.

Keywords Hybrid systems · Safety verification · Decidability · Robustness

1 Introduction

Terminating algorithms for the verification of hybrid systems are known only for very special cases. In fact, most classes of hybrid systems verification problems are known to be undecidable [15]. Recently, there have been attempts at circumventing this [9–11] by observing that, in practice, hybrid systems can never model a given real system precisely, but only up to perturbations. Hence it suffices to verify robust systems, that is, systems that do not change the desired property under perturbations.\(^1\)

We say that a problem is quasi-decidable iff a (possibly non-terminating) algorithm exists whose result is always correct, but which is required to terminate only for robust inputs. We show quasi-decidability of safety verification of a class of hybrid systems that allows arbitrary Boolean combinations of non-linear differential equalities and inequalities for defining

---

\(^1\)In fact, in the special case of timed automata, there is a whole stream of work on avoiding the verification of non-robust properties, see for example [20, 25].

This is an extended version of a conference paper [23]. This work was supported by the Czech Science Foundation (GAČR) grant number P202/12/J060 with institutional support RVO:67985807.

S. Ratschan (✉)
Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic
e-mail: stefan.ratschan@cs.cas.cz
the continuous flow, and arbitrary Boolean combinations of non-linear equalities and inequalities for defining the set of initial and unsafe states, and for defining the set of possible discontinuous jumps of the system.

Theoretical results such as this one (see also [7–11, 21]) have heavy practical consequences: Up to now, hybrid systems verification algorithms have been evaluated purely experimentally, on finitely many benchmark examples. However, one would like a practical verification algorithm to terminate for all robust inputs. Hence we now have a formal tool to evaluate the power of practical verification algorithms.

Our result holds for a very general class of hybrid systems that includes non-linear differential equalities and inequalities. In contrast to that, in Fränzle’s result [10] all defining constraints have to be polynomial and especially, the continuous flow has to be given not in the form of differential equations but in the form of polynomial flows which, in general, does not even allow the modeling of linear differential equations. In the case where the safety property does not hold (i.e., falsification of the safety property), we in fact also restrict ourselves to polynomials, however, this includes polynomial differential equations (in contrast to polynomial flows). In the case where the safety property holds (i.e., verification of the safety property), we even allow the constraints defining the differential equations to contain transcendental function symbols such as sin and exp.

In contrast to our earlier work [9], for the result in this paper we use a continuous time model which results in fundamental additional difficulties which we will discuss in detail in Sect. 5, along with further related work.

The content of the paper is as follows: In Sect. 2 we define our basic notions and the main theorem of the paper. In the following two sections we prove this theorem by providing both an algorithm for verification (Sect. 3) and falsification (Sect. 4). In Sect. 5 we discuss related work, and in Sect. 6 we conclude the paper.

2 Hybrid systems and their quasi-decidability

In this section we describe the solved problem in detail. In the literature, state spaces of dynamical systems are usually defined using tuples (for example, tuples in \( \mathbb{R}^n \)). Here, we take a little bit more flexible approach, that allows us to directly access the individual tuple elements using names. For these names we use a finite set \( V \) whose elements we call \( \textit{variables} \). Moreover, we use the set \( \dot{V} = \{ \dot{v} | v \in V \} \) to access the values of derivatives, and the set \( V' = \{ v' | v \in V \} \) to access the result of a discrete state change (i.e., of jumps).

Moreover, we fix a finite set \( M \) whose elements we call \( \textit{modes} \), and use the additional specific variable name \( \text{mode} \) to access them, and the variable name \( \text{mode}' \) to access them in the case of the result of a discrete jump.

Now we call a function that assigns to some symbols from \( \{\text{mode}, \text{mode}'\} \cup V \cup \dot{V} \cup V' \) a real value a \textit{valuation}. These valuations will take the role of tuples to form the state space of hybrid systems. For a subset \( X \) of \( \{\text{mode}, \text{mode}'\} \cup V \cup \dot{V} \cup V' \) we denote the set of valuations that assigns values exactly to the elements of \( X \) by \( \Gamma(X) \).

For every valuation \( \sigma \) in \( \Gamma(V) \), we denote by \( \text{Prime}(\sigma) \) the corresponding valuation with primed variables, that is, \( \text{Prime}(\sigma) \) is a valuation in \( \Gamma(V') \), and for all \( v \in \{\text{mode}\} \cup V, \text{Prime}(\sigma)(v') = \sigma(v) \).

For two valuations \( \sigma_1, \sigma_2 \) that coincide on joint variables, we define their concatenation \( \sigma_1 \bullet \sigma_2 \) as the valuation that is defined on the union of the two domains of definition and always assigns the corresponding value. That is, for \( \sigma_1 \in \Gamma(X_1) \) and \( \sigma_2 \in \Gamma(X_2) \) such that
for all \( v \in X_1 \cap X_2, \sigma_1(v) = \sigma_2(v) \), we have that for all \( v \in X_1, (\sigma_1 \bullet \sigma_2)(v) = \sigma_1(v) \), and for all \( v \in X_2, (\sigma_1 \bullet \sigma_2)(v) = \sigma_2(v) \).

**Definition 1** A hybrid system is a tuple of the form \((S, Init, Flow, Jump, Unsafe)\) where \(S\) (the state space of the hybrid system) is a subset of \(\Gamma(\{mode\} \cup V)\) such that for every \( v \in V \) we have a non-empty closed real interval \(I_v\) such that \(S = \{\sigma \mid \sigma \in \Gamma(\{mode\} \cup V), \forall v \in V, \sigma(v) \in I_v\}\). In other words, the continuous part of the state space has the form of a hyper-rectangle. In addition,

- \( Init \subseteq S \),
- \( Flow \subseteq \Gamma(\{mode\} \cup V \cup \dot{V}) \), such that for all \( \sigma \in Flow \), for all \( v \in V, \sigma(v) \in I_v \),
- \( Jump \subseteq \Gamma(\{mode\} \cup V \cup \{mode'\} \cup V') \), such that for all \( \sigma \in Jump \), for all \( v \in V, \sigma(v) \in I_v \) and \( \sigma(v') \in I_v \), and
- \( Unsafe \subseteq S \).

That is, a hybrid system has a set of initial and unsafe elements that are sub-sets of the state space. Moreover, it relates derivatives to state space elements, and relates state space elements to primed versions of state space elements. Note that the set \( Flow \) does not necessarily relate a derivative to all state space elements. The case where no derivative is related to a certain state space element simply expresses the fact that no flow is possible, and hence a jump has to be taken (viz. the notion of a forced or urgent transition).

We will use the following objects to describe continuous evolution of hybrid systems:

**Definition 2** A flow of length \( t \) over \( S \subseteq \Gamma(\{mode\} \cup V) \) is a function \( \phi : [0, t] \rightarrow S \) such that

- \( \phi(s)(\text{mode}) \) is constant over all \( s \in [0, t] \), and
- for every \( v \in V \), the function \( \phi^v \) that assigns to every \( s \in [0, t] \) the value \( \phi(s)(v) \), is differentiable.

Based on this, we define \( \dot{\phi} : [0, t] \rightarrow \Gamma(\dot{V}) \) in such a way that for every \( s \in [0, t] \), \( v \in V \), \( \phi(s)(\dot{v}) = \dot{\phi}^v(s) \).

The property we study in this paper is reachability of the set of unsafe states:

**Definition 3** For a given hybrid system \((S, Init, Flow, Jump, Unsafe)\), an error trajectory is a sequence of flows \((\phi_0, \ldots, \phi_n)\) over \( S \) of lengths \((t_1, \ldots, t_n)\) such that, for all \( i \in \{0, \ldots, n\} \)

- if \( i < n \), then \( \phi_i(t_i) \bullet \text{Prime}(\phi_{i+1}(0)) \in Jump \) or \( \phi_i(t_i) = \phi_{i+1}(0) \)
- for all \( s \in [0, t_i] \), \( \phi_i(s) \bullet \dot{\phi}_i(s) \in Flow \),

and \( \phi_0(0) \in Init, \phi_n(t_n) \in Unsafe \). A hybrid system is safe if it does not have an error trajectory.

A notion of solution of a hybrid system immediately follows from this definition after dropping the condition that \( \phi_n(t_n) \in Unsafe \). However, we will not need an explicit definition of solution in this paper, since the notion of error trajectory is precisely what is needed for defining safety of a hybrid system.

For describing hybrid systems we use constraints. We define an arithmetical term to be an expression that may contain variables in \( V \), rational constants, and function symbols in
Now we define a constraint to be a Boolean combination of two types of atomic constraints:

- equalities and inequalities of the form \( t \leq c \), where \( t \) is an arithmetical term, \( r \in \{=, \leq, \geq\} \), and \( c \) is a rational number.
- equalities and inequalities of the form \( \text{mode} = m \) or \( \text{mode} \neq m \), where \( m \in M \) (we call this a mode constraint).

A flow constraint is a constraint that, in addition to the above, allows atomic constraints of the form \( \dot{v} \leq t \), where \( r \in \{=, \leq\} \), \( v \) is a variable from \( V \), and \( t \) is an arithmetical term over \( V \). A jump constraint is a constraint that, in addition to the variables in \( \{\text{mode}\} \cup V \), allows their primed versions, that is, variables in \( \{\text{mode}'\} \cup V' \).

The definition of the semantics of such constraints is straight-forward. We denote the function from valuations to real numbers described by a term \( t \) by \( \llbracket t \rrbracket \). We write \( \sigma \models C \) for the fact that a valuation \( \sigma \) satisfies a constraint \( C \), and we write \( \llbracket C \rrbracket \) for the set of valuations satisfying \( C \). We use corresponding definitions for flow and jump constraints in analogy.

Now we have a way of syntactically describing hybrid systems using constraints:

**Definition 4** For a given state space \( S \), and constraints Init, Flow, Jump, and Unsafe we call the tuple \((S, \text{Init}, \text{Flow}, \text{Jump}, \text{Unsafe})\) a hybrid systems description. Furthermore we denote by

\[ \llbracket (S, \text{Init}, \text{Flow}, \text{Jump}, \text{Unsafe}) \rrbracket \]

the hybrid system \( (S, \llbracket \text{Init} \rrbracket, \llbracket \text{Flow} \rrbracket, \llbracket \text{Jump} \rrbracket, \llbracket \text{Unsafe} \rrbracket) \).

In this case we also say that the hybrid system fulfills the corresponding hybrid systems description. We straightforwardly lift Definition 3 from hybrid systems to hybrid system descriptions.

**Example 1** For illustrating the above definitions, consider the following simple hybrid system. We assume a set of variables \( V = \{x_1, x_2\} \), and a set of modes \( M = \{m_1, m_2\} \). The hybrid system has a state space \( S = \{\sigma \mid \sigma \in \Gamma(\{\text{mode}\} \cup V), \sigma(\text{mode}) \in \{m_1, m_2\}, \sigma(x_1) \in [0, 1], \sigma(x_2) \in [0, 1]\} \). The set of initial states are given by the constraint

\[ \text{mode} = m_1 \land x_1 = 0 \land x_2 = 0. \]

The constraint \( x_2 \geq 1 \) describes the unsafe states, and hence, safety of a state does not depend on the mode of this state. The hybrid system may switch modes from \( m_1 \) to \( m_2 \) if \( x_1 \geq 0.4 \), that is, the constraint Jump is of the form

\[ \text{mode} = m_1 \land x_1 \geq 0.4 \land \text{mode}' = m_2 \land x'_1 = x_1 \land x'_2 = x_2. \]

The continuous behavior is quite simple: In mode \( m_1 \) a flow is only possible as long as \( x_1 \leq 0.5 \). In both modes, \( x_1 \) evolves with a derivative in the interval \([0.9, 1.1]\), while \( x_2 \)

---

\(^{2}\) For a function \( f : \mathbb{R}^n \to \mathbb{R} \), compact intervals \( I_1, \ldots, I_n \), we need to be able to compute an interval \( J \supseteq f(I_1, \ldots, I_n) \) such that the over-approximation of \( J \) over \( f(I_1, \ldots, I_n) \) can be made arbitrarily small. Note that this requires continuity of \( f \) but not Lipschitz continuity. For example, we could include \( \sqrt{|x|} \), which is not Lipschitz continuous.
evolves deterministically with slope 1 in mode $m_1$ and $-1$ in mode $m_2$. So we have the flow constraint
\[
\text{(mode } = m_1 \land \dot{x}_1 \geq 0.9 \land \dot{x}_1 \leq 1.1 \land \dot{x}_2 = 1 \land x_1 \leq 0.5) \\
\lor (\text{mode } = m_2 \land \dot{x}_1 \geq 0.9 \land \dot{x}_1 \leq 1.1 \land \dot{x}_2 = -1).
\]

Observe that in mode $m_2$, the value of variable $x_2$ decreases. Moreover, the system can stay in mode $m_1$ only as long as $x_1 \leq 0.5$, and so $x_2$ can increase only for a limited time. So this hybrid system does not have an error trajectory, and hence it is safe.

It is well-known that—except for very special cases—checking whether a hybrid system is safe is an undecidable problem [15]. However, in the real world, we will not be able to implement a hybrid system description exactly and we do not want to prove a hybrid systems description safe, if the system fulfilling this description is safe but there is a system that fulfills the description up to small perturbations and is unsafe. Hence it suffices to have an algorithm that can prove safety of hybrid systems descriptions for which all hybrid systems that fulfill the description up to small perturbations are safe (such as the example above).

This is a general situation for undecidable problems in domains prone to perturbations. Hence it is worthy to discuss the problem description in a general form, independent of a specific definition of the notion of "to fulfill up to small perturbations". We will only later (Definition 9) formalize what we exactly mean by that notion in our domain of hybrid systems. Moreover, by delaying that definition (which is quite involved), the reader can grasp the essence of the problem before delving into details.

In general, we have the situation illustrated in Fig. 1. Here the syntactic level is depicted on the top, and the semantic level on the bottom. We have a language $\mathcal{L}$ (in our case, the set of all hybrid systems descriptions) describing corresponding objects in $\left[\mathcal{L}\right]$ (in our case hybrid systems).
systems). We have some property \( P \) (in our case, safety of hybrid system descriptions) on \( \mathcal{L} \) such that for a language element \( D \in \mathcal{L} \), \( P(D) \) holds iff \( \llbracket P \rrbracket (\llbracket D \rrbracket) \) holds for a certain property \( \llbracket P \rrbracket \) (in our case, safety of a hybrid system). However, when we try to implement \( D \) in the real world, the result will not precisely fulfill the hybrid systems description \( D \), but will fulfill \( D \) only up to some perturbation of size \( \varepsilon \). Hence, we will have a set \( \llbracket D \rrbracket_\varepsilon \) of objects fulfilling \( D \) up to perturbations of size \( \varepsilon \). It can be the case that \( \llbracket P \rrbracket \) holds on all elements of \( \llbracket D \rrbracket_\varepsilon \) (this is the case for \( D_1 \) in the figure), or only on some elements (\( D_2 \) in the figure). This difference is described by the following definition.

**Definition 5** Let \( \mathcal{L} \) be a language describing elements of a set \( \llbracket \mathcal{L} \rrbracket \), and let \( P \) be a property on \( L \), and \( \llbracket P \rrbracket \) a property on \( \llbracket L \rrbracket \), such that for all \( D \in \mathcal{L} \), \( P(D) \) iff \( \llbracket P \rrbracket (\llbracket D \rrbracket) \). Let \( D \in \llbracket L \rrbracket \), and let \( \llbracket D \rrbracket_\varepsilon \) be the set of all elements of \( \llbracket \mathcal{L} \rrbracket \) fulfilling \( D \) up to \( \varepsilon \). Then \( P \) holds robustly on \( D \) iff there is a real number \( \varepsilon > 0 \) (the robustness margin) such that \( \llbracket P \rrbracket \) holds on all elements of \( \llbracket D \rrbracket_\varepsilon \).

In the figure, \( P \) holds robustly on \( D_1 \). For \( D_2 \), the value \( \varepsilon \) is not a robustness margin. There might be a smaller value \( \varepsilon' \), and a smaller corresponding set \( \llbracket D_2 \rrbracket_\varepsilon' \) such that \( \llbracket P \rrbracket \) holds on all its elements. If, however small we choose \( \varepsilon' \), some elements of \( \llbracket D_2 \rrbracket_\varepsilon' \) still do not fulfill \( \llbracket P \rrbracket \), \( P \) does not hold robustly on \( D_2 \). Only in that case do we not require our algorithms to terminate, that is, in that case an algorithm trying to verify safety of a given hybrid system is allowed to run forever. This is the essential point, why the following notion of quasi-decidability is weaker than decidability.

**Definition 6** We call a given property \( P \) on a language \( \mathcal{L} \) quasi-semidecidable iff there is an algorithm \( A \) such that for a given \( D \in \mathcal{L} \),

- if \( A(D) \) terminates then \( P(D) \) (i.e., \( A \) is correct),
- \( A(D) \) terminates if \( P \) holds robustly on \( D \).

If both \( P \) and \( \neg P \) are quasi-semidecidable then \( P \) is quasi-decidable.

The definitions above depend on the notion of ”fulfilling a language element up to \( \varepsilon \)”. We will spend the rest of this section on defining this in our case, that is, defining the notion of a hybrid system fulfilling a hybrid systems description up to \( \varepsilon \).

Due to reasons discussed in footnote 3, we have to take into account syntactic perturbations here. We define this using a distance measure on constraints. Note however, that in the following only the limit case of this definition is relevant, since Definition 5 does not consider a fixed robustness margin \( \varepsilon \), but only requires existence of an \( \varepsilon > 0 \).

The basic idea for defining this distance measure is, that two constraints are the same up to ”addition of constants up to a certain size”:

**Definition 7**

- We call a term basic, if it is either a variable, or a constant, or a term of the form \( x + c \), where \( x \) is a variable, and \( c \) a constant. If the set of variables contained in a basic term (this is either a singleton set or the empty set) is the same in two basic terms, we define the distance between these terms as the distance between the corresponding constants, using the constant 0 if one of the terms does not contain a constant. If the set of contained variables is not the same in both basic terms, their distance is \( \infty \).
The distance \( d(C, C') \) between two constraints \( C \) and \( C' \) is \( \varepsilon \) iff \( C' \) can be obtained from \( C \) by replacing some basic terms by basic terms of finite distance and \( \varepsilon \) is the maximum of these distances. Otherwise, the distance is \( \infty \).

**Example 2** For measuring the distance between the constraint \((x + 2)^2 + 1x \leq 0\) and \(x^2 + 2x \leq 0\) we observe that for getting from the first to the second constraint we have to replace the basic term \(x + 2\) by \(x\), and the basic term 1 by the basic term 2. The distance is the maximum of the distances of corresponding basic terms, that is, the distance is 2.

**Example 3** The constraints \((x - 2)^2 - 1 \leq 0\) and \(x^2 - 4x + 4 - 1 \leq 0\), although semantically equivalent, have infinite distance. This does not pose any problem here. On the contrary, this makes our result stronger, since it leads to many hybrid systems descriptions being robust, and hence to a strong termination condition for our algorithms (in Fig. 1 the blobs in the lower part become smaller, resulting in more blobs to completely lie in \( P \)).

We continue with defining an analogon of the notion of "fulfilling a hybrid systems description up to \( \varepsilon \)" for our constraint language.

**Definition 8** A set \( P \) of valuations is an \( \varepsilon \)-perturbed solution set of a constraint \( C \) iff

- for every valuation \( \sigma \in P \), there is a constraint \( C^* \) with \( d(C, C^*) \leq \varepsilon \) such that \( \sigma \models C^* \), and
- for every valuation \( \sigma \not\in P \), there is a constraint \( C^* \) with \( d(C, C^*) \leq \varepsilon \) such that \( \sigma \not\models C^* \).

In other words, the set \( P \) may contain valuations that do not satisfy the constraint, and may not contain valuations that do satisfy constraint, but we have to make sure that in both cases the error that we make is not too large. Note that this does not necessarily mean that \( P \) is the solution set of a perturbed constraint \( C^* \):

**Example 4** The interval \([-1, 1]\) is a 1-perturbed solution set of the constraint \( x = 0 \). However, there is no \( \varepsilon \) such that \( x = \varepsilon \) has the solution set \([-1, 1]\).

Lifting this definition to hybrid systems is straightforward:

**Definition 9** Given a hybrid system description \((Init, Flow, Jump, Unsafe)\), a hybrid system \((Init, Flow, Jump, Unsafe)\) fulfills \((Init, Flow, Jump, Unsafe)\) up to \( \varepsilon \) iff

- \( Init \) is an \( \varepsilon \)-perturbed solution set of \( Init \),
- \( Flow \) is an \( \varepsilon \)-perturbed solution set of \( Flow \),
- \( Jump \) is an \( \varepsilon \)-perturbed solution set of \( Jump \), and
- \( Unsafe \) is an \( \varepsilon \)-perturbed solution set of \( Unsafe \).

Note that—in analogy to individual constraints—a hybrid system \( H \) fulfilling a hybrid system description \( H \) up to \( \varepsilon \) does not necessarily mean that \( H \) fulfills a hybrid system description that is a perturbation of \( H \) (see Example 4 above).

Since Definition 9 was the last missing element of the definition of quasi-decidability, after setting \( \mathcal{L} \) to the set of hybrid system descriptions, and \( P \) to their safety in Definition 6, we now have a complete formalization of the notion of quasi-decidability of hybrid systems. So we are ready to formulate the main theorem of this paper:
Theorem 1 Safety of hybrid system descriptions is quasi-semidecidable. Moreover, it is quasi-decidable in the case where we allow only addition and multiplication as function symbols in hybrid system descriptions.

A proof of this theorem consists of two quasi-semidecidability proofs, one for the positive case of verification of the safety property, and one for the negative case of falsification of the safety property. We will use the following two sections for the two corresponding parts of the proof. Within these sections we will provide respective algorithms for verifying and falsifying hybrid systems.

3 Quasi-semidecidability of verification

For proving quasi-semidecidability of verification we use the fact that for every hybrid system there is a rectangular \( \varepsilon \)-approximation [14]. Here we have to overcome two major obstacles:

- The original proof of this existence property was not constructive.
- Although rectangular automata have a much simpler structure than general hybrid systems, their safety is still undecidable.

Before solving these problems, we introduce a representation of rectangular sets: A box is a function that assigns to some variables in \( V \cup \dot{V} \cup V' \) a non-empty closed real interval, and to some variables in \{mode, mode'\} a subset of modes from \( M \). Throughout the paper we use the situation that a box \( B \) does not assign a value to a given variable as a shortcut for the value \( B(v) \) being \( M \), if \( v \in \{\text{mode}, \text{mode}'\} \), and being \([−\infty, \infty]\), otherwise. We will say that a box has dimension \( d \) iff it assigns \( d \) real intervals (i.e., \( d \) intervals not equal to \([−\infty, \infty]\)). We lift set membership to boxes by defining a valuation \( \sigma \) to be element of a box \( B \) iff for every variable \( v \) on which \( \sigma \) is defined, \( \sigma(v) \in B(v) \). Analogously we lift other set operations such as \( \subseteq \) and \( \cap \) using the corresponding variable-wise operations on intervals and sets of modes, respectively. Box union \( \cup \) is defined by lifting union for variables in \{mode, mode'\}, and interval union (the smallest interval containing both arguments) for the other variables. For boxes we define concatenation analogously as for valuations. We call a box proper, if it only assigns intervals (and no modes).

A sat-box (for satisfiability box) is either a box, or the value \( \bot \) which we call the empty box. Such sat-boxes will be used for flow constraints where we either deduce unsatisfiability or a box bounding the set of possible derivatives. A sat-box has dimension \( d \) iff it is equal to \( \bot \) or if it is a box of dimension \( d \) (hence \( \bot \) can have any dimension). Sometimes we will write \( \mathbf{F} \) for \( \bot \) and \( \mathbf{T} \) for the unique zero-dimensional box, and use them in the role of the corresponding Boolean constants. The box operations \( \cap \) and \( \cup \) can be easily lifted from boxes to sat-boxes by considering \( \bot \) to be the smallest element in the \( \subseteq \) order. Also, the element relation \( \in \) can be naturally lifted by defining \( \bot \) to have no element (which, of course, corresponds to its name ”empty box”).

Now we start with removing the first obstacle mentioned at the beginning of this section: computing a rectangular over-approximation of a hybrid system such that the over-approximation error is smaller than a given bound.

The algorithm uses interval arithmetic as its basis. For a term \( \tau \), and proper box \( B \), let \( I(\tau)(B) \) denote the evaluation of \( \tau \) on \( B \) using interval arithmetic [19]. For polynomials, computation with interval endpoints can be implemented exactly, in rational number arithmetic. For terms containing transcendental function symbols such as sin, however, one has
to use (conservative) rounding [24]. Here we assume the usage of fixed-precision floating-point arithmetic. Moreover, to ensure convergence (see Lemma 1 below for details), we assume that the used precision goes to infinity as the size of the box $B$ goes to zero.

The result of interval arithmetic over-approximates the set of all values the term $t$ takes in the box $B$, due to the so-called Fundamental Theorem of Interval Arithmetic [18].

Property 1

$$I(t)(B) \supseteq \{ \llbracket t \rrbracket(\sigma) \mid \sigma \in B \}$$

Now we can over-approximate the satisfiability information of constraints by defining the symbol $\models_I$ (interval satisfiability check) for a box $B$ as follows:

- $B \models_I \text{mode} = m$ if $m \in B(\text{mode})$, and $\text{F}$ otherwise,
- $B \models_I t \geq 0$, where $t$ does not contain dotted variables, is $\text{T}$ iff there exists a real value $x \in I(t)(B)$ such that $x \geq 0$, and $\text{F}$, otherwise,
- $B \models_I C_1 \land C_2$ is $B \models_I C_1 \cap B \models_I C_2$, and
- $B \models_I C_1 \lor C_2$ is $B \models_I C_1 \cup B \models_I C_2$.

Example 5 Let $C$ be the constraint $x^2 - 1 = 0 \land x - 2 \geq 0$, and let $B$ be the box $x \mapsto [-10, 0]$. Interval arithmetic evaluates the terms in $C$ recursively. So $I(x^2)(B) = [0, 100]$, and $I(x^2 - 1)(B) = [-1, 99]$. Since this interval contains zero, $(B \models_I x^2 - 1 = 0) = \text{T}$. Moreover, $I(x - 2)(B) = [-12, -2]$, and $(B \models_I x - 2 \geq 0) = \text{F}$. In the zero-dimensional case, intersection and union of boxes implements conjunction and disjunction of the corresponding Boolean values. So $(B \models_I C) = \text{T} \cap \text{F} = \text{F}$.

Remember that, by default, variables are assigned the interval $[-\infty, \infty]$. Hence the semantics is also well-defined in cases where the branches of a conjunction (or disjunction) contain different variables.

We generalize the interval satisfiability check to constraints containing dotted variables (denoting derivatives). In this case, the result is a sat-box, whose dimension (if containing a box) is equal to the number of dotted variables. The purpose of this definition is to over-approximate the projection of the solution set of the constraint to these variables:

- $B \models_I \dot{a} = t$ is defined as $\{ \dot{a} \mapsto I(t)(B) \}$
- $B \models_I \dot{a} \leq t$ is defined as $\{ \dot{a} \mapsto [-\infty, I(t)(B)] \}$
- $B \models_I \dot{a} \geq t$ is defined as $\{ \dot{a} \mapsto [I(t)(B), \infty] \}$

The rest of the definition is kept unchanged.

Example 6 Let $C$ be the flow constraint $\dot{x} = x^2 \land x - 2 \geq 0$, and let $B$ be the box $x \mapsto [1, 3]$. Then $B \models_I \dot{x} = x^2$ is the box $\{ \dot{x} \mapsto [1, 9] \}$ and $(B \models_I x - 2 \geq 0) = \text{T}$. Hence $\{ \dot{x} \mapsto [1, 9] \} \cap \text{T} = \{ \dot{x} \mapsto [1, 9] \}$ (remember that $\text{T}$ is the unique zero dimensional box that assigns to every variable the default interval $[-\infty, \infty]$).

For the slightly modified constraint $x^2 - 1 = 0 \land x - 10 \geq 0$, however, $B \models_I x - 10 \geq 0$ evaluates to $\text{F}$, and hence also the whole constraint.

This definition fulfills its purpose due to the following generalization of the fundamental theorem of interval arithmetic to our constraints:

Theorem 2 For every constraint $C$, box $B$ on the undotted variables of $C$, valuation $\sigma \in B$, and valuation $\dot{\sigma}$ on the dotted variables of $C$ such that $\sigma \bullet \dot{\sigma} \models C$, we have $\dot{\sigma} \in B \models_I C$. 
Proof Let \( B, \sigma, \dot{\sigma} \) arbitrary, but fixed, fulfilling the assumptions above. We prove that \( \dot{\sigma} \in B \models I C \). We proceed by induction over the structure of \( C \). We have the following base cases:

- \( C \) is of the form \( t = 0 \), where \( t \) is a term. We have \( \sigma \bullet \dot{\sigma} \models t = 0 \), and hence \( \llbracket t \rrbracket (\sigma \bullet \dot{\sigma}) = 0 \) and since \( t \) does not contain dotted variables, also \( \llbracket \sigma \rrbracket (\sigma) = 0 \). To prove that \( \dot{\sigma} \in B \models I C \), we have to prove that \( 0 \in I(\tau)(B) \). This holds, since due to the fundamental theorem of interval arithmetic, \( \llbracket \sigma \rrbracket (\sigma) \in I(\tau)(B) \) for \( \sigma \in B \).

- \( C \) is of the form \( \dot{x} = \tau \). In this case, since \( \sigma \bullet \dot{\sigma} \models \dot{x} = \tau \), it holds that \( \dot{\sigma} = \llbracket \tau \rrbracket (\sigma) \). To prove that \( \dot{\sigma} \in B \models I C \), we have to prove that \( \dot{\sigma} \in I(\tau)(B) \). This holds since due to the fundamental theorem of interval arithmetic, \( \llbracket \tau \rrbracket (\sigma) \in I(\tau)(B) \) for \( \sigma \in B \).

- \( C \) is of the form \( \dot{x} \leq \tau \). In this case, since \( \sigma \bullet \dot{\sigma} \models \dot{x} \leq \tau \), it holds that \( \dot{\sigma} \leq \llbracket \tau \rrbracket (\sigma) \). To prove that \( \dot{\sigma} \in B \models I C \), we have to prove that \( \dot{\sigma} \in [-\infty, I(\tau)(B)] \), that is, \( \dot{\sigma} \leq I(\tau)(B) \).

This holds since due to the fundamental theorem of interval arithmetic, for all \( x \), \( \llbracket \tau \rrbracket (x) \in I(\tau)(B) \), and hence \( \llbracket \tau \rrbracket (x) \leq I(\llbracket \tau \rrbracket)(B) \).

- the other cases of atomic constraint are analogous to the previous cases.

The induction step is easy. \( \square \)

Example 7 Continuing Example 6, let in addition \( \sigma \) be the valuation \( \{x \mapsto 2\} \) (which is an element of \( B \)), and let \( \dot{\sigma} \) be the valuation \( \{\dot{x} \mapsto 4\} \) (for which \( \sigma \bullet \dot{\sigma} \models C \)). Then the box \( B \models I C \) which is \( \{\dot{x} \mapsto [1, 9]\} \) contains the valuation \( \{\dot{x} \mapsto 4\} \).

In the special case of constraints without dotted variables, interval satisfiability just over-approximates satisfiability:

Corollary 1 For every constraint \( C \) without dotted variables, box \( B \) on the variables of \( C \), if \( B \) contains a valuation \( \sigma \) such that \( \sigma \models C \), then \( (B \models I C) = T \).

Equivalently, \( (B \models I C) = F \) implies that there is no valuation \( \sigma \in B \) such that \( \sigma \models C \). Since the implication only points in one direction, in the case of \( (B \models I C) = T \) one cannot conclude anything about the satisfiability of \( C \), and in the case of an unsatisfiable constraint one cannot conclude anything about \( (B \models I C) \). In particular, it can, but need not necessarily happen that for a non-robust constraint unsatisfiable \( C \), \( (B \models I C) \) gives the precise result:

Example 8 For the term \( x^2 \) and the box \([-10, 10]\) interval arithmetic \( I(x^2)([-10, 10]) \) may compute the precise result \([0, 100]\). Then, for the unsatisfiable but not robust constraint \( x^2 < 0 \), we get the precise result \( (B \models I x^2 < 0) = F \). If however, \( I(x^2)([-10, 10]) \) is computed as \([-0.000001, 100]\) then the result is \((B \models I x^2 < 0) = T \).

Now we present an algorithm for which we will prove that it over-approximates a given hybrid system arbitrarily closely. For bounding the over-approximation error we use a bound on the size of the boxes. For a non-empty interval \([a, b]\), its width is defined to be \( b - a \), and for a non-empty set of modes \( M^* \subseteq M \), we define its width to be zero if \( M^* \) is a singleton set, and \( \infty \), otherwise. We define the diameter \( \text{diam}(B) \) of a box \( B \) to be the maximum width of \( B(v) \) over all variables \( v \) on which \( B \) is defined (i.e., not equal \([-\infty, \infty]\)).

The algorithm in Fig. 2 approximates a given hybrid systems description using a hybrid systems description completely defined by boxes. Here we use the notation \( x \in [\overline{a}, \overline{a}] \) as a short-cut for the constraint \( a \leq x \land x \leq a \). The idea is to put a grid of boxes onto the state space, and then
**Input:**
- a hybrid systems description 
  \((S, \text{Init}, \text{Flow}, \text{Jump}, \text{Unsafe})\),
- a strictly positive real value \(\delta\)

\(G \leftarrow \) set of boxes of diameter \(\delta\) covering the state space \(S\)

\(\text{Init}_R \leftarrow \bigvee_{B \in G, B \models \text{Init}} [\text{mode} = B(\text{mode}) \land \bigwedge_{v \in V} v \in B(v)]\)

\(\text{Flow}_R \leftarrow \bigvee_{B \in G} [\text{mode} = B(\text{mode}) \land \bigwedge_{v \in V} v \in B(v) \land \bigwedge_{v \in V} \dot{v} \in (B \models \text{Flow})]\)

\(\text{Jump}_R \leftarrow \bigvee_{B, B' \in G, (B, B') \models \text{Jump}} [\text{mode} = B(\text{mode}) \land \bigwedge_{v \in V} v \in B(v) \land \text{mode}' = B'(\text{mode}) \land \bigwedge_{v \in V} v' \in B'(v)]\)

\(\text{Unsafe}_R \leftarrow \bigvee_{B \in G, B \models \text{Unsafe}} [\text{mode} = B(\text{mode}) \land \bigwedge_{v \in V} v \in B(v)]\)

\((S, \text{Init}_R, \text{Flow}_R, \text{Jump}_R, \text{Unsafe}_R)\)

**Fig. 2** Over-approximating abstraction

- to test on each box using the interval satisfiability check, whether it might contain an initial or unsafe state,
- to test for every pair of boxes whether it might contain a jump between them, and
- to compute an interval containing the possible derivatives for each box.

In contrast to the discrete time case [9], here it does not suffice to abstract to a purely discrete system. The reason is that in discrete time, if the hyper-rectangles are sufficiently small, they can separate two subsequent steps of the system. However, for continuous evolution, this is not possible.

We denote the result computed by the algorithm in Fig. 2 by \(A(H, \delta)\). This is again a hybrid system description, and from Theorem 2 it easily follows that \(A(H, \delta)\) over-approximates \(H\):

**Theorem 3** For the result \((S, \text{Init}_R, \text{Flow}_R, \text{Jump}_R, \text{Unsafe}_R)\) of the algorithm application \(A((S, \text{Init}, \text{Flow}, \text{Jump}, \text{Unsafe}), \delta)\), \(\text{Init}\) implies \(\text{Init}_R\), \(\text{Flow}\) implies \(\text{Flow}_R\), \(\text{Jump}\) implies \(\text{Jump}_R\), and \(\text{Unsafe}\) implies \(\text{Unsafe}_R\).

And hence the result of the algorithm in Fig. 2 can be used to prove safety of the original system.

**Corollary 2** If \([A((S, \text{Init}, \text{Flow}, \text{Jump}, \text{Unsafe}), \delta)]\) is safe, then \([A((S, \text{Init}, \text{Flow}, \text{Jump}, \text{Unsafe})])\) is also safe.

However, this does not guarantee anything about the amount of over-approximation of the algorithm. In order to arrive at bounds for this over-approximation, we first study such bounds for constraints. In earlier work [9] we proved results bounding the over-approximation of \(\models\) for constraints without dotted variables. We generalize those results here to the case with dotted variables:

**Lemma 1** For every constraint \(\mathcal{C}\), box \(B\) defined on all undotted variables of \(\mathcal{C}\), for all \(\varepsilon > 0\) there is a \(\delta > 0\) such that for every box \(B'\) with \(B' \subseteq B\), \(\text{diam}(B') < \delta\), for every \(\sigma \in B', and
for every \( \hat{\sigma} \in (B' \models_I C) \), there is a \( C^* \) with \( d(C, C^*) \leq \varepsilon \), such that
\[
\sigma \bullet \hat{\sigma} \models C^*.
\]

**Proof** For proving this lemma we use the fact (which we will call convergence of interval arithmetic in the rest of the proof) that for every arithmetical term \( \varepsilon \) with function symbols in the set \( \{+, \times, ^*, \exp, \sin, \cos\} \), denoting a function \( \llbracket e \rrbracket \) and box \( S \) for every \( \varepsilon > 0 \) there is a \( \delta > 0 \) such that for every box \( B \) with \( B \subseteq S \), \( \text{diam}(B) < \delta \), for all \( y \in I(\varepsilon)(B) \), there is an \( x \in B \) such that \( d(\llbracket e \rrbracket(x), y) \leq \varepsilon \). This fact follows from Lipschitz continuity of interval arithmetic (e.g., Theorem 2.1.1 in Neumaier’s book [19]). Moreover, due to Theorem 2.1.5 in the same book, this holds even in rounded interval arithmetic, as long as we let the used precision go to infinity as the size of the box \( B \) goes to zero which is precisely how we defined evaluation of terms in interval arithmetic.

Now let \( \varepsilon \), \( B \), \( t \) be as required by the assumptions of the lemma. We start with proving the special case that \( C \) is of the form: \( t = 0 \):

Let \( \delta_t \) be the value ensured for \( t \), \( B \), and \( \varepsilon \) by convergence of interval arithmetic. We choose \( \delta \) as \( \min\{\delta_t, \varepsilon\} \), and assume an arbitrary, but fixed box \( B', \sigma \), and \( \hat{\sigma} \) with \( B' \subseteq B \), \( \text{diam}(B') < \delta \), \( \sigma \in B' \), and \( \hat{\sigma} \in (B' \models I(t))(0) \).

From \( \hat{\sigma} \in (B' \models_I t = 0) \) we know that \( 0 \in I(t)(B') \). We construct a \( C^* \) with \( d(C, C^*) \leq \varepsilon \), \( \sigma \bullet \hat{\sigma} \models C^* \) by providing the necessary perturbations of \( C \).

Let \( x \) be an element of \( B' \) such that \( d(\llbracket x \rrbracket(x), 0) \leq \varepsilon \), as ensured by the convergence of interval arithmetic. We perturb (by adding corresponding constants)

- every undotted variable \( v \) in \( C \) by \( \sigma(v) + \varepsilon \) (this perturbation is smaller than \( \varepsilon \) since \( d(\sigma(v), v(x)) \leq \text{diam}(B') \leq \delta = \min\{\delta_t, \varepsilon\} \leq \varepsilon \)),
- and perturb the right-hand side of the constraint by \( \llbracket t \rrbracket(x) \), which is smaller than \( \varepsilon \) by choice of \( x \).

Then \( \sigma \bullet \hat{\sigma} \models C^* \), which is equivalent to \( \sigma \models C^* \), is equivalent to \( x \models t = c \), with \( c = \llbracket t \rrbracket(x) \). This holds according to the definition of \( \models \).

Now we look at the case where \( C \) is of the form \( \hat{a} = t \).

Let \( \delta_t \) be the value ensured for \( t \), \( B \), and \( \varepsilon \) by convergence of interval arithmetic. We choose \( \delta \) as \( \min\{\delta_t, \varepsilon\} \), assume an arbitrary but fixed box \( B', \sigma \), and \( \hat{\sigma} \) with \( B' \subseteq B \), \( \text{diam}(B') < \delta \), \( \sigma \in B' \), and \( \hat{\sigma} \in (B' \models_I \hat{a} = t) \).

From \( \hat{\sigma} \in (B' \models_I \hat{a} = t) \) we know that \( \hat{\sigma} \in I(t)(B') \). We construct a \( C^* \) with \( \sigma \bullet \hat{\sigma} \models C^* \) by providing the necessary perturbations. Let \( x \) be such that \( d(\llbracket t \rrbracket(x), \hat{\sigma}) \leq \varepsilon \), as ensured by the convergence of interval arithmetic.

We perturb

- every undotted variable \( v \) in \( C \) by \( \sigma(v) + \varepsilon \) (this perturbation is smaller than \( \varepsilon \) since \( d(\sigma(v), v(x)) \leq \text{diam}(B') \leq \delta = \min\{\delta_t, \varepsilon\} \leq \varepsilon \)),
- the dotted variables by \( \llbracket t \rrbracket(x) - \sigma \) (this perturbation is smaller than \( \varepsilon \) by choice of \( x \)),
- and do not perturb the right-hand side of the constraint.

Then \( \sigma \bullet \hat{\sigma} \models C^* \) is equivalent to \( x \bullet \{ \hat{a} \mapsto \llbracket t \rrbracket(x) \} \models C \), that is, \( x \bullet \{ \hat{a} \mapsto \llbracket t \rrbracket(x) \} \models \hat{a} = t \) which holds according to the definition of \( \models \).

In the case where \( C \) is an inequality, for example, of the form \( \hat{a} \leq t \), we have to consider two sub-cases:

- \( \hat{\sigma} \in I(t)(B') \): in this case, the proof for the equality case above works.
- \( \hat{\sigma} \notin I(t)(B') \): in this case, we choose \( C^* \) as \( C \), and we have: \( \sigma \bullet \hat{\sigma} \models C^* \) is \( \sigma \bullet \hat{\sigma} \models \hat{a} \leq t \), which according to the definition of \( \models \) is equivalent to \( \hat{\sigma} \leq \llbracket t \rrbracket(\sigma) \). This holds since \( \llbracket t \rrbracket(\sigma) \in I(t)(B') \), \( \hat{\sigma} \notin I(t)(B') \), and \( \hat{\sigma} < I(t)(B') \).
In the case where $C$ is of the form $\text{mode} = m$, the lemma easily holds by choosing $C^*$ to be equal to $C$, in which case $d(C, C^*) = 0$.

For considering general constraints with conjunction and disjunction, we proceed by induction. This easily goes through by choosing the minimum of the $\delta$ for the different atomic constraints and combining the $C^*$ for the different branches. \hfill $\square$

Using Lemma 1 we can bound the over-approximation of the algorithm in Fig. 2 up to arbitrary precision.

**Theorem 4** For every hybrid system description $H$, for all $\epsilon > 0$ there is a $\delta > 0$ such that $[A(H, \delta)]$ is an $\epsilon$-perturbed instance of $H$.

**Proof** Let $\delta_{C,B,\epsilon}$ be the value of $\delta$, as ensured by Lemma 1 for the constraint $C$, the box $B$, and $\epsilon$. Let $\epsilon > 0$ be arbitrary, but fixed. Choose $\delta$ as the minimum of $\delta_{C,B,\epsilon}$ over all constraints $C$ defining $H$, and boxes $B$ forming the state space (one box for each mode).

We assume that $H$ is of the form $(S, \text{Init}, \text{Flow}, \text{Jump}, \text{Unsafe})$, and $[A(H, \delta)]$ is of the form $(S, [\text{Init}_H], [\text{Flow}_H], [\text{Jump}_H], [\text{Unsafe}_H])$. To prove that $[A(H, \delta)]$ is an $\epsilon$-perturbed instance of $H$ we have to prove the corresponding result for each pair of corresponding constraints of $H$ and $A(H, \delta)$. Here, in each case, Theorem 3 implies the second item of Definition 8. Hence it suffice to prove the first item for each pair of corresponding constraints:

- To prove that $[\text{Init}_H]$ is a $\epsilon$-perturbed instance of $\text{Init}$, we have to prove that for every $\sigma \in [\text{Init}_H]$, there is a constraint $\text{Init}^*$ with $d(\text{Init}, \text{Init}^*) \leq \epsilon$ such that $\sigma \models \text{Init}^*$. Let $\sigma$ be an arbitrary, but fixed element of $[\text{Init}_H]$. Then $\sigma$ satisfies at least one disjunct of $\text{Init}_H$. Let $B$ be the mode/box pair generating this disjunct. Then $\sigma \in B$, $B \models \text{Init}$ and $\text{diam}(B) \leq \delta_{\text{Init}, S, \epsilon} \leq \delta$, where $S$ is the box forming the state space of the mode of $\sigma$. Then, by Lemma 1, there is a constraint $\text{Init}^*$ with $d(\text{Init}, \text{Init}^*) \leq \epsilon, \sigma \models \text{Init}^*$.

- Flow: To prove that $[\text{Flow}_H]$ is a $\epsilon$-perturbed instance of $\text{Flow}$, we have to prove that for every $\sigma \bullet \dot{\sigma} \in [\text{Flow}_H]$, there is a constraint $\text{Flow}^*$ with $d(\text{Flow}, \text{Flow}^*) \leq \epsilon$ such that $\sigma \bullet \dot{\sigma} \models \text{Flow}^*$. Let $\sigma \bullet \dot{\sigma}$ be an arbitrary, but fixed element of $[\text{Flow}_H]$. Then $\sigma \bullet \dot{\sigma}$ satisfies at least one disjunct of $\text{Flow}_H$. Let $B$ be the mode/box pair generating this disjunct. Hence $\sigma \in B$, $\dot{\sigma} \in (B \models \text{Flow})$ and $\text{diam}(B) \leq \delta_{\text{Flow}, S, \epsilon} \leq \delta$, where $S$ is the box forming the state space of the mode of $\sigma$. Then, by Lemma 1, there is a constraint $\text{Flow}^*$ such that $d(\text{Flow}, \text{Flow}^*) \leq \epsilon$, and $\sigma \bullet \dot{\sigma} \models \text{Flow}^*$.

- Jump and Unsafe: analogous to $\text{Init}$ \hfill $\square$

The hybrid system $A(H, \delta)$ has a very simple form that is equivalent to a rectangular automaton. Still, this rectangular automaton is not necessarily initialized and hence it belongs to an undecidable class [15]. However, after explicitly solving the flow constraints, it can be completely defined by polynomials. Moreover, it has a bounded state space. Hence one can apply a result by Fränzle [10] which provides an algorithm that, while it does not terminate always, still terminates for all robust inputs. Hence we have:

**Theorem 5** Safety verification of the results of $A(H, \delta)$ is quasi-decidable.

However, it is possible that $A(H, \delta)$ is not robust—even if $H$ is robust. In the case of such non-robustness Fränzle’s algorithm does not terminate. This can be circumvented:
Theorem 6 Safety verification of non-linear hybrid systems is quasi-semidecidable

Proof Let \( F_t \) be a version of Fränzle’s algorithm [10] for safety verification that, if it terminates within \( t \) time units, it return the corresponding (Boolean) result, and otherwise returns false. We use the following algorithm:

\[
i \leftarrow 1
\]
while there is no \( j \in \{1, \ldots, i\} \) such that \( F_{2^j}(A(H, 1/2^j)) \)
\[
i \leftarrow i + 1
\]
return true

This algorithm obviously is correct. It remains to prove termination for robustly safe \( H \).

Due to Theorem 4, if \( H \) is robustly safe, then there is a strictly positive real number \( \delta \) such that also \( \left[ A(H, \delta) \right] \) is robustly safe. Moreover, due to the nature of Definition 8, also for all \( \delta' < \delta \), \( \left[ A(H, \delta') \right] \) is robustly safe. Hence we can choose \( n \) such that \( \left[ A(H, 1/2^n) \right] \) is robustly safe. Assume that Fränzle’s algorithm (that terminates for all robustly safe inputs) needs time \( t \) to prove safety of \( \left[ A(H, 1/2^n) \right] \). Eventually the above algorithm will start \( F_{2^1}(A(H, 1/2^1)) \) with \( 2^i \) being greater than \( t \) which will prove safety. \( \square \)

4 Quasi-semidecidability of falsification

In this section we will present an algorithm for falsifying safety of hybrid systems. Here we will take the assumption that all terms in the constraints defining hybrid systems are polynomial (i.e., do not contain any function symbols distinct from addition and multiplication).

We are looking for an algorithm that terminates for all robustly unsafe inputs. Recall that robustness is defined based on the notion of \( \varepsilon \)-perturbed solution sets of constraints. Note that—as a consequence of Definition 8—\( \varepsilon \)-perturbed solution sets of a flow constraint \( \dot{x} = f(x) \) correspond to \( \varepsilon \)-perturbed solution sets of \( \dot{x} \geq f(x) \wedge \dot{x} \leq f(x) \) (see also the discussion and example after the definition). Since in the latter both occurrences of \( f \) can be perturbed independently, the empty set is a \( \varepsilon \)-perturbed solution set of \( \dot{x} = f(x) \) and a corresponding perturbed hybrid system may have no flows at all, and hence be vacuously not unsafe.

This corresponds to the fact that modeling a physical system using ordinary differential equations introduces some modeling error that is not captured by the plain ODE \( \dot{x} = f(x) \). Hence we expect a user to explicitly include the possible modeling error, for example, by writing inequalities of the form \( \dot{x} \geq f(x) - \varepsilon \wedge \dot{x} \leq f(x) + \varepsilon \), for a small but non-zero real constant \( \varepsilon \).

For an algorithm for falsifying safety it suffices to abstract to a finite state system. We approximate trajectories using piecewise affine functions. We start with showing how to test whether the affine pieces fulfill the given flow constraint. Here we will use the term ”point” to denote valuations in the state space of a given hybrid system, and we call a flow \( \phi \) affine iff for every \( v \in V \), \( \phi^v \) is affine (see Definition 2, note that this means that the derivative of \( \phi^v \) is constant).

Definition 10 Two non-identical points \( p \) and \( \tilde{p} \) with \( p(\text{mode}) = \tilde{p}(\text{mode}) \) satisfy a flow constraint \( \text{Flow} \) iff there exists an affine flow \( \phi \) of length \( t \), such that \( \phi(0) = p, \phi(t) = \tilde{p} \), and for all \( s \in [0, t] \), \( \phi(s) \bullet \phi(s) \models \text{Flow} \). In such a case we also write \( p \xrightarrow{\text{Flow}} \tilde{p} \).
Input:

- a hybrid systems description 
  \((S, \text{Init}, \text{Flow}, \text{Jump}, \text{Unsafe})\),
- a strictly positive real value \(\delta\)

\[ \begin{align*}
G & \leftarrow \text{set of boxes of diameter } \delta \text{ covering the state space } S \\
\text{Init} & \leftarrow \{ s(B) \mid B \in G, s(B) \models \text{Init} \} \\
\text{Trans} & \leftarrow \{ \langle s(B), s(B') \rangle \mid B \in G, B' \in G, \\
& \quad s(B) \bullet \text{Prime}(s(B')) \models \text{Jump} \lor \text{Flow} \xrightarrow{s(B')} s(B') \} \\
\text{Unsafe} & \leftarrow \{ s(B) \mid B \in G, s(B) \models \text{Unsafe} \}
\end{align*} \]

\((\text{Init}, \text{Trans}, \text{Unsafe})\)

Fig. 3 Under-approximating abstraction

This definition requires the existence of functions (flows), that is, it contains higher-order quantifiers. Such quantifiers cannot directly be handled algorithmically. But, using the fact that the derivative of an affine flow is constant on the whole corresponding line segment, one can replace the higher-order quantifier by a first-order quantifier, that is, a quantifier over real numbers:

\[ \text{Lemma 2} \quad \text{For two non-identical points } p \text{ and } \tilde{p} \text{ such that } p(\text{mode}) = \tilde{p}(\text{mode}), \text{we have that } p \xrightarrow{\text{Flow}} \tilde{p} \text{ if there is a real constant } \lambda > 0, \text{ such that for all points } q \text{ on the line segment between } p \text{ and } \tilde{p}, \]
\[ q \bullet \{ v \mapsto \lambda(\tilde{p}(v) - p(v)) \mid v \in V \} \models \text{Flow}. \]

The check provided by this lemma is decidable due to our assumption that our constraints defining hybrid systems, and in particular the constraint \(\text{Flow}\), are polynomial [26]. Hence it can serve as a basis for an algorithm for computing under-approximating abstractions.

In this algorithm—as shown in Fig. 3—we again put a grid of a certain diameter onto the state space. Then, for each box \(B\) in this grid we choose a sample point \(s(B)\), for example, the midpoint of \(B\), and check the constraints defining the hybrid system on these sample points. Again, this check (which is undecidable for more general constraints [24]) is possible due to our restriction to polynomials.

We denote the result computed by the algorithm by \(\hat{A}(H, \delta)\). This is a finite state system \((\text{Init}, \text{Trans}, \text{Unsafe})\). An error trajectory of such a system is a sequence \(x_1, \ldots, x_n\), such that \(x_1 \in \text{Init}, x_n \in \text{Unsafe}\) and for all \(i \in \{1, \ldots, n - 1\}\), \((x_i, x_{i+1}) \in \text{Trans}\).

The algorithm is sound, that is, it in fact computes an under-approximation:

\[ \text{Theorem 7} \quad \text{For a given hybrid system description } H \text{ and } \delta > 0, \text{ if } \hat{A}(H, \delta) \text{ has an error trajectory, then also } \langle H \rangle \text{ has one.} \]

For proving a bound on the amount of under-approximation, we use the following metric on valuations.

\[ \text{Definition 11} \quad \text{The distance between two valuations } \sigma_1 \in \Gamma(X) \text{ and } \sigma_2 \in \Gamma(X) \text{ is defined by } \]
\[ d(\sigma_1, \sigma_2) \doteq \max_{v \in X} \{(d(\sigma_1(v), \sigma_2(v))\}. \]
where

- for modes $m_1, m_2 \in M$, $d(m_1, m_2) = 0$, if $m_1 = m_2$, and $\infty$, otherwise, and
- for real numbers $a_1$ and $a_2$, $d(a_1, a_2) = |a_1 - a_2|$.

Now we prove that it is possible to approximate flows arbitrarily closely by a piecewise affine function with the pieces starting and ending at grid points.

**Lemma 3** Let $\phi$ be a flow of length $t$ that is both Lipschitz and differentiable. Then, for every $\varepsilon > 0$ there is a $\delta > 0$ such that for every regular grid on the state space $S$ of mesh $\delta > 0$ there is a sequence $\psi_1, \ldots, \psi_k$ of affine flows of length $t_1, \ldots, t_k$ such that for all $i \in \{1, \ldots, k\}$,

- if $i < k$, then $\psi_i(t_i) = \psi_{i+1}(0)$,
- both $\psi_i(0)$ and $\psi_i(t_k)$ are grid elements, and
- for every point $t_\phi \in [0, t_1]$, there is $t_\phi$ in $[0, t]$ such that $d(\psi_i(t_\phi), \phi(t_\phi)) < \varepsilon$, and $d(\dot{\psi_i}(t_\phi), \dot{\phi}(t_\phi)) < \varepsilon$.

The intuition of the proof is the following: We decompose $\phi$ into segments where for each variable, the corresponding slope stays within some interval of bounded size. As a consequence, every line starting near the beginning of the segment and ending near its end has bounded distance from $\phi$. This allows us to construct a sequence of lines being close enough to $\phi$.

For formalizing this idea (see proof below) we will need the following:

**Lemma 4** For every $\delta > 0$ there is a bound $\beta_\delta > 0$ such that for every flow $\phi$ of length $t$ with $t < \delta$ and every box $D$ of width $\delta$ such that for all $s \in [0, t]$, $\dot{\phi}(s) \in D$, for every affine flow $\psi$ of length $t$ such that $d(\phi(0), \psi(0)) < \delta$ and $d(\phi(t), \psi(t)) < \delta$, the distance of $\phi$ and $\psi$, and of their derivatives, is bounded by $\beta_\delta$. Moreover, the bound $\beta_\delta$ goes to zero as $\delta$ goes to zero.

**Proof** W.l.o.g. we can assume that $d(\phi(0), \psi(0)) = d(\phi(t), \psi(t)) = \delta$. Here, we only prove the case where $\psi(0) = \phi(0) - \delta$, and $\psi(t) = \phi(t) - \delta$, the other case is dual.

Then (see Fig. 4), the maximal distance between $\phi$ and $\psi$ is bounded by the maximal distance of two line segments $\phi_a$ with domain $[0, \frac{t}{2}]$ and $\phi_b$ with domain $[\frac{t}{2}, t]$, such that
\( \phi_a(0) = \phi(0) \) and \( \phi_a \) has slope max \( D, a \) and \( \phi_b(t) = \phi(t) \) and \( \phi_b \) has slope min \( D \) (max and min are taken variable-wise). This maximal distance goes to zero with \( \delta \).

Moreover, due to the mean-value theorem, \( \phi \) attains the slope of \( \psi \) somewhere in the interval \([0, t]\). Hence, for every \( s \in [0, t] \), \( \psi(s) \in D \), and since also \( \phi(s) \in D \) the distance \( d(\dot{\phi}(s), \dot{\psi}(s)) \) is bounded and goes to zero with \( \delta \). \( \square \)

Now we are ready to prove Lemma 3:

**Proof** Let \( \varepsilon > 0 \) be arbitrary, but fixed. Let \( \delta \) be such that the \( \beta_\delta \) ensured by Lemma 4 is smaller than \( \varepsilon \).

Due to Lipschitz continuity of \( \phi \) there are \( t_1, \ldots, t_k \) and boxes \( D_1, \ldots, D_k \) such that

- \( 0 = t_0 < t_1 < \cdots < t_k = t \)
- for every \( i \in \{1, \ldots, k\} \), \( t_i - t_{i-1} < \delta \),
- for every \( i \in \{1, \ldots, k\} \) the box \( D_i \) has width \( \delta \) and for all \( t \in [t_{i-1}, t_i] \), the vector \( \dot{\phi}(t) \in D_i \).

Take a grid of mesh \( \delta \), and construct \( \psi_1, \ldots, \psi_k \) as the sequence of affine flows such that for every \( i \in \{0, \ldots, k\} \), \( \psi_i \) has length \( t_i - t_{i-1} \), and the \( i \)-th vertex consists of a grid element close to \( \phi(t_i) \). Due to Lemma 4 the distance between \( \phi \) and \( \psi_1, \ldots, \psi_k \) is bounded by \( \beta_\delta \), and hence also by \( \varepsilon \). \( \square \)

Now we observe that if a valuation \( x' \) is sufficiently close to a valuation \( x \) that robustly satisfies a constraint, than \( x' \) satisfies the constraint also:

**Lemma 5** Let \( C \) be a constraint and let \( \varepsilon > 0 \). Let \( x' \) be such that there is an \( x \) with \( d(x, x') \leq \varepsilon \), such that \( x \) is an element of all sets that fulfill \( C \) up to \( \varepsilon \). Then \( x' \models C \).

**Proof** Since \( x \) is in all sets that fulfill \( C \) up to \( \varepsilon \), not only \( x \models C \), but also \( x \models C^* \), if \( d(C, C^*) \leq \varepsilon \). Hence, for all \( x' \) with \( d(x, x') < \varepsilon \), \( x' \models C \). \( \square \)

Now we can now state the main theorem of this section:

**Theorem 8** If a hybrid system description \( H \) is robustly unsafe, then there is a \( \delta > 0 \) such that \( \dot{A}(H, \delta) \) is unsafe.

Before proceeding with the proof of this theorem we note once more that—according to Definition 5—a hybrid system description \( H \) is robustly unsafe iff there is a real number \( \varepsilon > 0 \) such that all elements of \( [H]_\varepsilon \) are unsafe. The elements of \( [H]_\varepsilon \) are the hybrid systems \( H \) fulfilling the hybrid system description \( H \) up to \( \varepsilon \). As already discussed (e.g., directly after Definition 9), this does not necessarily mean that \( H \) fulfills a hybrid system description that is the result of perturbing \( H \). In particular, as discussed at the beginning of this section, a flow constraint of the form \( \dot{x} = c \) has the empty set as a perturbed solution set.

We now prove Theorem 8:

**Proof** We assume that the finite state system \( \dot{A}(H, \delta) \) has the form \( (\text{Init}, \text{Trans}, \text{Unsafe}) \). Let \( H \) be robustly unsafe with robustness margin \( \varepsilon \). Let \( \phi_1, \ldots, \phi_p \) be a robust error trajectory of \( H \), that is, a trajectory that is an error trajectory of all \( H \) that fulfill \( H \) up to \( \varepsilon \).

Such a robust error trajectory of \( H \) exists due to the following observation: Consider a constraint \( C \). Let \( x \) be such that for all \( C^* \) with \( d(C, C^*) \leq \varepsilon \), \( x \models C^* \). Due to Definition 8,
such an \( x \) is in every \( \varepsilon \)-perturbed solution set of \( C \). The hybrid system containing, for every defining constraint, all those \( x \), has an error trajectory. This is the common error trajectory we need.

Now let \( l_1, \ldots, l_p \) be the lengths of \( \phi_1, \ldots, \phi_p \). For each \( i \in \{1, \ldots, p\} \), \( \phi_i \) satisfies the assumptions of Lemma 3 which ensures a \( \delta_i > 0 \) corresponding to our robustness margin \( \varepsilon \). Choose \( \delta \) as \( \min \{\delta_1, \ldots, \delta_i\} \). We will construct an error trajectory of \( \bar{A}(\mathbb{H}, \delta) \).

Take a grid of mesh \( \delta \). By Lemma 3 we know that there is a sequence of affine flows \( \psi_1, \ldots, \psi_k \) of lengths \( l'_1, \ldots, l'_k \) whose end-points are grid elements, and such that for every \( i' \in \{1, \ldots, k\} \) and \( t' \in [0, l'_{i'}] \) there is an \( i \in \{1, \ldots, p\} \) such that \( d(\phi_i(t), \psi_{i'}(t')) < \varepsilon \), and \( d(\dot{\phi}_i(t), \dot{\psi}_{i'}(t')) < \varepsilon \). Hence, by robustness of \( \mathbb{H} \), and Lemma 5, \( \psi_i(0) \xrightarrow{\text{Flow}} \psi_i(l'_{i'}) \), and so \( \langle \psi_i(0), \psi_i(l'_{i'}) \rangle \in \text{Trans} \).

Moreover, due to similar reasoning, for every \( i' \in \{1, \ldots, k-1\} \), \( \langle \psi_{i'}(l'_{i'}), \psi_{i'+1}(0) \rangle \in \text{Trans} \), \( \psi_0(0) \in \text{Init} \), and \( \psi_k(l'_{k}) \in \text{Unsafe} \). Hence the endpoints of \( \psi_1, \ldots, \psi_k \) form an error trajectory of \( \bar{A}(\mathbb{H}, \delta) \).

This result, and the fact that \( \bar{A}(\mathbb{H}, \delta) \) is a finite system and hence algorithmically checkable, together with Theorem 6 proves the main theorem of the paper, as stated at the end of Sect. 2.

5 Related work

A recent article [4, Sect. 5] includes a survey on the role of noise and robustness in continuous-time dynamical systems.

Similar quasi-decidability results as the ones presented in the present paper have been obtained (under different names) for systems with simpler dynamics: Fränzle [10, 11] provides results for the case where the input system is completely defined by polynomials. Especially, continuous evolution is given by explicit polynomial flows which, in general, does not even allow the modeling of linear differential equations, since these can have non-polynomial flows as solutions.

Puri and co-authors [21] show how to compute an over-approximation of Lipschitz differential inclusions with known Lipschitz constant over a finite time horizon. This implies a corresponding quasi-decidability result. In contrast to that, our result allows unbounded time, and does not require a previously known Lipschitz constant.

Collins [7] studies approximation of reach sets of dynamical system in an effective computable analysis framework which again implies a corresponding quasi-decidability result. He uses a discrete time model (such a model can in certain cases encode a continuous time model). In the continuous time case there is corresponding work on approximating reach sets over a finite time horizon [8].

Damm and co-authors [9] provide a similar result as ours for a discrete time model. The continuous time model employed in this paper, implies several additional difficulties:

- When considering syntactic descriptions of systems, in a discrete time model all variables vary over the state space of the system, whereas in a continuous time model, some variables (describing differentiation) do not. Hence these variables may take unbounded values even if the state space is bounded. This needs additional deduction mechanisms for capturing the set of possible values that these variable may take and proofs of their correctness (Theorem 2) and convergence (Lemma 1).
In a discrete time model, a trajectory only reaches finitely many states in a finite time interval, whereas in a continuous time model it usually reaches uncountably many. This uncountable set has to be captured by corresponding algorithms. As a consequence, in the case of verification, abstraction to a finite state systems, as used in the earlier paper, cannot capture system behavior arbitrarily closely, since even arbitrary refinements cannot separate two subsequent steps of the system. In the case of falsification, instead of just having to consider finitely many points (due to state space compactness and discrete time), we have to bound the distance of the abstraction to uncountably many points on an error trajectory.

Studying the effect of perturbations on dynamical systems is a classical research topic for continuous systems, as a summary see for example the textbook by Khalil [17]. However, only recently such have such questions received broader attention in the case of hybrid [13] or even completely discrete systems [1].

On the negative side, Henzinger and Raskin showed that certain undecidability results for hybrid systems continue to hold, even if in the proof one only allows encodings into robust trajectories [16]. This does not contradict our result for two reasons: First, quasi-decidability allows an algorithm that does not always (i.e., for non-robust inputs) terminate, whereas undecidability (even when based on robust trajectories) proves non-existence of an algorithm that terminates always. Second, in a similar way as Fränzle [10, 11], we require a compact state space, whereas Henzinger and Raskin do not (although their dynamics is much simpler than ours).

Regarding falsification, recent work [3, 6] explores so-called resolution-complete simulation algorithms. These give some completeness assurance but miss a few elements for a full quasi-decidability proof (e.g., the algorithms assume a Lipschitz constant on the function defining the differential equation, and they ignore errors due to time-discretization).

6 Conclusion

We proved that safety-verification of non-linear hybrid systems is quasi-decidable. Some of the algorithms used in the proof of quasi-decidability are not efficient in practice (especially checking robust rectangular hybrid systems). It remains an open problem to find verification algorithms that terminate for robust hybrid systems and are efficient in practice. Also, it is open, whether quasi-decidability holds even in the case of a non-compact state space.

A further interesting question is the precise relationship between the syntactic perturbations used in this paper and approaches studying perturbation of hybrid systems based on set-valued analysis [2], especially the application of corresponding results around well-posedness questions for hybrid systems [12, e.g.].

References

49(2):890–925
Princeton University Press, Princeton
theory applications. Automatica 42(4):573–587
Trans Autom Control 43:540–554
Syst Sci 57:94–124
TA, Sonntag ED (eds) Hybrid systems. LNCS, vol 1066. Springer, Berlin
2010: 7th annual conference on theory and applications of models of computation. LNCS, vol 6108.
Springer, Berlin, pp 397–408
J Symb Log 33:514–520
against clock-drift. In: 5th ifip int conf on theoretical comp sc, pp 537–553
26. Tarski A (1951) A decision method for elementary algebra and geometry. University of California Press,
Berkeley. Also in [5]
Chapter 6

Guaranteed Termination in the Verification of LTL Properties of Non-linear Robust Discrete Time Hybrid Systems

This chapter appeared as:

GUARANTEED TERMINATION IN THE VERIFICATION OF LTL PROPERTIES OF NON-LINEAR ROBUST DISCRETE TIME HYBRID SYSTEMS*

WERNER DAMM and GUILHERME PINTO

Department für Informatik, Carl von Ossietzky Universität Oldenburg,
26111 Oldenburg, Germany
werner.damm@informatik.uni-oldenburg.de

STEFAN RATSCHAN

Institute of Computer Science, Czech Academy of Sciences
Pod Vodárenskou věží 2
182 07 Praha 8
Czech Republic
stefan.ratschan@cs.cas.cz

Received 23 March 2006
Accepted 10 November 2006
Communicated by Yih-Kuen Tsay

We present a novel approach to the automatic verification and falsification of LTL requirements of non-linear discrete-time hybrid systems. The verification tool uses an interval-based constraint solver for non-linear robust constraints to compute incrementally refined abstractions. Although the problem is in general undecidable, we prove termination of abstraction refinement based verification and falsification of such properties for the class of non-linear robust discrete-time hybrid systems. We argue, that—in industrial practice—safety critical control applications give rise to hybrid systems that are robust. We give first results on the application of this approach to a variant of an aircraft collision avoidance protocol.

Keywords: Non-linear hybrid systems; abstraction refinement; constraint-solving; model-checking; robustness.

1. Introduction

This paper significantly extends previous semi-decidability results for LTL verification of non-linear discrete time hybrid systems with real-valued variables. Even

*This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS). See www.avacs.org for more information.
though this problem is in general undecidable (by reduction from 2 counter machines), we show, that by exploiting the natural property of robustness of real-life hybrid systems, an abstraction-refinement based approach—employing both approximations from above and below—is guaranteed to terminate, either establishing the truth of the requirement, or exhibiting a concrete counterexample, even for non-linear hybrid systems\(^a\). Although robustness cannot be checked algorithmically, we argue, that every reasonable designed system occurring in practice, is in fact robust, allowing our algorithm to terminate for all practically relevant problem instances.

In contrast, results from Fränzle [15, 16]—also based on robustness arguments—only handle polynomial flows (in a dense time setting). Our approach also improves over other approaches to hybrid systems verification [11, 31, 3, 29, 2, 1, 17, 32] in that termination is guaranteed even for a very rich class of models. Indeed, this is the first paper providing an algorithm with termination proof for full LTL properties for discrete time hybrid systems with non-linear dynamics, which are neither restricted to be controllable, nor to have only finite precision real-valued variables. The only underlying assumption of robustness is close to the one used by Girard and Pappas [17], in that validity may not depend on small perturbations of system variables. In contrast to [17] our notion of robustness applies to the richer setting of non-linear constraints and full LTL verification of the system model.

The presented approach primarily targets safety critical control applications, such as collision avoidance systems, where designs must guarantee separation of traffic agents by safety margins even in the presence of noise and (bounded) inaccuracies of sensors and actuators, and hence robustness of designs is a key requirement. Intuitively, for such applications, small variances in measurements or small deviations of actuator settings may not lead to a violation of safety margins between traffic agents. We will give a formal definition of this intuitive concept of robustness, which will be instrumental in establishing termination. Even though robustness cannot be decidable, robustness is "designed into" industrial safety critical applications: Development processes for such applications enforce various measures (such as simulation- or rapid-prototyping based impact analysis of noise, sensor- and actuator inaccuracies) to ensure that deployed designs meet such robustness requirements. Design models used for actual deployment, for example in airborne applications will thus have passed through maturity gates checking that all measures necessary for ensuring robustness have been performed.\(^b\). The termination argument of our approach thus applies to practically all industrially deployed safety critical applications. For models which do not meet the robustness assumptions the algorithm may fail to terminate, but it will never produce an incorrect answer.

\(^a\)We use the term "hybrid systems" for system models that integrate two different models of execution (state-transition systems for control, and difference equations or differential equations for the plant). The results of this paper apply to the special case of discrete-time hybrid systems.

\(^b\)We have validated this statement with our industrial cooperation partners in the automotive and avionics domain, see www.safeatrans.de
The approach will be illustrated by an air traffic conflict resolution example [34], where aircrafts follow circular trajectories along opposite directions, leading to a non-linear hybrid system.

As mathematical model we use discrete time hybrid automata, which in each time step of fixed duration update a set of real-valued variables as determined by assignments occurring as transition labels, allowing possibly non-linear arithmetic expressions. This subsumes the capability to describe the evolvment of plant variables by difference equations. Transition guards can be non-linear arithmetic constraints. Steps of the automata are assumed to take a fixed time-period (also called cycle-time), intuitively corresponding to the sampling period of the control unit, and determine the new mode and new outputs (corresponding to actuators) based on the sampled inputs (sensors). We allow arbitrary first-order LTL formulas as requirements. Atoms are arithmetic constraints over the real-valued variables of the hybrid automaton, thus allowing to both express response time requirements (such as “when crash sensor is activated, the airbag will be ignited within 3 cycles”), stability properties such as “the aircraft will be maintained at preselected height”, as well as safety properties (such as “the distance between two aircraft will always be greater than four miles”).

The decision to base our analysis on discrete time models of hybrid systems is motivated from an application perspective. Industrial design flows for embedded control software entail a transition from continuous time models in early analysis addressing control law design, to discrete time models as a basis for subsequent code generation. Typically, the control-law design is carried out in CASE tools such as Matlab-Simulink in a continuous time setting, validating both stability and safety requirements. Once these key requirements are established, engineers determine sampling rates using standard text book methods (e.g., [22]), and then informally validate the discretized model, typically by simulation. For example, if design engineers have decided to implement a certain control law as a periodic task with periodicity δ, then simulation would be used to "verify" that both stability of the controller as well as safety requirements are maintained in spite of the now limited observability of the plant at the chosen sampling rate δ. The methods of this paper replace this informal validation. They allow to formally prove for models satisfying the robustness assumptions that—even under the limited discrete time visibility of the plant—LTL requirements, and thus both stability properties as well as safety requirements, are guaranteed by the controller.

Related work for discrete time model for hybrid systems focusses on restricting the system model in order to guarantee theoretical decidability for verification by constructing a suitable finite abstraction. In [2, 1] this is done by requiring finite precision values in the observations; in [32] this is achieved by considering only controllable systems (where any state may be reached from any state). Recently (at the same time as the conference version of this paper), an abstraction/refinement approach was proposed [17] to solve safety verification problems for linear discrete
time hybrid systems based on robustness assumptions close to ours. In comparison, we consider full LTL verification of non-linear systems. In most cases, the literature on abstraction and refinement for hybrid systems considers a continuous time model \cite{11, 30, 18, 31}. Because of this, a direct comparison of the employed algorithms is not possible. We can, nevertheless, observe some differences concerning the way they compute the abstraction.

While, for example, also tools such as Hypertech \cite{19} and Checkmate \cite{10} do support analysis of non-linear systems, with Checkmate offering the highly optimized flow-pipe representation technique, none of these is guaranteed to terminate for robust models when proving temporal properties of hybrid systems. We also note the potential unsafeness of the Checkmate approach in the construction of the abstract transition relation due to rounding errors – in contrast, our constraint solver guarantees, that rounding errors are conservatively over-approximated in refinement steps. In contrast to \cite{11} we are able to prove a termination criterion for abstraction refinement, and can provide safe abstractions for both verifying and falsifying safety properties. The high potential of interval-based evaluation methods for hybrid system verification has already been demonstrated \cite{31, 28, 29} for a more restricted logic, and without termination guarantees.

There are mechanisms for approximating non-linear continuous-time hybrid systems by rectangular automata arbitrarily closely \cite{18}. However, the approximation has to be done manually, and even verifying only rectangular safety properties on the resulting approximations is still an undecidable problem.

The paper is organized as follows: Section 2 elaborates the notion of robust hybrid systems, leading to a new notion of robust satisfaction and robust falsification of LTL properties; Section 3 shows how to approximate robust satisfaction from above and robust falsification from below by exact satisfaction/falsification on finite approximations, and proves, that for any property that is robustly satisfied (falsified) by a (non-linear) robust hybrid system, one can find a finite approximation that establishes this fact; based on this it shows the termination of an abstraction refinement algorithm for the verification and falsification of LTL properties of discrete-time hybrid systems; Section 4 concludes the paper.

2. Robust Hybrid Systems

The authors have substantial experience in analyzing industrial control unit designs for automotive \cite{4, 6, 8, 14} and avionics applications \cite{5, 25}, among others in the context of the competence cluster SafeTRANS (www.safeTRANS.de.org), and the Integrated Project SPEEDS funded by the European commission under contract number AIP5-CT-2006-033471. Based on this industrial background, we derive the following observations:

- For any sensor inputs, a combination of filtering, plausibility checking and voting will be used to derive what is often called validated inputs.
- This preprocessing will in particular guarantee a minimal separation be-
tween values assumed by validated inputs, in the following sense: assume, that \( v \leq 5 \) appears as guard of a transition, then altering the guard to \( v \leq 5 \pm \varepsilon \) for some \( \varepsilon \) smaller than a sensor-dependent constant does not change the mode-switching behavior of the system.

- To take into account noise on actuators and un-modeled disturbances, the controller will enforce a safety margin, separating all legal undisturbed runs from forbidden plant regions by some minimal application dependent constant (catering for noise and disturbances). To this end, deviations induced by disturbances and noise on actuators are detected using closed-loop control, and correcting actions to avoid forbidden states are designed to cater for this difference between ideal and measured trajectories.

Designers hence solve the task to guarantee a safety property \( \varphi \) even in the presence of noise on sensors and actuators and un-modeled disturbances. This entails, that the classical notion of satisfiability is in fact too weak. What is called for, is a notion of \textit{robust satisfiability}, which guarantees \( \varphi \) even in the presence of small bounded uncertainties. In the remainder of this section, we will derive a formal definition of such a notion of robust satisfiability.

We use constraints over the current values and the values at the next time step to specify the transition relation. We also use constraints to explicate the predicates observable on the hybrid system, which define the atomic predicates to be used in first-order LTL requirement specifications on our systems. In addition to variables ranging over reals we will use special \textit{mode variables} written in sans serif font that range over symbolic constants denoting modes of hybrid systems.

**Definition 1.**

- An arithmetic expression is a term (in the predicate-logical sense) with function symbols in \{+; \times; \div; \sin, \cos, \exp\}.
- An atomic arithmetic state space constraint is of the form \( e \rel r c \), where \( e \) is an arithmetic expression, \( r \in \{\neq, =, <, >, \leq, \geq\} \) is a relational operator, and \( c \) is a real-valued constant.
- A mode constraint is an expression of the form \( \text{mode} = m \), where \( m \) is a symbolic constant and \( \text{mode} \) a mode variable.
- A state space constraint is a Boolean combination of atomic arithmetic state space constraints and mode constraints.

For formally modeling discrete time hybrid systems we assume a finite set \( M \) of symbolic constants called \textit{modes} and a finite set \( X = \{x_1, \ldots, x_k\} \) of \textit{real-valued variables} (in the formal development, we do not further distinguish between sensors, variables, and actuators). We assume that the real-valued variables range over bounded intervals \( \{I^{x_1}, \ldots, I^{x_k}\} \). Moreover, for specifying the variable values at the next time step we will denote by \( X' \) the primed versions of the variables in \( X \).

Note that in an engineering context one usually knows bounded intervals for the reasonable values of real variables. Our method can also be used to check that these
intervals are never left, by formulating a corresponding safety verification problem that shows that no variable ever reaches a value too close to the boundary of its interval.

**Definition 2 (discrete time hybrid system)** A discrete time hybrid system $S$ is a tuple $S = (\tau, \pi_0, \pi_1, \ldots, \pi_k, \delta)$ where

- $\tau$ is a disjunction of state space constraints of the form $mode = m \land guard \land mode' = m' \land transitions$ where
  - $m \in M$ and $m' \in M$,
  - $guard$ is a conjunction of atomic arithmetic state space constraints that contains only variables in $X$,
  - $transitions$ is a conjunction of atomic arithmetic state space constraints that contains only variables in $X \cup X'$.
- $\pi_0$ is a state space constraint containing only variables in $X \cup \{\text{mode}\}$, restricting the initial valuation, and
- $\pi_1, \ldots, \pi_k$ are additional state space constraints containing only variables in $X \cup \{\text{mode}\}$, that we will later use in LTL queries (the observed propositions), and
- $\delta$ is the sampling rate in time units, a positive real number.

Discrete time hybrid systems are sufficiently expressive to express both plant dynamics as well as (possibly hybrid) controllers. Time is modeled implicitly, in that each step corresponds to a fixed unit delay $\delta$, as motivated in the introduction.

Our example is a discretized variant of an aircraft collision avoidance protocol that we took from [19] (originally developed in [34]), exhibiting non-linear dynamics. Two aircraft, flying in a straight line and orthogonal trajectories at the same altitude, initiate a collision avoidance maneuver when the distance between them reaches 8 miles. Both aircraft turn 90 degrees to the right and start a semi-circle trajectory to the left, as shown on the left-hand side of Figure 1, with fixed angular velocities. The linear velocity is also fixed and the same for both aircraft. After completing the semi-circle, they resume their original trajectories.

The continuous time dynamics during the collision avoidance maneuver is given by

$$
\dot{x}_r = -v_1 + v_2 \cos \phi_r + \omega_1 y_r \\
\dot{y}_r = v_2 \sin \phi_r - \omega_1 x_r \\
\dot{\phi}_r = \omega_1 - \omega_2
$$

where $x_r$ and $y_r$ are the relative planar coordinates of aircraft two relative to aircraft one, $\phi_r$ is the angle between the vector speed of aircraft two relative to the vector speed of aircraft one, and $v_1$ ($v_2$) and $\omega_1$ ($\omega_2$) are the linear respectively angular velocity of aircraft 1 (aircraft 2), as illustrated by the right-hand side of Figure 1. Please consult [34] for the non-trivial derivation of the system dynamics. For the
purpose of this paper we further specialize the dynamics by choosing a constant linear velocity \( v_1 = v_2 = 1 \) and angular velocities \( \omega_1 = 1 \) and \( \omega_2 = 0.95 \). With these instantiations, the model specializes to

\[
\begin{align*}
x_r &= -1 + \cos \phi_r + y_r \\
y_r &= \sin \phi_r - x_r \\
\dot{\phi}_r &= 0.05
\end{align*}
\]

We are interested in analyzing discretized versions of this collision avoidance strategy under a fixed sampling rate \( \delta \) with respect to the overall safety requirement, that the distance between the two aircraft never becomes smaller than 4 miles. In a typical design flow, this corresponds to the situation, where the time model in Matlab-Simulink is now changed from continuous time to discrete time, in order to analyze the impact of discretization in maintaining the collision avoidance requirement. To this end, we derive a discrete time version using Euler discretization wrt. the chosen sampling rate \( \delta \), by replacing each differential equation \( \dot{v} = f(v) \) by \( v' = v + \delta \cdot f(v) \), yielding

\[
\begin{align*}
\phi_r' &= \phi_r + \delta \cdot (0.05) \land \ x_r' &= x_r + \delta \cdot (y_r - 1 + \cos \phi_r) \land \ y_r' &= y_r + \delta \cdot (\sin \phi_r - x_r)
\end{align*}
\]

where \( \delta \) is the sampling period of the controller in seconds. We analyze the system for an initial region with \( \phi = 1.57 \land x^2 + y^2 = 64 \land x > 0 \land y < 0 \), which models an initial configuration before entering the collision avoidance maneuver (before the right-turn of the aircrafts) at a distance of 8 miles, restricted to the cases where the intersection of the trajectories lies ahead of both aircraft.

**Definition 3.**

- A valuation \( \sigma \) is a mapping that assigns a mode in \( M \) to the mode variable mode and, for each \( 1 \leq i \leq k \) a real value in \( \mathbb{R}^{x_i} \) to the variable \( x_i \). We denote the set of all valuations by \( \Sigma \).
- We denote by \([\pi]\) the set of all valuations satisfying a state space constraint \( \pi \), and similarly by \([\pi']\) the set of pairs of valuations \( (\sigma, \sigma') \) satisfying the
transition constraint $\tau$, where primed (resp. unprimed) variables are interpreted over $\sigma'$ (resp. $\sigma$).

- Given a set $\Gamma$, we call a tuple $(-, Q_0, Q_1, \ldots, Q_k)$, with $\rightarrow \subseteq \Gamma \times \Gamma$, and $Q_0, \ldots, Q_k \subseteq \Gamma$, an (extended) transition system over $\Gamma$.
- Given a hybrid system $H$ of the form $(\tau, \pi_0, \pi_1, \ldots, \pi_k, \delta)$ we denote by $[H]$ the transition system $([\tau], [[\pi_0]], \ldots, [[\pi_k]])$ over $\Sigma$.
- A run of a system $(\tau, \pi_0, \pi_1, \ldots, \pi_k, \delta)$ is a mapping $\theta : \mathbb{N} \rightarrow \Sigma$ such that for all $t \in \mathbb{N}$, $(\theta(t), \theta(t+1)) \in [\tau]$.

We use first-order LTL formulas such as $G \neg x \geq 10$ to formalize requirements on discrete time hybrid systems. Still, the results of this paper hold for any temporal logic using only universal path quantifiers, such as ACTL*. Since steps have a defined duration, real-time constraints can be expressed using the next-time operator. As atoms we allow the observed propositions $\pi_0, \ldots, \pi_k$. We employ standard syntax and semantics of LTL as can be found in various textbooks [13]—the needed adaption to our definition of extended transition system is a trivial exercise. Especially we write $T \models \varphi$ to signify that the extended transition system $T$ satisfies $\varphi$.

Note that we do not treat the state space constraint $\pi_0$ that specifies the initial states in any special way (e.g., by allowing only runs that start in an initial state). Instead, we encode initial states into the queries by using LTL formulae of the form $\pi_0 \rightarrow G \varphi$ (i.e., $\neg \pi_0 \lor G \varphi$).

Robustness of a hybrid system $S$ is defined relative to a temporal specification $\varphi$: it requires, that the validity of $\varphi$ does not depend on small perturbations of $S$. The formal definition is based on a metric between arithmetic constraints [26]. For $S$ to be robust wrt. $\varphi$ requires the existence of a bound $\varepsilon$, such that if $\varphi$ holds in $S$, then it must also hold in any $S'$ defined by constraints that have distance at most $\varepsilon$ from the constraints defining $S$. Intuitively, this entails that avoiding forbidden plant states may not depend on small inaccuracies of sensors or actuators. Indeed, controller designs in which changing a guard of the form $x \leq c$ to $x \leq (c+\varepsilon)$ (mirroring sensor inaccuracy) or changing an actuator setting from $a' = e$ to an assignment $a' = e \pm \varepsilon$ (modeling a small error in actuator settings) causes forbidden states to be reached would not be acceptable (and not “robust”, under our formal definition).

We now define these concepts more formally:

**Definition 4.**

- The distance between two valuations $\sigma_1, \sigma_2$ is defined by $d(\sigma_1, \sigma_2) \triangleq$
  - $\infty$, if $\sigma_1(\text{mode}) \neq \sigma_2(\text{mode})$, and
  - $\max\{d(\sigma_1(x), \sigma_2(x)) \mid x \in X\}$, where $d(a, b) \triangleq |a - b|$, otherwise.
- The distance between two atomic arithmetic constraints $e \leq r \leq c$ and $e' \leq r' \leq c'$ (we assume that all arithmetic constraints have been brought into this form) is defined by $d(e \leq r \leq c, e' \leq r' \leq c') \triangleq \infty$, if $e \neq e'$ or $r \neq r'$, and $d(e, c')$, otherwise.
• The distance between two mode constraints \( \text{mode} = m_1 \) and \( \text{mode} = m_2 \) is \( \infty \) if \( m_1 \neq m_2 \) and 0, otherwise.

• The distance between two constraints \( \phi, \phi' \) is defined by \( d(\phi, \phi') \) as:
  
  - \( \infty \), if \( \phi \) and \( \phi' \) have a different Boolean structure or do not have mode constraints at the same places, and
  
  - the maximum of the distance between two corresponding atomic (arithmetic or mode) constraints, otherwise.

The key definition of this paper, reported below, captures our intuition that safety properties should be guaranteed even under disturbances, as long as these are bounded by some \( \varepsilon \). To this end, we define a non-standard semantics of discrete time hybrid systems that allows transitions that miss the original transition predicate only by a slight margin below some \( \varepsilon \). For a safety property to be robustly satisfied, there must exist a degree of perturbation \( \varepsilon > 0 \) such that the safety property is true in all \( \varepsilon \)-perturbed systems.

We start with defining the notion of an \( \varepsilon \)-perturbed solution set by stating a condition when \( x \) is allowed to be in such a solution set, and a condition when \( x \) is allowed to be not in such a solution set. In both cases we require the existence of some perturbations establishing the corresponding fact.

**Definition 5.** A set \( P \) is an \( \varepsilon \)-perturbed solution set of a constraint \( \phi \) iff

- for every \( x \in P \), there is a constraint \( \phi^* \) with \( d(\phi, \phi^*) \leq \varepsilon \) and an \( x^* \) with \( d(x, x^*) \leq \varepsilon \) such that \( x^* \models \phi^* \), and

- for every \( x \notin P \), there is a constraint \( \phi^* \) with \( d(\phi, \phi^*) \leq \varepsilon \) and an \( x^* \) with \( d(x, x^*) \leq \varepsilon \) such that \( x^* \not\models \phi^* \).

In each item of this definition we employ two types of perturbations: a syntactic perturbation of the constraint and a semantic perturbation in the solution space. The motivation for the use of syntactic perturbation lies in the fact that two systems that show the same behavior when corresponding exactly to their model, might show radically different behavior under perturbations. This can be seen, for example, on the constraints \( 0 = 0 \) and \( 0 \leq 1 \) which have the same solution set but behave radically different when perturbing the zero on the left-hand sides of these constraint. The motivation for the use of semantic perturbation lies in the fact that we also want to model drift that external effects might introduce into the system. Moreover, this second type of perturbation also simplifies the proofs in the rest of the paper significantly.

Probably the reason why some authors only consider one of these two types of perturbations (e.g., only semantic [17]) is the fact that in the context of robustness considerations both perturbations behave differently only for special cases (e.g., the constraint \( 0 = 0 \), whose solution set vanishes under syntactic perturbations, but stays the same under semantic perturbations).

Definition 5 extends to hybrid systems as follows:
**Definition 6.** A transition system \((\rightarrow, Q_0, Q_1, \ldots, Q_k)\) is an \(\varepsilon\)-perturbed manifestation of a hybrid system \((\tau, x_0, x_1, \ldots, x_k)\) iff \(\rightarrow\) is an \(\varepsilon\)-perturbed solution set of \(\tau\), and for each \(i \in \{0, \ldots, k\}\), \(Q_i\) is an \(\varepsilon\)-perturbed solution of \(x_i\).

This allows us to define robustness of a hybrid system relative to a temporal specification.

**Definition 7.** An LTL formula \(\varphi\) is satisfied by a hybrid system \(S\) with robustness \(\varepsilon\) \((S \models_{\varepsilon} \varphi)\) iff for all \(\varepsilon\)-perturbed manifestations \(T\) of \(S\), \(T \models \varphi\). An LTL formula \(\varphi\) is robustly satisfied by \(S\) \((S \models_{\varepsilon} \varphi)\) iff there is an \(\varepsilon > 0\) such that \(S \models_{\varepsilon} \varphi\).

For example, a system that starts in state \(x = 0\) and evolves according to the transition constraint \(x' = x\), satisfies the LTL formula \(G \neg x \geq 1\), but does not robustly satisfy it, because any transition constraint of the form \(x' = x + \varepsilon\), with \(\varepsilon > 0\), will eventually violate the constraint \(\neg x \geq 1\). On the other hand, a system that evolves according to the transition constraint \(x' = x - 1\), robustly satisfies this LTL formula.

**Definition 8.** An LTL formula \(\varphi\) is falsified by a hybrid system \(S\) with robustness \(\varepsilon\) \((S \not\models_{\varepsilon} \varphi)\) iff for all \(\varepsilon\)-perturbed manifestations \(T\) of \(S\), \(T \not\models \varphi\). An LTL formula \(\varphi\) is robustly falsified by \(S\) \((S \not\models_{\varepsilon} \varphi)\) iff there is an \(\varepsilon > 0\) such that \(\varphi\) is falsified by \(S\) with robustness \(\varepsilon\).

For example, a system that starts in a state fulfilling \(0 \leq x \leq 1\), and evolves according to the transition constraint \(x \leq x' \land x' \leq x + 1/10\) robustly falsifies the LTL formula \(G \neg x \geq 10\).

It is worthwhile to reflect on the nature of the two simple examples of non-robust satisfaction respectively falsification, by setting them into an application perspective. Both examples would indeed fail short of being accepted in industrial settings, because they fail to compensate the possible effect of disturbances and noise. Consider, for example, a speed monitoring system for a train. The example following Definition 7 would correspond to a design, in which the current controlling the engine is set to maintain the targeted speed, without checking that the actual speed of the train is meeting the targeted speed. Unmodelled disturbances, such as the slope of the track, can easily cause the actual speed to grow beyond the critical maximal speed (potentially causing derailing of the train): just consider the situation, where the train is running through a segment of the track with a constant negative slope, adding in each step an \(\varepsilon\) unintended increment to the current speed. It is exactly for this reason that no control-engineer would rely on open-loop control for such applications; indeed, by providing a feedback-loop compensating for a possible difference between the intended set point (where \(x\) is to remain unchanged forever), and the real value in the physical system, the unintended growth of \(x\) would be detected and compensated. Also the example illustrating Definition 8 is an open-loop system.
3. Effective Construction of Finite Abstractions with Bounded Imprecision

Our approach follows the abstraction refinement paradigm. In contrast to previous approaches, we are able to prove termination of the refinement loop under the assumption that the analyzed model is robust. In this section we introduce the key instrument—a bound on the degree of imprecision introduced by abstraction. By proving that incremental refinements make the degree of imprecision converge to zero, any desired degree of precision can be reached. We also show in this section, that such abstractions can be efficiently computed even for non-linear hybrid systems, using interval arithmetic. The last part of this section puts all pieces together in defining an algorithm for proving robust first-order LTL properties and proving its termination.

From now on, we fix a discrete time hybrid system $S = (\tau, \pi_0, \pi_1, \ldots, \pi_r, \delta)$, and a LTL requirement $\varphi$ on $S$ over the atoms $\pi_0, \ldots, \pi_r$. For the rest of the development, it will be convenient to assume, that negations occur only in literals, and that all atoms appear under the scope of a negation (this can easily be achieved by adapting the relational operators in arithmetic constraints). This allows us to over-approximate the behavior of a hybrid system by over-approximating the observed propositions $\pi_0, \ldots, \pi_r$ in the same direction as the transition relation $\tau$, allowing more uniformity in the algorithms and proofs. So, by over-approximating the solution set of $\pi_0$ and $\pi_1$ in a query of the form $G(\neg \pi_0 \lor F \neg \pi_1)$, we under-approximate the literals $\neg \pi_0$ and $\neg \pi_1$.

We use abstractions that approximate the behavior of the original system, and then we measure the approximation error introduced by these abstractions.

**Definition 9.** Let $T$ be a transition system over $\Gamma$ of the form $(\rightarrow, Q_0, \ldots, Q_r)$ and let $T'$ be a transition system of the form $(\rightarrow', Q'_0, \ldots, Q'_r)$ over $\Gamma'$. Then $T'$ abstracts $T$ ($T' \succeq T$, $T \preceq T'$) iff there is a function $H$ (the abstraction function) such that

- for all $i \in \{0, \ldots, r\}$, for all $s \in Q_i$, $H(s) \in Q'_i$, and
- for all $s, s_1$ with $s \rightarrow s_1$, $H(s) \rightarrow H(s_1)$.

Clearly, for transition systems $T$ and $T'$ such that $T \subseteq T'$ (with $\subseteq$ defined element-wise) $T \preceq T'$ (e.g., using the identity abstraction function). Moreover, the relation $\preceq$ is transitive.

The abstraction relation implies the existence of corresponding runs:

**Lemma 10.** For transition systems $T$ and $T'$ such that $T'$ abstracts $T$, for every run $\theta$ of $T$ there is a run $\theta'$ of $T'$ such that for all $i \in \{1, \ldots, r\}$, for all $t \in \mathbb{N}$, $\theta(t) \in Q_i$ implies $\theta'(t) \in Q'_i$.

The proof is easy by element-wise application of the abstraction function. Moreover, due to Theorem 5.6. in Clarke et. al. [12] we have:
Lemma 11. For every transition system $T$ and $T'$, for every LTL formula $\varphi$, if $T \preceq T'$ then $T' \models \varphi$ implies $T \models \varphi$.

So, for showing satisfiability we will try to construct transition systems that abstract the original system (i.e., an over-approximation), and for falsification transition systems that are abstracted by the original system (i.e., an under-approximation).

3.1. **Over-approximating abstraction**

We use predicate abstraction, tuned to our application domain of hybrid systems. In this framework, the abstract state space is given by a finite set of first-order predicates $P$, which jointly cover the concrete state space, that is for all $\sigma$ in $\Sigma$ there is a $p \in P$ such that $\sigma \in \llbracket p \rrbracket$ (note that in contrast to some approaches in software model checking, here the abstract states are formed by the predicates themselves and not Boolean combinations of them).

Different approaches for finding $P$ have been discussed in the literature. For example, an initial set of predicates can be derived from transition guards and atomic formulas in the specification logic [7]; or a certain class of predicates, such as convex polyhedra [10], or hyper-rectangles [29] can be used.

For a given finite set of predicates $P$, we construct an abstraction $\overline{a}_P(S)$ (natural abstraction) of $[S]$. It is a transition system whose transition relation is the set of all $(p, p')$ for which there is a pair $(\sigma, \sigma') \in \llbracket \tau \rrbracket$ such that $\sigma \models p$ and $\sigma' \models p'$. The set of initial states, and the observed propositions are defined canonically as the set of all $p$ for which there is a $\sigma \in \llbracket \pi \rrbracket$ such that $\sigma \models p$. We get an abstraction function between the concrete infinite state transition system $[S]$ and $\overline{a}_P(S)$ by assigning to each state space element $\sigma$ of $S$ a predicate $p$ such that $\sigma \models p$. Due to Lemma 11 for all first-order LTL formulas $\varphi$, $\overline{a}_P(S) \models \varphi$ implies $S \models \varphi$.

Note that here the abstract transition relation also might contain self-loops, that is, transitions from a predicate to itself, if the transition relation $\tau$ specifies transitions between between two elements satisfying the same abstract state. If all trajectories eventually leave a certain region, then the corresponding self-loops will be removed as soon as the abstraction is fine enough. This allows the method to prove progress properties.

We now introduce the notion of the diameter of a predicate abstraction to later measure the degree of imprecision introduced by an abstraction.

**Definition 12.** The diameter $\text{diam}(p)$ of a predicate $p \in P$ is defined as the supremum of $\{d(\sigma, \sigma^*) \mid \sigma \in \llbracket p \rrbracket, \sigma^* \in \llbracket p \rrbracket\}$. The diameter $\text{diam}(P)$ of a predicate abstraction over $P$ is defined as the maximal diameter of a predicate in $P$.

To bound the degree of imprecision of abstraction we will ensure that for every $\varepsilon > 0$ the abstraction eventually only represents a $\varepsilon$-perturbation of $S$. Hence, the query will eventually be proven on the abstraction. Since it is hard to compare the discrete and finite abstraction with the transition system denoted by $S$, we will
measure these perturbations not from the abstraction \( \overline{\alpha}_P(S) \) directly, but from the following continuous over-approximation:

**Definition 13.** A transition system \((\rightarrow, Q_0, Q_1, \ldots, Q_r)\) over a set of predicates \(P\) represents the transition system

\[
\gamma(\rightarrow, Q_0, \ldots, Q_r) = (\gamma(\rightarrow), \gamma(Q_0), \gamma(Q_1), \ldots, \gamma(Q_r)),
\]

where \(\gamma(R) = \bigcup_{p \in R}[p]\).

It is not hard to prove that, using an abstraction function that assigns to each \(p \in P\), an element of \([p]\), for every transition system \(T\) over \(P\), \(\gamma(T)\) abstracts \(T\). Hence any query \(\varphi\) that is satisfied by \(\gamma(T)\) is also satisfied by \(T\), and in particular \(\gamma(\overline{\alpha}_P(S)) \models \varphi\) implies that model checking the abstraction will succeed, that is \(\overline{\alpha}_P(S) \models \varphi\).

So we are left with the task of showing that \(\gamma(\overline{\alpha}_P(S))\) will be sufficiently close to \([S]\). For this, given a constraint \(\phi\), let \([\phi]_{\varepsilon}\) be the set of all \(x\) for which there is a \(\phi^*\) with \(d(\phi, \phi^*) \leq \varepsilon\) and an \(x^*\) with \(d(x, x^*) \leq \varepsilon\) such that \(x^* \models \phi^*\). Clearly this forms the maximal element of the \(\varepsilon\)-perturbed solution sets of a constraint \(\phi\) wrt. the partial order \(\subseteq\). Extending this to every constraint defining a hybrid system, we denote the transition system given by the resulting maximal elements by \([S]_{\varepsilon}\), and we have:

**Theorem 14.** \([S]_{diam(P)}\) abstracts \(\gamma(\overline{\alpha}_P(S))\).

We do not include the proof since it can be adapted from the proof of Theorem 17 below. We can conclude that an abstraction \(\overline{\alpha}_P(S)\) only introduces bounded perturbations since it can be sandwiched between the exact system \([S]\) and its perturbed version \([S]_{diam(P)}\) due to the fact \([S] \preceq \overline{\alpha}_P(S) \preceq \gamma(\overline{\alpha}_P(S)) \preceq [S]_{diam(P)}\).

The natural abstraction can be constructed effectively, if we do not allow the transcendental function symbols \(\sin, \cos, \exp\) in our constraints. For this we decide [33] satisfiability of \(p(x_1, \ldots, x_k) \land \tau(x_1, \ldots, x, x'_1, \ldots, x'_k) \land p(x'_1, \ldots, x'_k)\) for defining the abstract transition relation, respectively \(p \land \pi_i\) for determining the set of initial states and observed propositions. However, due to the huge complexity of the corresponding decision procedure [9], this approach is not viable in practice.

Consider thus a predicate abstraction of \(S\), where each predicate is of the form \(m = m \land B\), where \(B\) is a so-called box of the form \(\bigwedge_{i \in \{1, \ldots, k\}} c_{i, l} \leq x_i \leq c_{i, u}\). We will usually write such box predicates as pairs \(\langle m, B \rangle\).

In this case the computational effort in constructing the abstract transition relation can be drastically reduced by using interval arithmetic based tests instead of full decision procedures (the cost of a single test reduces from non-elementary in the number of variables to linear in the expression size). Moreover, this does not restrict the allowed function symbols to addition and multiplication. In this approach, transitions from box \(p\) to box \(p'\) are only added, if they cannot be excluded by interval arithmetic. We thus further abstract from the concrete transition behavior.
More specifically, we evaluate terms over boxes by extending all function symbols $f$ to corresponding functions $f^I$ over intervals. For example, the arithmetic expression $xy + 1$ for a box that restricts $x$ to $[-1, 1]$, and $y$ to $[1, 2]$, evaluates to $[-1, 1][1, 2] = [-2, 2]$ and $[1, 1] = [-1, 3]$. Given an arithmetic expression $e$ and a box $B$ we denote by $I(e)(B)$ the interval evaluation of $e$ over $B$.

The properties of interval evaluation of terms have been widely studied [24, 21]. Here we use a version that is extended to constraints. Using the Booleans $\{\mathbf{F}, \mathbf{T}\}$ with the order $\mathbf{F} < \mathbf{T}$ one can form Boolean intervals, which allows us to extend relations and connectives to intervals in a similar way as above. Hence we can evaluate Boolean combinations of equalities and inequalities over intervals. The formalization of this is a trivial exercise. For example, the evaluation of the constraint $2x \geq 0 \lor x - 2 \geq 0$ over a box restricting $x$ to $[1, 3]$ yields $[2, 2][1, 3] = [0, 0] \lor [1, 3] - [2, 2] = [2, 6] \geq [0, 0] \lor [-1, 1] = [0, 0] = [\mathbf{T}, \mathbf{T}] \lor [\mathbf{F}, \mathbf{T}] = [\mathbf{T}, \mathbf{T}]$. One can easily incorporate mode constraints by evaluating a constraint of the form $\mathit{mode} = m_0$ over a box predicate $\langle m, B \rangle$ to $[\mathbf{T}, \mathbf{T}]$ iff $m = m_0$ and to $[\mathbf{F}, \mathbf{F}]$, otherwise.

Whenever such an evaluation yields an interval $[\mathbf{F}, \mathbf{F}]$ we know that the corresponding constraint cannot hold. So we get a conservatively over-approximated satisfaction relation $\models_I$ such that $\langle m, B \rangle \models_I \phi$ iff $\mathbf{T}$ is in the interval evaluation of $\phi$ on $\langle m, B \rangle$. Hence $\langle m, B \rangle \not\models_I \phi$ tells us that $\phi$ cannot be satisfied by mode $m$ and an element of $B$, whereas $\langle m, B \rangle \models_I \phi$ does not tell us anything since in the case where interval evaluation is $[\mathbf{F}, \mathbf{T}]$, the element $\mathbf{T}$ might be spurious due to over-approximation.

Now, by using the over-approximated satisfiability $\models_I$ we get another abstraction $\overline{\alpha}_P(S)$ (the interval abstraction) for a given set of box predicates $P$. Since $\models_I$ over-approximates $\models$, also $\overline{\alpha}_P(S) \supseteq \overline{\alpha}_P(S)$, and hence $\overline{\alpha}_P(S) \supseteq \overline{\alpha}_P(S)$. However, we again have to establish that this abstraction only introduces bounded over-approximation:

We start with providing bounds for interval evaluation. By its Lipschitz continuity (e.g., Theorem 2.1.1 in Neumaier’s book [24]), it is easy to derive the following convergence result for interval evaluation of terms:

**Lemma 15.** For every arithmetic expression $e$ with function symbols in the set $\{+,*,-,\exp,\sin,\cos\}$, denoting a function $[e]$ and box $B$ there is a function $E: \mathbb{R}^+ \rightarrow \mathbb{R}^+$ such that $\lim_{x \rightarrow 0} E(x) = 0$, and for every box $B'$ with $[B'] \subseteq [B]$, for all $y \in I(e)(B')$, there is an $x \in [B']$ such that $d([e](x), y) \leq E(\text{diam}(B'))$.

Now we can bound the approximation of interval satisfaction on constraints:

**Lemma 16.** For every constraint $\phi$, mode $m$ and box $B$ there is a function $E: \mathbb{R}^+ \rightarrow \mathbb{R}^+$ with $\lim_{x \rightarrow 0} E(x) = 0$, such that for every box $B'$ with $[B'] \subseteq [B]$, $\langle m, B' \rangle \models_I \phi$ implies that there is a $\phi^*$ with $d(\phi, \phi^*) \leq E(\text{diam}(B'))$ and an $x \in [B']$ such that $\langle m, x \rangle \models \phi^*$. 
Proof. Let $\phi$, $m$, and $B$ be arbitrary but fixed. Let us first assume that $\phi$ is an atomic arithmetic constraint of the form $e \geq c$. Choose $E$ as provided by Lemma 15, let $B'$ be arbitrary, but fixed, and assume $B' \models I e \geq c$. In the case when $I(\phi)(B') = \{T\}$, the rest is trivial. In the case when $I(\phi)(B') = \{F, T\}$, $c \in I(e)(B')$, and we can choose $y = c$ in Lemma 15, which provides a corresponding $x \in [B']$ such that $d([e](x), c) \leq E(diam(B'))$. This implies $[e](x) \geq c - E(diam(B'))$. Choosing $\phi^*$ as $e \geq c - E(diam(B'))$ clearly $d(\phi, \phi^*) \leq E(diam(B'))$ and $\langle m, x \rangle \models \phi^*$.

The case of other atomic constraints with different relation symbols are similar, and the case of mode constraints is trivial. In the case where $\phi$ is non-atomic we can choose $E$ as the maximum of the $E$’s of its atomic sub-constraints and choose $\phi^*$ by taking for each atomic constraint the corresponding constraint constructed above.

\[\square\]

Note that in practice—in order to ensure efficiency—interval arithmetic is usually implemented using floating point arithmetic. In that case, all the necessary operations are rounded outwards. So, differently from other methods, we preserve correctness also under the presence of rounding. In principle, for achieving a tight enough over-approximation, one would need to adjust the precision of the used floating-point representation to the level of robustness of the given verification problem. However, experience has shown that for cases arising in practice the usual machine floats suffice.

Finally we can establish an analogous result to Theorem 14:

Theorem 17. There is a function $E : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ with $\lim_{x \rightarrow 0} E(x) = 0$, such that given a set of box predicates $P$, $[S]_{E(diam(P))}$ abstracts $\gamma(\alpha_P(S))$.

Proof. Let $\alpha_P(S)$ be of the form $(-, Q_1, \ldots, Q_r)$. Let $E_r$ be the function given by Lemma 16 for the transition constraint $\tau$ of $S$ and the box $I^{x_1} \times \ldots \times I^{x_k}$ bounding the state space, and $E_{s_1}, \ldots, E_{s_k}$ be the functions given by Lemma 16 for the state space constraints $\pi_1, \ldots, \pi_k$ of $S$ and the bound of the state space $I^{x_1} \times \ldots \times I^{x_k}$. Let $E(x) = \max\{x, E_r(x), E_{s_1}(x), \ldots, E_{s_k}(x)\}$. We prove that $[S]_{E(diam(P))} \supseteq \gamma(\alpha_P(S))$, with $\supseteq$ interpreted element-wise which implies the theorem.

- For an arbitrary, but fixed $i \in \{0, \ldots, r\}$, for proving that every element $\sigma$ of $\gamma(Q_i)$ is in the corresponding element of $[S]_{E(diam(P))}$, we prove that it is an element of an $E(diam(P))$-perturbed solution set of the corresponding state space constraint $\pi_i$. Observe that, by Definition 13, there is a corresponding state space element $p$ of $Q_i$ such that $\sigma \models p$. By definition of interval abstraction, $p \models \pi_i$. So, by Lemma 16, there is a $\pi^*_i$ with $d(\pi_i, \pi^*_i) \leq E_{s_i}(diam(P))$, and $\sigma^*$ with $\sigma^* \models \pi^*_i$. Since $diam(p) \leq diam(P)$, also $d(\sigma, \sigma^*) \leq diam(P)$. So, by Definition 5, every element $\sigma$ of $\gamma(\pi_i)$ satisfies the transition constraint up to $E(diam(P))$.

- For proving that every element $\langle \sigma, \sigma' \rangle$ of $\gamma(-, -)$ is in the corresponding element of $[S]_{E(diam(P))}$, we have to prove that it is an element of an
\(E(diam(P))\)-perturbed solution set of the transition constraint \(\tau\). Observe that by Definition 13 there is a corresponding transition \(\langle p, p' \rangle\) in \(\to\) such that \(\sigma \models p\), and \(\sigma' \models p'\). By definition of interval abstraction, \(\langle p, p' \rangle \models_\tau \tau^*\). So, by Lemma 16, there is a constraint \(\tau^*\) with \(d(\tau, \tau^*) \leq E_\tau(diam(P))\), and \(\langle \sigma^*, \sigma'' \rangle\) with \(\sigma^* \models \tau^*\). Since \(diam(p) \leq diam(P)\) and \(diam(p') \leq diam(P)\), also \(d(\sigma, \sigma^*) \leq diam(P)\) and \(d(\sigma', \sigma'') \leq diam(P)\). So, by Definition 5, every element \(\langle \sigma, \sigma' \rangle\) of \(\gamma(\to)\) satisfies the transition constraint up to \(E(diam(P))\). □

So also the abstraction \(\alpha_p(S)\) only introduces bounded perturbations since it can be sandwiched between the exact system \([S]\) and its maximally \(E(diam(P))\)-perturbed version \([S]_{E(diam(P))}\) due to the result \([S] \succeq [\alpha_p(S)] \succeq [\gamma(\alpha_p(S))] \succeq [S]_{E(diam(P))}\). By decreasing the diameter of \(P\), the precision of the abstraction can be arbitrarily increased. We will use these results in the development of an algorithm for proving robust satisfaction of LTL formulas of discrete time hybrid systems.

3.2. Under-approximating abstraction

Now we also construct a finite system that under-approximates the original system \(S\), hence \([S]\) now abstracts the under-approximation. We first present a simple and succinct approach, which helps to simplify the proof, and then point out possible improvements to achieve practical efficiency.

We choose a sample point \(s(p)\) for every predicate \(p \in P\). Then let \(\alpha_p(S)\) be the transition system whose transition relation is the set of all \(\langle s(p), s(p') \rangle\) such that \(p, p' \in P\), and \(\langle s(p), s(p') \rangle \models \tau\), and for which for every \(i \in \{0, \ldots, k\}\), the \(i\)-th observed proposition contains the set of all \(s(p)\) such that \(p \in P\), \(s(p) \models \pi_i\). In the case without transcendental function symbols this can be effectively constructed, and due to an identity abstraction function \(\alpha_p(S)\) is abstracted by \([S]\), which ensures the correctness of the under-approximation.

Now let, for a constraint \(\phi\), \([\phi]\), be the set of all \(x\) for which there is no \(\phi^*\) with \(d(\phi, \phi^*) \leq \varepsilon\) and no \(x^*\) with \(d(x, x^*) \leq \varepsilon\) such that \(x^* \not\models \phi^*\), which is the minimal element of the \(\varepsilon\)-perturbed solution sets of \(\phi\) with the subset relation \(\subseteq\). Again we extend this to hybrid systems, and use the result to bound the under-approximation error as follows:

**Theorem 18.** \([S]_{diam(P)} \succeq \alpha_p(S)\)

We do not include a proof here since it can be adapted from the proof of Theorem 19 below.

So, instead of falsifying an LTL formula against the original system \(S\) we can check it against \(\alpha_p(S)\). Moreover, by letting the diameter of \(P\) go to zero, this check will eventually succeed for robust systems.
However, there is no known algorithm that can check in all cases whether a term containing constants, addition, multiplication, and transcendental function symbols is zero. The reason is that it is not known how many digits after the comma have to be zero to be able to decide that the constant is zero. Hence, in that case, it also not in general possible to check the transition relation and observed propositions against the sample points.

In our solution we use interval arithmetic to enclose the value of the terms of the constraints at the sample points into small intervals of bounded width (or alternatively, compute their value up to a certain precision), and compute a conservative under-approximation based on that information. Again, this ensures correctness of the under-approximation ($\alpha_{P}^{I}(S) \leq [S]$). Moreover, since for robustly false problems finite precision suffices for falsification, by making the width of the computed intervals go to zero as the diameter of the abstraction goes to zero, the falsification will eventually succeed:

**Theorem 19.** There is a function $E : \mathbb{R}^{+} \rightarrow \mathbb{R}^{+}$ with $\lim_{x \to 0} E(x) = 0$, such that given a set of predicates $P$, $[S]_{E(diam(P))} \leq \alpha_{P}^{I}(S)$

**Proof.** Let $E_{a}$ be the function that assigns to $\varepsilon$ the maximal width of the intervals enclosing the terms used for computing $\alpha_{P}^{I}(S)$. Choose for $E$ the function such that $E(\varepsilon) = \max\{\varepsilon, E_{a}(\varepsilon)\}$. Assume that $[S]_{E(diam(P))}$ has the form $(\neg, Q_{0}, Q_{1}, \ldots, Q_{r})$, and $\alpha_{P}^{I}(S)$ has the form $(\neg', Q_{0}', Q_{1}', \ldots, Q_{r}')$.

For proving $[S]_{E(diam(P))} \leq \alpha_{P}^{I}(S)$, we use an abstraction function $H$ such that $H(x) = s(p)$ where $p$ is a predicate in $P$ such that $x \models p$. We prove the two items of Definition 9 as follows:

- Let $i \in \{0, \ldots, r\}$ be arbitrary, but fixed. Let $x \in Q_{i}$. We have to prove that $H(x)$ is in the corresponding element $Q_{i}'$ of $\alpha_{P}^{I}(S)$. By definition of $H$ this means to prove that for all $p$ with $p \in P$ and $x \models p$, $s(p) \in Q_{i}'$, that is, interval evaluation of $\pi_{i}$ on $s(p)$, as used in the construction of $\alpha_{P}^{I}(S)$, yields a true value.

  Since the distance between $x$ and $s(p)$ is smaller than $diam(P)$ and hence also smaller than $E(diam(P))$, and due to the definition of $[S]_{E(diam(P))}$ we know that for all constraints $\pi_{i}$ with $d(\pi_{i}, \pi_{i}') \leq E(diam(P))$, and hence also for all constraints $\pi_{i}'$ with $d(\pi_{i}, \pi_{i}') \leq E(diam(P))$, $s(p) \models \pi_{i}'$. Hence interval evaluation of $\pi_{i}$ will be precise enough to ensure that $s(p) \in Q_{i}'$.

- Let $x, x_{1}$ be such that $x \rightarrow x_{1}$. We have to prove that $H(x') \rightarrow' H(x_{1})'$. By definition of $H$ this means to prove that for all $p$ and $p_{1}$ with $p, p_{1} \in P$, $x \models p$, and $x_{1} \models p_{1}$, we have that $s(p) \rightarrow' s(p_{1})$, that is interval evaluation of $\tau$ on $(s(p), s(p_{1}))$, as used in the construction of $\alpha_{P}^{I}(S)$, yields a true value.
Since the distance between $x$ and $s(p)$ and the distance between $x_1$ and $s(p_1)$ are both smaller than $diam(P)$, and hence also smaller than $E(diam(P))$, and due to the definition of $[S]_{E(diam(P))}$ we know that for all constraints $\tau^*$ with $d(\tau, \tau^*) \leq E(diam(P))$, and hence also for all constraints $\tau^*$ with $d(\tau, \tau^*) \leq diam(P)$, $\langle s(p), s(p_1) \rangle \models \tau^*$. Hence interval evaluation of $\tau$ will be precise enough to ensure that $s(p) \rightarrow^* s(p_1)$. $\square$

This approach will typically need a set of predicates with a very small diameter to be able to guarantee the existence of transitions. We now sketch some improvements, based on numerical local search methods and argue, that for realistic designs this will lead to drastic improvements.

We exploit typical testing strategies for dynamic systems. Consider as an example electronic stability control applications. Test engineers would typically force a car into areas where the different modes of such an envelope protection system become active, testing each mode in isolation. For example, they would from a stable driving situation speed into a curve in a way causing oversteering without control intervention, and after recovery of stability proceed to a next maneuver, for example, testing the ability to control partial road-icing. Such examples teach two lessons:

- Test-cases require only a bounded history - each threat of violating envelop protection, as well as the timely invocation of control-strategies compensating for the attack on car stability, can be tested in isolation, starting from a "home state", in which the car is stable.
- Secondly, the test driver is forcing the open control inputs (e.g., angle of steering wheel, rate of acceleration) towards meeting the triggering condition for the "relevant" mode of the envelope protection algorithm. Heuristics for efficient construction of under-approximations should thus find settings of open control inputs moving the plant from a stable "home state" towards predicates guarding transitions towards recovery modes of the system.

This motivates an approach that analyzes sequences of predicates $p_0, \ldots, p_t$ whose corresponding transitions have not yet been excluded in an over-approximating abstraction. It then uses using an iterative numerical method with a sample point in each predicate as starting point to find a concrete trajectory running through this sequence. By recursively analyzing longer and longer sequences the method will build longer and longer concrete trajectories. The above analysis shows that this will eventually lead to trajectories from home state to home state, allowing us to build transitions of the underapproximation for which we know that they can be concatenated, and hence be used to form concrete counter-examples.

Since the approach uses sample points as starting points for the numerical search method, it inherits the convergence properties ensured by the theorems above. And since we know from the above analysis that usually an analysis of sequences of short
3.3. Proving robust satisfaction and falsification

Now assume as given a temporal specification $\varphi \in LTL$, with the arithmetic atoms $\Pi = \{\pi_0, \ldots, \pi_n\}$ occurring negatively. Since the aim of the current paper is to establish the overall approach, we only give a basic algorithm for abstraction refinement, whose efficiency can be significantly improved according to the directions outlined below. The key result of this section is, that the abstraction refinement algorithm is guaranteed to terminate, if $\varphi$ is robustly satisfied or falsified by $S$.

We introduce the following basic algorithm: Create a sequence of partitions $P_0, P_1, \ldots$ such that the diameter of the partitions goes to zero. If at a certain iteration $i$, $\alpha_{P_i}^f(S)$ falsifies $\varphi$, terminate with the result that $\varphi$ is robustly falsified in $S$. If $\alpha_{P_i}^f(S)$ satisfies $\varphi$, terminate with the result that $\varphi$ is robustly satisfied by $S$. Here we start with $P_0 = \{(m, I^2 \times \ldots \times I^n) \mid m \in M\}$ as the initial partition. We ensure that the diameter of the partition goes to zero by splitting the largest box in $P_m$ along the biggest side-length to obtain $P_{m+1}$ (recall that the real-valued variables range over bounded intervals).

Before providing our main result and applying the algorithm to a more complex example, we illustrate its behavior on a toy example of a safety verification problem. To keep the example simple, we concentrate on its continuous behavior and completely ignore possible discrete behavior. We assume that the constraint $\tau$ is of the form $x' = x_1/2 \land x_2 = x_2/2$, that the set of initial states is described by a constraint $\pi_0$ of the form $x_1 \leq 3 \land x_2 \leq 3$ and the set of unsafe states is described by a constraint $\pi_1$ of the form $x_1 \geq 6$. The assumption of a state space $[0,8] \times [0,10]$ will lead to the initial partition $\{[0,8] \times [0,10]\}$. The corresponding abstraction consists of the single abstract state $[0,8] \times [0,10]$ with the set of initial states and set of unsafe states both being equal to $[0,8] \times [0,10]$ (see Figure 2, where the left-hand side shows the partitioning of the state space, and the right-hand the corresponding abstraction). This abstraction is clearly unsafe, and hence a finer partition is needed.

The algorithm will then split $[0,8] \times [0,10]$ along the biggest side-length which will result in an abstraction consisting of the two abstract states $[0,8] \times [0,5]$, $[0,8] \times [5,10]$. This abstraction is still not safe since $[0,8] \times [0,5]$ is still both in the set of initial and the set of unsafe states (see Figure 3).

After two further splittings, the abstraction consisting of the four abstract states $[0,4] \times [0,5]$, $[0,4] \times [5,10]$, $[4,8] \times [0,5]$, and $[4,8] \times [5,10]$ (see Figure 4) will be safe, proving the safety of the original system.

The following main result opens a new line of attack to the verification of nonlinear hybrid systems.

**Theorem 20.** The basic algorithm is guaranteed to terminate with definite answer if $S$ robustly satisfies $\varphi$ or $\varphi$ is robustly falsified by $S$. 
Proof. The abstraction refinement procedure ensures that the diameter of the abstraction goes to zero. If $S$ robustly satisfies $\varphi$, the fact that $\gamma(\tilde{\alpha}_{\mathcal{P}_m}(S))$ abstracts $\tilde{\alpha}_{\mathcal{P}_m}(S)$ due to Theorem 17, and transitivity of abstraction implies that $\tilde{\alpha}_{\mathcal{P}_m}(S)$ is abstracted by $[S]_{\varepsilon}$ with $\varepsilon$ going to zero as $m$ goes to infinity. Let $\tau > 0$ be such
that $\overline{[S]} \models \varphi$ which is ensured by robustness. Thus, there is an $m$, from which on $\varepsilon$ will be smaller than $r$. Then $\overline{\sigma}_{P_m}(S) \models \varphi$ and the algorithm succeeds. The case where $\varphi$ is robustly falsified in $S$ is similar. 

Note that this theorem also includes robust progress properties. In that case, the algorithm will eventually remove all unnecessary transitions in the abstraction that lead from a predicate to itself.

Clearly, further work is needed, to make this algorithm practically efficient. Still—in order to evaluate its efficiency potential—we have implemented it using the programming language O’Camil. Our implementation keeps a list of boxes for each mode, starting with one box per mode. It computes the abstraction by going through each box, and marking it as initial or unsafe if interval arithmetic cannot disprove the corresponding constraint. Moreover, on every pair consisting of this box and another box, it checks whether interval arithmetic can disprove the transition constraint, and if it cannot, it puts a transition in the abstraction. If this abstraction is safe, verification is finished. If it is not, the box with the biggest side-length is split, and the abstraction is recomputed. This continues until a safe abstraction can be found, which is ensured for robust inputs due to the results of this paper.

For the interval arithmetic checks, the implementation uses the constraint propagation engine RSolver [27]. In some cases, this allows us, in a similar way as in the continuous time case [29, 28], to prove that only a part of a given box can be reachable from any other box, in which case we only keep this smaller part in the box list. Moreover, after splitting a box, we do not recompute the whole abstraction, but only the part that was involved in the split.

Already in this basic form, the algorithm yields promising results for realistic examples. More specifically, for our collision-avoidance example, a designer will ensure that the necessary distance of the two planes should not only be kept for this very example, but this should also be the case when the behavior of the planes changes slightly. The results of this paper imply, that in such a robust case our verification algorithm terminates.

In fact, for $\delta = 0.6$, our interval arithmetic prototype proves in about 20 seconds that the safety margin of the planes is maintained, using an abstraction that has only 100 boxes.

4. Conclusion

This paper opens a novel line of attack to the verification of non-linear hybrid systems. We have argued for the naturalness of the notion of robust satisfaction, and demonstrated how to construct a series of increasingly more accurate abstractions, which for robust designs is guaranteed to converge to a sufficiently precise model to prove or falsify temporal specifications of hybrid systems in a rich specification logic with first-order arithmetic constraints, able to express real-time requirements. Though we have chosen LTL as the temporal framework in this paper, the devel-
opment only exploits safeness of the constructed abstractions; it is well known [13], that also ACTL* properties are preserved under the performed abstractions.

The base algorithm is compatible with many of the optimization techniques for abstraction refinement. Promising directions for optimization currently under investigation in the large scale collaborative research project AVACS include:

*Initial Partitioning.* We refine $B_0$ to approximately discriminate all guards and arithmetic constraints in $\Pi$, over-approximating their shapes by boxes. This approach is already realized as part of another research activity for verification of hybrid systems based on predicate abstraction techniques [7].

*Counterexample guided abstraction refinement:* We incrementally analyze counterexample fragments for concretization [11]. We do so, by applying the constraint propagation based solver for non-linear constraints [27] to the corresponding first-order formula. If the constraint is unsolvable, we dismiss the counterexample fragment as spurious by encoding the corresponding information into an automaton-based representation of the abstraction.

*Local search for counter-examples:* Instead of just testing samples in the abstract states for counter-examples, we use local search (based on a Newton-like method) to find samples that form counter-examples.

We see this paper hence as a promising starting point in exploiting the usage of interval-based constraint solving techniques for the verification of non-linear hybrid systems.

**Acknowledgements:** The authors are grateful to the anonymous reviewers of the paper for their helpful comments.

**References**


Chapter 7

Safety Verification for Probabilistic Hybrid Systems

This chapter appeared as:

Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Moritz Hahn. “Safety Verification for Probabilistic Hybrid Systems”. In: European Journal of Control 18.6 (2012). 572–587
Safety Verification for Probabilistic Hybrid Systems

Lijun Zhang\textsuperscript{1,\ast\ast}, Zhikun She\textsuperscript{2,***}, Stefan Ratschan\textsuperscript{3,****}, Holger Hermanns\textsuperscript{4,*****}, Ernst Moritz Hahn\textsuperscript{4,\ast}\textsuperscript{*}

\textsuperscript{1} DTU Informatics, Technical University of Denmark, Denmark; \textsuperscript{2} LMIB and School of Mathematics and Systems Science, Beihang University, China; \textsuperscript{3} Institute of Computer Science, Czech Academy of Sciences, Czech Republic; \textsuperscript{4} Department of Computer Science, Saarland University, Germany

\textbf{Keywords:} Hybrid system, probabilistic, abstraction refinement, safety.

1. Introduction

Hybrid systems constitute a general and widely applicable class of dynamical systems with both discrete and continuous components. Conventional hybrid system formalisms [3, 13, 27, 30] capture many characteristics of real systems. However, in many modern application areas, also probabilistic dynamics occur. This is especially true for wireless sensing and control applications, where message loss probabilities and other random effects (node placement, node failure, battery drain) turn the overall control problem into a problem that can only be managed with a certain, hopefully sufficiently large, probability: due to the influence of these random effects, the success (for example keeping the hybrid system in a safe region) can only be guaranteed with a probability less than one.

The need to integrate probabilities into hybrid systems formalisms has led to several different notions of stochastic hybrid systems, each from a distinct perspective [1, 2, 9, 11, 32]. The most important distinction lies in the point of attack where to introduce randomness. One option is to replace deterministic jumps by probability distributions over deterministic jumps. Another option is...
to generalise the differential equation components inside
a mode by a stochastic differential equations component.
More general models can be obtained by blending the
above two choices, and by combining them with mem-
oryless timed probabilistic jumps [8], and with nondeter-
minism. One prominent example of such a blend is the
model of piecewise-deterministic Markov processes [15],
a deterministic hybrid system model augmented with
memoryless timed probabilistic jumps.

An important problem in hybrid systems theory is that of
reachability analysis. In general terms, a reachability anal-
ysis problem consists in evaluating whether a given system
may reach certain unsafe states, starting from certain initial
states. This problem is associated with the safety verifica-
tion problem: to prove that the system can never reach any
unsafe state. In the probabilistic setting, the safety ver-
ification problem can be formulated as that of checking
whether the probability that the system trajectories reach
an unsafe state from an initial state can be bounded by
some given probability threshold.

In this paper, we focus on the model of probabilistic
hybrid automata [32], an extension of hybrid automata
where jumps involve probability distributions. This adds
the possibility to represent model-component failures,
message losses, buffer overflows and the like. Since these
phenomena are important aspects when aiming at faith-
ful models for networked and embedded applications, the
interest in this formalism is growing [16, 33]. We are striv-
ing for computational tools to support the analysis of the
models of Sproston [32] and more complicated models,
based on mathematically sound foundations.

Up to now, foundational results on the probabilistic
reachability problem for probabilistic hybrid automata
are scarce. Since they form a strict superclass of hybrid
automata, this is not surprising. Decidability results are
available for probabilistic linear hybrid automata and
probabilistic o-minimal hybrid automata [32].

This paper explains how we can harvest and combine
recent advances in the hybrid automata and the probabilis-
tic automata worlds, in order to treat the general case. We
are doing so by computing safe overapproximations via
abstractions in the continuous as well as the probabilis-
tic domain. One of the core challenges then is how to
construct a sound probabilistic abstraction over a given
covering (i.e., set of abstract states, each of which is
formed by a subset of the concrete state space, such that
their union equals the complete concrete state space) of
the state space. For this purpose, we first consider the
non-probabilistic hybrid automaton obtained by replac-
ing probabilistic branching with nondeterministic choices.
Provided that we have obtained a finite abstraction for this
classical hybrid automaton, we then decorate it with proba-
bilities to obtain a probabilistic abstraction, namely a finite
probabilistic automaton [31]. We show the soundness of
the approach, which allows us to verify probabilistic safety
properties on the abstraction: if such a property holds
in the abstraction, it holds also in the concrete system.
Otherwise, refinement is required to obtain a more precise
result.

Our abstraction approach can be considered as an
orthogonal combination of definitions of abstraction for
hybrid automata [4, 30], and for Markov decision pro-
cesses [14, 19]. Because of this orthogonality, abstrac-
tions of probabilistic hybrid automata can be computed
via abstractions for non-probabilistic hybrid automata
and Markov decision processes. To show the applica-
bility of this combination, we implemented a proto-
type tool, ProHVer, that builds an abstraction via a
combination of existing techniques for classical hybrid
automata [17] as well as methods for Markov decision
processes [14, 19, 25]. Subsequently, a fixed-point engine
computes the reachability probabilities on the abstraction,
which provides a safe upper bound on the reachability
probability in the model semantics. If needed, iterative
refinement of the hybrid abstraction is performed. We
report several successful applications of this prototypical
implementation on different case studies. To the best of
our knowledge, this is the first implementation that auto-
matically checks safety properties for probabilistic hybrid
automata.

The framework considered here has the advantage of
orthogonality: if a non-probabilistic abstraction is used
that differs from the one we employ in this paper, then
this abstraction can be extended with probabilities in a
very similar fashion. Furthermore, future computational
advances in hybrid automata analysis can directly be
harvested for the model of probabilistic hybrid automata.

Organisation of the paper. We first discuss related work
in Section 2, and then recall the definitions we use in
Section 3. In Section 4, we introduce the notion of abstrac-
tions for probabilistic hybrid automata, and discuss how
to compute the abstractions for safety verification prob-
lems. We illustrate our approach by applying it on a small
model in Section 5. In Section 6, we describe an imple-
mentation of our algorithm and apply it on several case
studies. Section 7 concludes the paper.

2. Related Work

The model considered in this paper extends (nondeter-
mistic) hybrid automata with probabilistic discrete jumps.
Davis [15] introduced piecewise-deterministic Markov
processes, whose state changes are triggered sponta-
nously as in continuous-time Markov chains. Apart
from forced and spontaneous probabilistic jumps, general
stochastic hybrid systems [10, 20] comprise stochastic
differential equations [5]. They can incorporate random perturbations, such as Brownian motion, into the continuous dynamics. While these models—with and without nondeterminism—enjoy a very rich variety of applications, their analysis is limited and often based on Monte-Carlo simulations [1, 11, 12, 22] of systems without nondeterminism. Another approximate analysis technique, restricted to systems without nondeterminism, relies on the use of stochastic simulation functions [23].

From the hybrid automata perspective, the general verification problem for safety properties is known to be undecidable. Certain classes (e.g., initialised rectangular automata [18], o-minimal hybrid automata [26]) are decidable, and there are algorithms that construct finite bisimulation quotients for them. These methods have been lifted to probabilistic hybrid automata by Sproston [32], and this can be used to compute exact results, rooted in a bisimulation-based abstraction. In these special cases, we can parametrise our principal technique in such a way that it yields the very same exact results. In practice however, our method gives us the liberty to use an abstraction that overapproximates the behaviour, and is tailored to the problem at hand. The computational results will then not be exact, but overapproximating. This relies on the use of nondeterminism as a powerful abstraction mechanism, and extends to undecidable classes as well. Indeed, we treat the general case using the fact that a practical verification can—to a certain extent—circumvent the decidability barrier by a heuristic algorithm: we exploit tools which can, in practice, verify hybrid automata belonging to undecidable classes, to verify corresponding probabilistic hybrid automata (cf., also the notion of quasi-decidability [29]). While Sproston focuses on decidability results for particular classes of probabilistic hybrid models, this paper considers safety verification for models in full generality. The paper is based on an earlier conference paper [34].

Abstraction approaches have also successfully been applied to probabilistic timed automata [24, 25], a class of probabilistic hybrid automata, where only derivatives of constant value 1 occur. Their abstract analysis is based on difference-bound matrices (DBMs), and does not extend to the general setting considered here. Fränzle et al. [16, 33] use stochastic SAT to solve reachability problems on probabilistic hybrid automata. Their analysis is limited to depth-bounded reachability properties, i.e., the probability of reaching a location within at most $N$ discrete jumps.

## 3. Preliminaries

In this section, we repeat the definition of conventional hybrid automata, in the style of [30], followed by the definition of probabilistic hybrid automata [32].

### 3.1. Hybrid Automata

#### 3.1.1. Syntax

We fix a variable $m$ ranging over a finite set of discrete modes $M = \{m_1, \ldots, m_n\}$ and variables $x_1, \ldots, x_k$ ranging over the real numbers $\mathbb{R}$. We denote by $S$ the resulting state space $M \times \mathbb{R}^k$. For denoting the derivatives of $x_1, \ldots, x_k$ we use variables $\dot{x}_1, \ldots, \dot{x}_k$, ranging over $\mathbb{R}$ correspondingly. The primed versions $m', x_1', \ldots, x_k'$ shall be used to describe the next state in a transition. For simplicity, we sometimes use the vector $\bar{x}$ to denote $(x_1, \ldots, x_k)$, and $(m, \bar{x})$ to denote a state. Similar notations are used for the primed and dotted versions $\bar{x}', \bar{x}$.

In order to describe hybrid automata, we use constraints that are arbitrary Boolean combinations of equalities and inequalities over terms. This way, constraints represent subsets of given sets. In case $v$ is contained in the set represented by a given constraint, we say that $v$ satisfies a given constraint. These constraints are used, on the one hand, to describe the possible flows and jumps and, on the other hand, to mark certain parts of the state space (e.g., the set of initial/unsafe states). A state-space constraint is a constraint over the variables $m, \bar{x}$, and represents a subset of $M \times \mathbb{R}^k$. A flow constraint is a constraint over the variables $m, \bar{x}, \dot{\bar{x}}$. It represents a subset of $M \times \mathbb{R}^k \times \mathbb{R}^k$. An example of a flow constraint for the case $n = 2$ and $k = 1$ is:

$$((m = m_1 \rightarrow \dot{x} = x) \land (m = m_2 \rightarrow \dot{x} = -x)).$$

In general, an invariant that has to hold in a mode can be modelled by formulating a flow constraint that does not allow a continuous behaviour in certain regions. As an example, the flow constraint $(m = m_1 \rightarrow 1 \leq x \leq 5)$ expresses that in mode $m_1$ the invariant $1 \leq x \leq 5$ holds.

For capturing the jump behaviours, we introduce the notion of update constraints. An update constraint $\omega$, also called a guarded command, has the form: $\text{condition} \rightarrow \text{update}$ where condition is a state-space constraint over $m, \bar{x}$, and update is an expression denoting a function $M \times \mathbb{R}^k \rightarrow M \times \mathbb{R}^k$ which is called the reset mapping for $m$ and $\bar{x}$. Intuitively, assume that the state $(m, \bar{x})$ satisfies condition, then the mode $m$ and variable $\bar{x}$ are updated\(^1\) to the new state $\text{update}(m, \bar{x})$.

\(^1\) Our definition of jumps is deterministic, as in [4], i.e., if a jump is triggered for a state satisfying condition, the successor state is updated deterministically according to update. In [30], the jump is defined to be nondeterministic: if a state satisfies condition, a successor will be selected nondeterministically from a set of states. Our method can be easily extended to this. We restrict to deterministic jumps for simplicity of the presentation in this paper.
A jump constraint is a finite disjunction $\bigvee_{u \in U} u$ where $U$ is a set of guarded commands. The constraint $\bigvee_{u \in U} u$ can be represented by the set $U$ for simplicity.

**Definition 3.1:** A hybrid automaton is a tuple $\mathcal{H} = (\text{Flow}, U, \text{Init}, \text{Unsafe})$ consisting of a flow constraint $\text{Flow}$, a finite set of update constraints $U$, a state-space constraint $\text{Init}$, describing the set of initial states, and a state-space constraint $\text{Unsafe}$, describing the set of unsafe states.

A flow of length $l$ in a mode $m$ is a function $r : [0, l] \mapsto \mathbb{R}^k$ such that
- if $l > 0$, then $r$ is differentiable for all $t \in [0, l]$, and we require that $(m, r(t), \dot{r}(t))$ satisfies $\text{Flow}$, where $\dot{r}$ is the derivative of $r$, and
- if $l = 0$, we require that there exists a constant $c \in \mathbb{R}^k$ such that $(m, r(t), c)$ satisfies $\text{Flow}$.

Provided that the constraint of a jump is enabled, in systems modelled in a reasonable manner, it will usually be the case that the corresponding jump leads to a state in which a valid flow exists. This also restricts valid states of the hybrid automaton.

### 3.1.2. Transition System Semantics

The semantics of a hybrid automaton is a transition system with an uncountable set of states.

**Definition 3.2:** A transition system is a tuple $(S, T, S_{\text{Init}}, S_{\text{Unsafe}})$ where
- $S$ is the (possibly uncountable) set of states,
- $T \subseteq S \times S$ is the transition relation,
- $S_{\text{Init}} \subseteq S$ is the set of initial states and
- $S_{\text{Unsafe}} \subseteq S$ is the set of unsafe states.

The semantics of $\mathcal{H} = (\text{Flow}, U, \text{Init}, \text{Unsafe})$ is a transition system $T_{\mathcal{H}} = (S_{\mathcal{H}}, T_{\mathcal{H}}, S_{\text{Init}}, S_{\text{Unsafe}})$ with state set $S = S_{\mathcal{H}} \times \mathbb{R}^k$, set of initial states $S_{\text{Init}} = \{s \in S \mid s \text{ satisfies } \text{Init}\}$, and unsafe states $S_{\text{Unsafe}} = \{s \in S \mid s \text{ satisfies } \text{Unsafe}\}$. The transition set $T$ is defined as the union of two transition relations $T_C, T_D \subseteq S \times S$, where $T_C$ corresponds to transitions due to continuous flows defined by:
- $((\vec{x}, m, \vec{x}'), (m, \vec{x}')) \in T_C$, if there exists a flow $r$ of length $l$ in mode $m$ such that $r(0) = \vec{x}$ and $r(l) = \vec{x}'$; and
- $T_D$ corresponds to transitions due to discrete jumps.

The transition due to an update constraint $u : \text{condition} \rightarrow \text{update}$, denoted by $T_D(u)$, is defined by:
- $((m, \vec{x}), (m', \vec{x}')) \in T_D(u)$ if $(m, \vec{x})$ satisfies the guard condition and it holds that $(m', \vec{x}') = \text{update}(m, \vec{x})$.

Then, we define $T_D = \bigcup_{u \in U} T_D(u)$.

In the rest of the paper, if no confusion arises, we use $\text{Init}$ to denote both the constraint for the initial states and the set of initial states. Similarly, $\text{Unsafe}$ is used to denote both the constraint for the unsafe states and the set of unsafe states.

### 3.2. Probabilistic Automata

For defining the semantics of a probabilistic hybrid automaton, we recall first the notion of a probabilistic automaton [31]. It is an extension of a transition system with probabilistic branching.

We first introduce some notation. Let $S$ be a (possibly uncountable) set. A distribution over $S$ is a function $\mu : S \rightarrow [0, 1]$ such that (a) the support $\text{Supp}(\mu) = \{s \in S \mid \mu(s) > 0\}$ is finite, and (b) it is $\sum_{s \in S} \mu(s) = 1$. Let $\text{Distr}(S)$ denote the set of all distributions over $S$. For an arbitrary but fixed state $s \in S$, a Dirac distribution for $s$, denoted by $\text{Dirac}_s$, is a distribution over $S$ such that $\text{Dirac}_s(s) = 1$, that is, $\text{Supp}(\text{Dirac}_s) = \{s\}$. The Dirac distribution will be used to describe the continuous evolution of a probabilistic hybrid automaton.

**Definition 3.3:** A probabilistic automaton $M$ is a tuple $(S, \text{Steps}, \text{Init}, \text{Unsafe})$, where $\text{Steps} \subseteq S \times \text{Distr}(S)$, $\text{Init} \subseteq S$, and $\text{Unsafe} \subseteq S$. Here,
- $S$ denotes the set of states,
- $\text{Init}$ is the set of initial states,
- $\text{Unsafe}$ the set of unsafe states, and
- $\text{Steps}$ the transition relation.

We use $s \rightarrow \mu$ as a shorthand notation for $(s, \mu) \in \text{Steps}$, and call $\mu$ a successor distribution of $s$. Let $\text{Steps}(s)$ be the set $\{\mu \mid (s, \mu) \in \text{Steps}\}$. We assume that $\text{Steps}(s) \neq \emptyset$ for all $s \in S$.

A finite path of $M$ is finite sequence $\sigma = s_0\mu_0 s_1\mu_1 \ldots s_n$ such that $s_i \rightarrow \mu_i$ and $\mu_i(s_{i+1}) > 0$ for all possible $0 \leq i < n$. Infinite paths are defined by taking $n = \infty$. We denote by first($\sigma$) the first state $s_0$ of $\sigma$, by $\sigma[i]$ the $(i + 1)$-th state $s_i$, and, if $\sigma$ is finite, by last($\sigma$) the last state of $\sigma$. Let $\text{Path}$ be the set of all infinite paths and $\text{Path}^*$ the set of all finite paths.

The nondeterministic choices in $M$ are resolved by adversaries. Intuitively, an adversary of $M$ is a measurable map $A : \text{Path}^* \rightarrow \text{Distr}(\text{Steps})$ such that $A(\sigma)(s, \mu) > 0$ implies that $s = \text{last}(\sigma)$ and $s \rightarrow \mu$. If $A(\sigma)(s, \mu) > 0$, then the successor distribution $\mu$ should be selected from state $s$ with probability $A(\sigma)(s, \mu)$. Given an initial state $s$, an adversary $A$ induces a discrete-time Markov chain with state space $\text{Path}^*$ in an obvious way. We let $\text{Prob}^A$ denote the corresponding unique probability measure [28] over $\text{Path}$.
3.3. Probabilistic Hybrid Automata

Now we recall the definition of probabilistic hybrid automata, by equipping the discrete jumps with probabilities. This is needed to model, for example, component failure or message losses.

3.3.1. Syntax

For capturing the probabilistic jump behaviours, a probabilistic guarded command $c$ is defined to have the form

$$\text{condition} \rightarrow p_1 : \text{update}_1 + \ldots + p_{q_c} : \text{update}_{q_c}$$

where $q_c \geq 1$ denotes the number of probabilistic branching of $c$, $p_i > 0$ for $i = 1, \ldots, q_c$ and $\sum_{i=1}^{q_c} p_i = 1$, condition is a constraint over $(m, \bar{x})$, and update$_i$ is an expression denoting a reset mapping for $m$ and $\bar{x}$ for all $i = 1, \ldots, q_c$. Intuitively, if a state $(m, \bar{x})$ satisfies the guard condition, a jump to states $(m_1, \bar{x}_1), \ldots, (m_{q_c}, \bar{x}_{q_c})$ occurs such that $(m_i, \bar{x}_i) = \text{update}_i(m, \bar{x})$ is selected with probability $p_i$ for $i = 1, \ldots, q_c$. Observe that for different $i \neq j$, it could be the case that $(m_i, \bar{x}_i) = (m_j, \bar{x}_j)$. In this paper we assume that $q_c$ is finite for all $c$. As for the non-probabilistic setting, we assume that conditions are enabled only if corresponding jumps lead to a state in which a flow exists.

An example for the case $n = 2, k = 1$ and $q_c = 2$ is:

$$(m = m_1 \land x \geq 10) \rightarrow 0.2 : (m' = m_1 \land x' = 0)$$
$$+ 0.8 : (m' = m_2 \land x' = x + 1).$$

Definition 3.4: A probabilistic hybrid automaton is a tuple $\mathcal{H} = (\text{Flow}, C, \text{Init}, \text{Unsafe})$ where Flow, Init, Unsafe are the same as in the hybrid automaton, and $C$ is a finite set of probabilistic guarded commands.

A probabilistic hybrid automaton induces a classical hybrid automaton where probabilistic branching is replaced by nondeterministic choices. Intuitively, the semantics of the latter spans the semantics of the former.

Definition 3.5: Let $c = (\text{condition} \rightarrow p_1 : \text{update}_1 + \ldots + p_{q_c} : \text{update}_{q_c})$ be a probabilistic guarded command. It induces a set of $q_c$ update constraints: $\text{ind}(c) = \{ u_1, \ldots, u_{q_c} \}$ where $u_i$ corresponds to the update condition $\text{update}_i$ for $i = 1, \ldots, q_c$. Moreover, for a set $C$ of probabilistic guarded commands we define $\text{ind}(C) = \bigcup_{c \in C} \text{ind}(c)$.

Let $\mathcal{H} = (\text{Flow}, C, \text{Init}, \text{Unsafe})$ be a probabilistic hybrid automaton. The induced hybrid automaton is a tuple $\text{ind}(\mathcal{H}) = (\text{Flow}, \text{ind}(C), \text{Init}, \text{Unsafe})$.

3.3.2. Semantics

The semantics of a probabilistic hybrid automaton is a probabilistic automaton [32]. Let $\mathcal{H} = (\text{Flow}, C, \text{Init}, \text{Unsafe})$ be a probabilistic hybrid automaton. Let $\text{ind}(\mathcal{H})$ denote the induced hybrid automaton, and let $T_{\text{ind}(\mathcal{H})} = (S, T, \text{Init}, \text{Unsafe})$ denote the transition relation representing the semantics of $\text{ind}(\mathcal{H})$. Recall that $T = T_C \cup T_D$ where $T_C$ corresponds to transitions due to continuous flows and $T_D$ corresponds to transitions due to discrete jumps.

The semantics of $\mathcal{H}$ is the probabilistic automaton $\mathcal{M}_H = (S, \text{Steps}, \text{Init}, \text{Unsafe})$ where $S, \text{Init}, \text{Unsafe}$ are the same as in $T_{\text{ind}(\mathcal{H})}$, and $\text{Steps}$ is defined as the union of two transition relations $\text{Steps}_C, \text{Steps}_D \subseteq S \times \text{Distr}(S)$. Here, as in the non-probabilistic setting, $\text{Steps}_C$ corresponds to transitions due to continuous flows, while $\text{Steps}_D$ corresponds to transitions due to discrete jumps. Both of them are defined respectively as follows.

For each transition $((m, \bar{x}), (m', \bar{x}')) \in T_C$ in $\text{ind}(\mathcal{H})$, there is a corresponding transition in $\mathcal{H}$ from $(m, \bar{x})$ to $(m', \bar{x}')$ with probability 1. So, $\text{Steps}_C$ is defined by:

$$\text{Steps}_C = \{ ((m, \bar{x}), \text{Dirac}(m', \bar{x}')) \mid ((m, \bar{x}), (m', \bar{x}')) \in T_C \}.$$ 

Now we discuss transitions induced by discrete jumps. First, for a probabilistic guarded command $c$, we define its corresponding set $\text{Steps}_D(c)$. Let $\text{ind}(c) = \{ u_1, \ldots, u_{q_c} \}$ be as defined in Definition 3.5. Then, for a number of $q_c + 1$ arbitrary states $(m, \bar{x}), (m_1, \bar{x}_1), \ldots, (m_{q_c}, \bar{x}_{q_c}) \in S$ satisfying the condition $((m, \bar{x}), (m_i, \bar{x}_i)) \in T_D(u_i)$ for $i = 1, \ldots, q_c$, we introduce the transition $((m, \bar{x}), \mu) \in \text{Steps}_D(c)$ with

$$\mu(m_i, \bar{x}_i) = \sum_{j \in \{ j \mid m_j = m_i \land x_j = x_i \} } p_j$$

for $i = 1, \ldots, q_c$. Then, $\text{Steps}_D$ is defined to be $\bigcup_{c \in C} \text{Steps}_D(c)$. Recall that we have assumed that $q_c$ is finite for all $c$. This implies $\text{Supp}(\mu)$ is finite for all transitions $(s, \mu)$ with $s \in S$.

3.3.3. Safety Properties

For hybrid automata, the safety property asserts that the unsafe states can never be reached. For probabilistic hybrid automata, however, the safety property expresses that the maximal probability of reaching the set $\text{Unsafe}$ is bounded by some given threshold $p$. In the following we fix a certain threshold $p$.

Let $\text{Reach}(\text{Unsafe})$ denote the set of paths $[\sigma] \in \text{Path} \mid \exists i. \sigma[i] \in \text{Unsafe}$. The automaton $\mathcal{H}$ is called safe if for each adversary $A$ and each initial state $s$ of $\mathcal{M}_H$, $\text{Prob}^A(\text{Reach}(\text{Unsafe})) \leq p$ holds. Since safety verification in the non-probabilistic setting is an undecidable problem [18], this implies that the safety problem in the
probabilistic setting is also undecidable. In the following, we develop a framework for a heuristic algorithm to deal with such a probabilistic safety verification problem for general probabilistic hybrid automata.

Example 3.6: Consider the probabilistic hybrid automaton illustrated in Fig. 1. It contains the modes $m_1$, $m_2$ and the continuous variables $x_1$, $x_2$ which both range over the interval $[0, 2]$, i.e., $S = \{m_1, m_2\} \times [0, 2] \times [0, 2]$ (note that the state space can be restricted via state-space constraints). The set of initial states is given by the constraint $\text{Init}(m, (x_1, x_2)) = (m = m_1 \land x_1 = 0 \land x_2 = 0)$. The constraint $\text{Unsafe}(m, (x_1, x_2)) = (x_1 \geq 1 \land x_2 \geq 1.5)$ describes the set of unsafe states. The hybrid automaton can switch modes from $m_1$ to $m_2$ with two possibilities if $x_2 = 1$, i.e.,

\[
(m = m_1 \land x_2 = 1) \quad \rightarrow 0.95 : (m' = m_2 \land x'_1 = x_1 \land x'_2 = x_2) \\
\quad \quad + 0.05 : (m' = m_2 \land x'_1 = x_1 \land x'_2 = x_2 + 0.5).
\]

The corresponding flow constraint is:

\[
(m = m_1 \rightarrow (x_1 = 1 \land x_2 = 1 \land 0 \leq x_1 \leq 1)) \\
\land (m = m_2 \rightarrow (x_1 = 1 \land x_2 = 1)).
\]

The constraint $0 \leq x_1 \leq 1$ in Flow forces a jump from mode $m_1$ to $m_2$ if $x_1$ becomes 1.

The maximal reachability probability is 0.05, as can be seen as follows: from the initial state, we can only apply the probabilistic guarded command once and exactly if we wait until $x_2 = 1$, as seen from the guard. With probability 0.95, we then move to a state with $m = m_2, x_1 = 1, x_2 = 1$. In this case, we cannot reach unsafe states, because in the following timed transitions $x_2$ is decreased and thus we will never have $x_2 \geq 1.5$. However, with probability 0.05 we jump to an unsafe state directly. Because of this, the probabilistic system is safe if we have a threshold of, e.g., p = 0.051 > 0.05.

3.3.4. Simulation Relations

We recall the notion of simulations between probabilistic automata. Intuitively, if $M_2$ simulates $M_1$, that is, $M_2$ is an overapproximation of $M_1$, then $M_2$ can mimic all behaviours of $M_1$. Thus, this allows us to verify safety properties on the abstraction $M_2$ instead of $M_1$.

To establish the notion of simulations, we introduce first the notion of weight functions [21], which establish the correspondence between distributions.

**Definition 3.7:** Let $\mu_1 \in \text{Distr}(S_1)$ and $\mu_2 \in \text{Distr}(S_2)$ be two distributions. For a relation $R \subseteq S_1 \times S_2$, a weight function for $(\mu_1, \mu_2)$ with respect to $R$ is a function $\Delta : S_1 \times S_2 \rightarrow [0, 1]$ such that

1. $\Delta(s_1, s_2) > 0$ implies $(s_1, s_2) \in R$,
2. $\mu_1(s_1) = \sum_{s_2 \in S_2} \Delta(s_1, s_2)$ for $s_1 \in S_1$, and
3. $\mu_2(s_2) = \sum_{s_1 \in S_1} \Delta(s_1, s_2)$ for $s_2 \in S_2$.

We write $\mu_1 \subseteq_R \mu_2$ if and only if there exists a weight function for $(\mu_1, \mu_2)$ with respect to $R$.

Now, we recall the notion of simulations [31], adapted to reachability properties. The simulation requires that every successor distribution of a state of $M_1$ is related to a successor distribution of its corresponding state of $M_2$ via a weight function.

**Definition 3.8:** Given two probabilistic automata $M_1 = (S_1, \text{Steps}_1, \text{Init}_1, \text{Unsafe}_1)$ and $M_2 = (S_2, \text{Steps}_2, \text{Init}_2, \text{Unsafe}_2)$, we say that $M_2$ simulates $M_1$, denoted by $M_1 \subseteq M_2$, if and only if there exists a relation $R \subseteq S_1 \times S_2$, which we will call simulation relation from now on, such that

1. for each $s_1 \in \text{Init}_1$ there exists an $s_2 \in \text{Init}_2$ with $(s_1, s_2) \in R$.
2. for each $s_1 \in \text{Unsafe}_1$ there exists an $s_2 \in \text{Unsafe}_2$ with $(s_1, s_2) \in R$, and there does not exist an $s' \in S_2 \setminus \text{Unsafe}_2$ such that $(s_1, s') \in R$.
3. for each pair $(s_1, s_2) \in R$, if there exists $(s_1, s_1) \in \text{Steps}_1$, then there exists a distribution $\mu_2 \in \text{Distr}(S_2)$ such that $(s_2, \mu_2) \in \text{Steps}_2$ and $\mu_1 \subseteq_R \mu_2$.

4. Abstractions for Probabilistic Hybrid Automata

Various abstraction refinement techniques have been developed for verifying safety properties of non-probabilistic hybrid automata. All of them have a common strategy: the set $S$ is covered by a finite set of abstract states, each representing a set of concrete states. Then, an abstraction is constructed, that is an overapproximation of the original system. Afterwards, the safety property is checked on the abstraction. If no abstract unsafe state is reachable, the original system is safe since the abstraction overapproximates the original system. If not, the covering
might have been chosen too coarse, and a refinement step is needed.

Following this idea, refinement techniques for abstractions based on subsuming concrete states by validity of predicates on the state variables have been used [4, 13, 30].

Let \( \mathcal{H} = (\text{Flow}, c, \text{Init}, \text{Unsafe}) \) be a probabilistic hybrid automaton. The aim of this section is—indeed of which abstraction technique is used—to develop a framework for constructing an abstraction for \( \mathcal{H} \), which is a finite probabilistic automaton. First we introduce the notion of abstract states, which form a (not necessarily disjoint) covering of the concrete state space:

**Definition 4.1:** An abstract state is a pair \((m, B)\) where \( m \in \mathcal{M} \) and \( B \subseteq \mathbb{R}^2 \). A covering \( \mathcal{B} \) is a finite set of abstract states such that \( \mathcal{S} = \bigcup \{(m, \bar{x}) \mid (m, B) \in \mathcal{B} \land \bar{x} \in B\} \).

In the above definition, for two abstract states \((m, B_1)\) and \((m, B_2)\) we might have \( B_1 \cap B_2 \neq \emptyset \). For instance, dependent on the abstraction technique used, \( B_1 \) and \( B_2 \) might have overlapping borders [30], or common interiors [17].

Fig. 2 illustrates how this section is organised. Given a probabilistic hybrid automaton \( \mathcal{H} \) and an abstract state space \( \mathcal{B} \), we introduce the quotient automaton of both \( \text{ind}(\mathcal{H}) \) and \( \mathcal{H} \) with respect to \( \mathcal{B} \) in Section 4.1. In Section 4.2, we show the soundness with respect to the quotient automaton (cf., Lemma 4.6 and Lemma 4.7).

The quotient automaton is in general hard to compute. Thus, in Section 4.3 we introduce general abstractions, which overapproximate the quotient automata conservatively. In Section 4.4, we discuss how the abstraction for the given probabilistic hybrid automaton is constructed (see Fig. 2): we first construct the abstraction of the induced hybrid automaton, from which the abstraction of the probabilistic setting is obtained afterwards.

### 4.1. Quotient Automaton for \( \mathcal{H} \)

We define the quotient automaton for the probabilistic hybrid automaton \( \mathcal{H} \). First we define the quotient automaton for the induced hybrid automaton \( \text{ind}(\mathcal{H}) \). As a convention, we use \( T, \mathcal{I}, \mathcal{U} \) to denote the set of transitions, initial states, unsafe states in the quotient automata.

**Definition 4.2:** Let \( \mathcal{H} = (\text{Flow}, c, \text{Init}, \text{Unsafe}) \) be a probabilistic hybrid automaton, and let \( \mathcal{B} \) denote the abstract state space. Further, let \( T_{\text{ind}}(\mathcal{H}) = (S, T_C U T_D, \text{Init}, \text{Unsafe}) \) denote the automaton representing the semantics of \( \text{ind}(\mathcal{H}) \). The quotient automaton for \( T_{\text{ind}}(\mathcal{H}) \), denoted by \( \text{Quo}_{\text{ind}}(\mathcal{H})(\mathcal{B}) \), is a finite transition system \((B, \mathcal{I}, \mathcal{U})\) where

- \( \mathcal{I} = \{(m, B) \in \mathcal{B} \mid \exists \bar{x} \in B. (m, \bar{x}) \in \text{Init}\} \)
- \( \mathcal{U} = \{(m, B) \in \mathcal{B} \mid \exists \bar{x} \in B. (m, \bar{x}) \in \text{Unsafe}\} \)
- \( T_C \) corresponds to the set of abstract transitions due to continuous flow: \( T_C = \{(m, B_1, (m', B_2)) \in B^2 \mid \exists \bar{x}, \bar{y} \in B. \bar{x} \neq \bar{y}

\)\( \bar{x} \in B_1, (m, \bar{x}), (m', \bar{y}) \in T_C\} \)
- \( T_D \) corresponds to the set of abstract transitions due to discrete jumps. We first define the transition induced by one fixed update \( u \in \text{ind}(c) \): \( T_D(u) = \{(m, B_1, B_2) \in B^2 \mid \exists \bar{x} \in B. \bar{x} \in B_1, (m, \bar{x}), (m', \bar{x}) \in T_D(u)\} \). Then, let \( T_D = \bigcup_{u \in \text{ind}(c)} T_D(u) \).

In the following let \( \mathcal{H} = (\text{Flow}, c, \text{Init}, \text{Unsafe}) \) be a probabilistic hybrid automaton, let \( \mathcal{M}_H = (S, \text{Steps}_C U \text{Steps}_D, \text{Init}, \text{Unsafe}) \) denote the probabilistic automaton representing the semantics of \( \mathcal{H} \), and let \( \mathcal{B} \) be an abstract state space. As in the induced non-probabilistic setting, we define a quotient automaton, denoted by the probabilistic automaton \( \text{Quo}_{\mathcal{H}}(\mathcal{B}) \), for the abstract state space \( \mathcal{B} \). For this, we first introduce the set of lifted distributions:

**Definition 4.3:** Let \( \mathcal{H}, \mathcal{M}_H \) and \( \mathcal{B} \) be as described above. Let \( c \in \mathcal{C} \) and assume that \((s, \mu) \in \text{Steps}_D(c) \) in \( \mathcal{M}_H \). By definition of \( \text{Steps}_D(c) \), there exist states \((m_1, \bar{x}_i), \ldots, (m_{q_{c}}, \bar{x}_{q_{c}}) \in S \) satisfying the condition \((m, \bar{x}, (m_i, \bar{x}_i)) \in T_D(u_i)\) for \( i = 1, \ldots, q_{c} \). Then, for arbitrary abstract states \((m_1, B_1), \ldots, (m_{q_{c}}, B_{q_{c}})\) with \( \bar{x}_i \in B_i \) for \( i = 1, \ldots, q_{c} \) we introduce the distribution \( \mu' \in \text{Distr}(\mathcal{B}) \) with:

\[
\mu'(m_i, B_i) = \sum_{j \mid (m, B_i) = (m_j, \bar{x}_j)} \mu(m_j, \bar{x}_j).
\]

The set of lifted distributions lift\( \mathcal{B}(\mu) \) contains all such \( \mu' \).

Let \( \mu \) be the distribution according to a probabilistic guarded command \( c \). Since the covering \( \mathcal{B} \) is in general not disjoint, a concrete state \((m_i, \bar{x}_i)\) might belong to more than one abstract states. In this case, \( \mu \) induces more than one
Safety Verification for Probabilistic Hybrid Systems

Fig. 3. Illustrating the abstract discrete transitions in the quotient automaton.

lifted distribution. In the above definition, this is reflected by the way of defining one specific lifted distribution \( \mu' \), for which we first fix to which abstract state each concrete state \((m_i, \bar{x}_i)\) belongs. Note that if \( B \) is a disjoint partitioning of \( S \), then the set \( \text{lift}_B(\mu) \) is a singleton. We now introduce the quotient automaton for the probabilistic hybrid automaton:

**Definition 4.4:** Let \( \mathcal{H}, \mathcal{M}_H \) and \( B \) be as described above. The quotient automaton for \( \mathcal{M}_H \) with respect to \( B \) is defined by \( \text{Quo}_H(\mathcal{B}) = (B, ST, \tau, UT) \) where \( \tau \) and \( UT \) are defined as for \( \text{Quo}_{\text{ind}}(\mathcal{H})(\mathcal{B}) \), and \( ST = ST_C \cup ST_D \) is the set of abstract transitions where:

- \( ST_C \) corresponds to the set of abstract transitions due to continuous flows: \( ST_C = \{(m, B), \text{Dirac}_{(m, B')} \} \in B \times \text{Distr}(B) \mid \exists \bar{x} \in B, \exists \bar{x}' \in B'. ((m, \bar{x}), (m, \bar{x}')) \in \text{Steps}_C \).
- \( ST_D \) corresponds to the set of abstract transitions due to discrete jumps. We first define the transitions induced by one fixed probabilistic guarded command \( c \). We let \( ST_D(c) = \{(m, B), \mu' \} \in B \times \text{Distr}(B) \mid \exists \bar{x} \in B, \exists \bar{x}' \in B'. ((m, \bar{x}), (m, \bar{x}')) \in \text{Steps}_{D}(c), \mu' \in \text{lift}_B(\mu) \}. \) Then, let \( ST_D = \bigcup_{c \in C} ST_D(c) \).

**Example 4.5:** Consider Fig. 3 and assume we have a probabilistic guarded command \( c = \text{condition} \rightarrow p_1 : u_{p_1} + \ldots + p_4 : u_{p_4} \). Thus \( q_c = 4 \). The abstract states are represented by circles, labelled with the corresponding type. The concrete states are represented by black points, labelled with only the evaluation of the variables (assume that all of them are different). Thus the point labelled with \( s_0 \) represents the state \((m_0, s_0)\) and so on. Arrows are transitions in the concrete models, where the labels represent the probability \( p_i \) of the corresponding update \( u_{p_i} \) of \( c \).

Consider the two (there may be more) concrete transitions in \( \text{Ind}_{\text{ind}}(\mathcal{H}): \{(m_0, s_0), (m_1, s_1)\}, \{(m_0, s_0), (m_1, s_2)\} \in T_D \). Both of them lead from \((m_0, B_0)\) to the same abstract state \((m_1, B_1)\). By Definition 4.2, we have that \(((m_0, B_0), (m_i, B_i)) \in T_D \) for \( i = 1, 2, 3, 4 \) in \( \text{Quo}_{\text{ind}}(\mathcal{H})(\mathcal{B}) \).

We have a concrete transition \(((m_0, s_0), \mu) \) where \( \mu \) is defined by: \( \mu(s_i) = p_i \) for \( i = 1, 2, 3, 4 \). Assume first that \( B_0, B_1, B_2, B_3 \) are disjoint. By Definition 4.3, \( \text{lift}_B(\mu) = \{\mu'\} \) where \( \mu' \) is defined by: \( \mu'(m_1, B_1) = p_1 + p_2, \mu'(m_2, B_2) = p_3, \) and \( \mu'(m_3, B_3) = p_4 \). Then, by Definition 4.4, this induces an abstract transition \(((m_0, B_0), \mu') \in ST_D \) in \( \text{Quo}_H(\mathcal{B}) \).

Assume now that the abstract states \( B_1 \) and \( B_2 \) are not disjoint, and that \( s_2 \) is on the common border of \((m_1, B_1) \) and \((m_2, B_2) \) (which implies also \( m_1 = m_2 \)). In this case, the set \( \text{lift}_B(\mu) \) contains another element \( \mu'' \) which defined by: \( \mu''(m_1, B_1) = p_1, \mu''(m_2, B_2) = p_2 + p_3 \) and \( \mu''(m_3, B_3) = p_4 \). Again by Definition 4.4, \( \mu'' \) induces another abstract transition \(((m_0, B_0), \mu'') \) in \( \text{Quo}_H(\mathcal{B}) \).

### 4.2. Soundness

Given a probabilistic hybrid automaton \( \mathcal{H} \) and a set of abstract states \( B \), we defined a probabilistic quotient automaton \( \text{Quo}_H(\mathcal{B}) \). The following lemma shows that this automaton conservatively overapproximates \( \mathcal{M}_H \).

**Lemma 4.6:** \( \text{Quo}_H(\mathcal{B}) \) simulates \( \mathcal{M}_H \).

**Proof:** We define \( R = \{(m, \bar{x}), (m', B) \} \in S \times B \mid m = m' \wedge \bar{x} \in B \}. \) It suffices to show that \( R \) is a simulation relation. \((m, \bar{x}), (m', B) \) is a pair. The first two conditions for simulation relations are trivially satisfied. It remains to show the third condition. There are two types of transitions starting from \((m, \bar{x}) \) in \( \mathcal{M}_H \): the case \((m, \bar{x}), \text{Dirac}_{(m, m')} \in \text{Steps}(c) \) is trivial and skipped. Now consider the case \((m, \bar{x}), \mu \in \text{Steps}_D(c) \). \( \exists \bar{x} \in B, \exists \bar{x}' \in B'. ((m, \bar{x}), (m, \bar{x}')) \in \text{Steps}_D(c), \mu' \in \text{lift}_B(\mu) \}. \) Then, let \( ST_D = \bigcup_{c \in C} ST_D(c) \). Let \( c \) and \( \text{Ind}(c) = \{u_1, \ldots, u_{q_c}\} \) be as described in Definition 3.5, and let \((m_i, \bar{x}_i) = u_i(m, \bar{x}) \), where assume that \( i = 1, \ldots, q_c \). Note that it could be the case that, for \( i \neq j \), we have \( \bar{x}_i = \bar{x}_j \). Moreover, let \((m_i, B_i) \in B \) denote an abstract state satisfying \( \bar{x}_i \in B_i \). By construction of the relation \( R \), we know that \(((m_i, \bar{x}_i), (m_j, B_j)) \in R \). By the definition of \( \text{Steps}(c) \), we have that \(((m, B), \mu') \in \text{Steps}(c) \) where \( \mu'(m_i, B_i) = \sum_{j \in \text{Ind}(c) \cap \{m = m_1 \wedge B = B_1\}} \mu_j \). Define \( \Delta \) for \((m, \bar{x}) \) with respect to \( B \) by: \( \Delta((m_i, \bar{x}_i), (m_j, B_j)) \) equals \( \mu(m_i, \bar{x}_i) \) for \( i = 1, \ldots, q_c \), and equals 0 otherwise. It remains to show that \( \Delta \) is the proper weight function. For the first condition, assume \( \Delta((m', \bar{x}'), (m', B')) > 0 \). By the definition of \( \Delta \), we have \( m' = m' \) and \( \bar{x}' \in B' \), implying \((m', \bar{x}'), (m', B')) \in R \). Now we show the second condition (the second condition is similar). Let \((m_i, B_i) \) be an abstract state (otherwise trivial). On the one hand, due
to the definition of $\mu'$, it is $\mu'(m_j, B_j) = \sum_{i \in I} p_i$ where $I = \{ i \mid m_i = m_j \land B_i = B_j \}$ denotes the set of all indices $i$ such that $(m_i, B_i) = (m_j, B_j)$. On the other hand, by the definition of $\Delta$, it holds $\sum_{i \in I} p_i = \sum_{x_i \in B_j} \mu(m_j, x_j) = \sum_{k \in \epsilon} \Delta((m_j, x_j), (m_j, B_j))$ (cf., Eq. (1)), which implies the third condition.

Since simulation on probabilistic automata preserves safety properties [31], we have the correctness of our construction:

**Lemma 4.7:** The abstraction preserves the safety property: if the probability of reaching UnSafe in $Quo_H(B)$ is bounded by $p$, this is also the case in $\mathcal{H}$.

### 4.3. Abstractions for $\mathcal{H}$

Consider the probabilistic hybrid automaton $\mathcal{H}$. The computation of the exact quotient automaton $Quo_{\mathcal{H}}(B)$ as defined in Definition 4.4 refers to concrete states, and often is hard or even impossible. In this subsection, we introduce the notion of abstractions which overapproximate the quotient automata. As a convention, we use the primed version $T', \mathcal{I}', \mathcal{U}'$ to denote the set of transitions, initial states, unsafe states in the abstraction.

**Definition 4.8:** Let $\mathcal{H} = \langle \text{Flow}, c, \text{Init}, \text{Unsafe} \rangle$ be a probabilistic hybrid automaton, and let $B$ denote the abstract state space. Then,

- $Abs_{\mathcal{H}}(B) = (B, T', \mathcal{I}', \mathcal{U}')$ is an abstraction of the quotient $Quo_{\mathcal{H}}(B)$ iff $T' = \bigcup_{u \in \text{ind}(c)} T'_c(u) \cup T'_c^c$ and it holds $T_c \subseteq T'_c$, $T'_c(u) \subseteq T'_c(u)$ for all $u \in \text{ind}(c)$, and we have $\mathcal{I} \subseteq \mathcal{I}'$ and $\mathcal{U} \subseteq \mathcal{U}'$.
- $Abs_{\mathcal{H}}(B) = (B, ST', \mathcal{I}', \mathcal{U}')$ is an abstraction of the quotient $Quo_{\mathcal{H}}(B)$ iff $ST' = \bigcup_{c \in C} ST'_c(c) \cup ST'_c^c$ and it holds $ST'_c(c) \subseteq ST'_D(c)$ for all $c \in C$, and it is $ST_c \subseteq ST'_c$, $\mathcal{I} \subseteq \mathcal{I}'$, and $\mathcal{U} \subseteq \mathcal{U}'$.

In that case, we say also that $Abs_{\mathcal{H}}(B)$ is an abstraction of the induced hybrid automaton $\text{ind}(\mathcal{H})$. Similarly, we say also that $Abs_{\mathcal{H}}(B)$ is an abstraction of the probabilistic hybrid automaton $\mathcal{H}$. Since the abstraction as defined may have more initial states, unsafe states and transitions than the quotient automaton, it is easy to verify that the abstraction simulates the corresponding quotient automaton. Since simulation is transitive, the abstraction also simulates the corresponding semantics automaton. Thus, the abstraction preserves also safety properties of $\mathcal{H}$.

### 4.4. Computing Abstractions

Let $\mathcal{H}$ be a probabilistic hybrid automaton. Existing methods [4, 17, 30] can be used to compute an abstraction $Abs_{\mathcal{H}}(B)$ for the induced hybrid automaton $\text{ind}(\mathcal{H})$.

In the following, we define an abstraction based on $Abs_{\text{ind}(\mathcal{H})}(B)$:

**Definition 4.9:** For a probabilistic hybrid automaton $\mathcal{H}$, let $B$ be the abstract state space, and $Abs_{\text{ind}(\mathcal{H})}(B) = (B, T'_D, \mathcal{I}', \mathcal{U}')$ be an abstraction of $\text{ind}(\mathcal{H})$. We define $Abs_{\mathcal{H}}(B) = (B, ST'_D \cup ST'_D^c, \mathcal{I}', \mathcal{U}')$ for $\mathcal{H}$ as follows:

- $ST'_D = \{(m, B), \text{Dirac}(m, B') \in B \times \text{Distr}(B) \mid ((m, B), (m, B')) \in T'_c(c)\}$.
- $ST'_D^c$ corresponds to the set of abstract transitions due to discrete jumps. We first define the transition induced by one fixed probabilistic guarded command $c = (\text{condition} \rightarrow p_1 : \text{update}_1 \ldots + p_4 : \text{update}_4$, and assume that all of the concrete states are different and are not

**Lemma 4.10:** Consider the abstraction $Abs_{\mathcal{H}}(B)$ as defined in Definition 4.9. Then, it holds that $ST'_D(c) \subseteq ST'_D^c(c)$, for all $c \in C$.

**Proof:** Fix $c \in C$. Assume that $((m, B), \mu') \in ST'_D(c)$. Then, by Definition 4.4, there exists $\bar{x} \in B$ and a transition $((m, \bar{x}), \mu) \in \text{Steps}_D(c)$ such that $\mu' = \text{lift}(\mu)$. For $i = 1, \ldots, q_c$, let $(m_i, x_i) = \text{update}(m, \bar{x})$, and let $(m_i, B_i)$ be the abstract states corresponding to the distribution $\mu'$ (cf., Definition 4.3), i.e., $\mu'(m_i, B_i) = \sum_j [m_i = m_j \land B_i = B_j] \mu(m_j, x_j)$. Obviously, $((m, \bar{x}), (m_i, x_i)) \in T_D(u_i)$. Since $x_i \in B_i$ it holds that $((m, B), (m_i, B_i)) \in T_D(u_i) \subseteq T'_D(u_i)$ for $i = 1, \ldots, q_c$. By Definition 4.9, we have that $((m, B), \mu') \in ST'_D^c(c)$.

The set of transitions $ST'_D^c(c)$ is indeed an overapproximation, which is illustrated as follows.

**Example 4.11:** Consider the fragment of the abstraction depicted in Fig. 4 in which we assume that the transitions correspond to the probabilistic guarded command $c$ with $q_c = 4$: condition $\rightarrow p_1 : \text{up}_1 \ldots + p_4 : \text{up}_4$. Assume that the set of the concrete states are different and are not
Fig. 4. Abstracting abstract discrete transitions.

on borders. (Note: only parts of successor distributions are depicted, and we assume that other parts, e.g. for state \((m_0, s_1)\), lead to abstract states outside the depicted fragment.)

Now we consider the distribution \(\mu^* \in \text{Distr}(B)\) which is defined as follows: \(\mu^*(m_1, B_1) = p_1 + p_2, \mu^*(m_2, B_2) = p_3\) and \(\mu^*(m_3, B_3) = p_4\). By the above assumption, no concrete successor distributions of \(s_0, s_1\) or \(s_2\) could induce \(\mu^*\) according Definition 4.3. Thus, by Definition 4.4, \(((m_0, B_0), \mu^*) \notin STP(\zeta)\). On the other hand, it holds \(((m_0, B_0), (m_1, B_1)) \in \text{TP}(u_1)\) for \(i = 1, 2\), \(((m_0, B_0), (m_2, B_2)) \in \text{TP}(u_3)\), and \(((m_0, B_0), (m_3, B_3)) \in \text{TP}(u_4)\). Thus, by Definition 4.9 we have that \(((m_0, B_0), \mu^*) \in STP(\zeta)\).

Lemma 4.10 implies that \(Abs_H(B)\) is an abstraction of \(Quo_H(B)\). Thus:

**Theorem 4.12:** For every probabilistic hybrid automaton \(H\), for every abstraction \(Abs_{ind}(H)(B)\) of the induced hybrid automaton \(ind(H)\), the safety of \(Abs_H(B)\) implies the safety of \(H\).

### 4.5. Computing Reachability Probabilities

First we note that, similar as in the non-probabilistic case, only abstract states reachable from the initial abstract states are of interest. Below, we discuss briefly how to compute the maximal probability of reaching the set of abstract unsafe states in the abstraction.

Given an initial state \(s\), the maximal probability of reaching the set of abstract unsafe states UnSafe from \(s\) is denoted by

\[
\text{Prob}_{\text{\text{UnSafe}}}^s (\text{Reach}(\text{UnSafe})) = \sup_A \text{Prob}_{\text{\text{UnSafe}}}^A (\text{Reach}(\text{UnSafe})),
\]

where \(A\) ranges over all adversaries. Given the threshold \(p\), the safety property is satisfied if \(\text{Prob}_{\text{\text{UnSafe}}}^s (\text{Reach}(\text{UnSafe})) < p\) for all initial states \(s\). The probability \(\text{Prob}_{\text{\text{UnSafe}}}^s (\text{Reach}(\text{UnSafe}))\) can be characterised [7] as follows: it equals 1 if \(s \in \text{UnSafe}\), it is 0 if \(\text{UnSafe}\) can’t be reached from \(s\) at all, and otherwise it is

\[
\max_{\mu \in \text{Steps}(s)} \sum_{s' \notin S} \mu(s') \text{Prob}_{\text{\text{UnSafe}}}^{s'} (\text{Reach}(\text{UnSafe})).
\]

This suggests the following well-known value iteration approach [6]. As a preprocessing, those states which cannot be reached by any initial state, or cannot reach the unsafe states can be pruned, since the probabilities for those states are 0. We iteratively compute the solution vector \(\bar{\nu}\) during the algorithm, and let \(\bar{\nu}^i\) denote the probability values in the \(i\)-th iteration. Initially, we let \(\bar{\nu}^0(s) = 1\) if \(s \in \text{UnSafe}\), and \(\bar{\nu}^0(s) = 0\) otherwise. The vector \(\bar{\nu}^{i+1}\) can then be computed recursively according to \(\bar{\nu}^{i+1}(s) = 1\) if \(s \in \text{UnSafe}\) and

\[
\bar{\nu}^{i+1}(s) = \max_{\mu \in \text{Steps}(s)} \sum_{s' \notin S} \mu(s') \bar{\nu}^i(s')
\]

else. Hence, \(\bar{\nu}(s)\) corresponds to the maximal probability of reaching \(\text{UnSafe}\) within a number of \(i\) steps, and moreover, it converges to \(\text{Prob}_{\text{\text{UnSafe}}}^s (\text{Reach}(\text{UnSafe}))\) if \(i \to \infty\).

In practice, a stopping criterion is needed to decide when the approximation is tight enough: in our tool (see Section 6) we choose the relative error stopping criterion, namely we stop if \(\max_s |\bar{\nu}^{i+1}(s) - \nu_i(s)| / |\bar{\nu}^{i+1}(s)| \leq \epsilon\) with \(\epsilon = 10^{-8}\).

### 5. An Illustrating Example

As an illustrating example, we here consider the thermostat example depicted in Fig. 5, which is extended from the one of Alur et al. [4]. There are four modes: Cool, Heat, Check and Error. The latter mode models the occurrence of a failure, where the temperature sensor gets stuck at the temperature checked last. The set of variables is \(\{t, x, T\}\), where \(T\) represents the temperature, \(t\) represents a local timer and \(x\) is used to measure the total time passed so far. Thus, in all modes it holds that \(i = 1\) and \(i = 1\). In each mode there is also an invariant constraint restricting the state space of this mode. Invariant constraints are used only for the sake of convenience and comparison with [4].

The given initial condition is \(m = \text{Heat} \wedge t = 0 \wedge x = 0 \wedge 9 \leq T \leq 10\). The unsafe constraint is \(m = \text{Error} \wedge x \leq 5\), which corresponds to reaching the Error mode within 5 time units. Assume that the probability threshold for this risk is specified to be \(p = 0.2\).

First we observe that initially it is \(s = 0\) and \(9 \leq T \leq 10\). The system cannot stay in the mode Heat for 2 time units, as this would increase the temperature by 4 units which violates the invariant \(T \leq 10\) at Heat. This means that the
Fig. 5. A probabilistic hybrid automaton for the thermostat.

system must switch to mode Cool to decrease the temperature. It would take some amount of time (approximately 0.41) units to decrease the temperature until reaching 6 such that the system can go back to Heat. Note that switching back to Heat would reset the local timer which implies that $t = 0$ and that $x \geq 0.41$. To reach the unsafe mode Error, the intermediate mode Check must be touched. Because of the guard $t \geq 2$ between Heat and Check, once the mode Check is reached, it holds that $t = 0$ and $x \geq 2.41$. Then, the system waits in Check for at least 0.5 time units. After the probabilistic jump from Check is triggered, it holds that $x \geq 2.91$. Then, the unsafe state could be reached with probability 0.05. With probability 0.95 the system goes back to mode Heat and it holds that $t = 0$ and $x \geq 2.91$. Reiterating the above analysis, reaching Error from Heat would again take at least 2.5 time units, which implies that there is only one chance to hit the unsafe mode Error within 5 time units. Thus, the probability is bounded by 0.05, which implies that the safety property is indeed satisfied.

Now we consider an abstraction of the thermostat example. The initial abstract state is $(Heat, B)$, where $B$ represents concrete valuations satisfying the constraint $t \geq 0, x \geq 0, t = x$ and $T \leq 10$. In Fig. 6, we depicted fragments of the abstract states and of those abstract transitions which lead to abstract unsafe states. Notably, in this abstraction there are two chances to touch abstract unsafe states, thus the probability amounts to $0.05 + 0.95 \cdot 0.05 = 0.0975$. The reason is that from the initial state Heat the abstract system does not need go back to Cool to let the temperature decrease, instead it can immediately switch to Check. This is due to the overapproximation of the abstract initial states. However, the computed probability for the threshold $p = 0.2$ is still good enough to prove the safety property. If instead the threshold were set between 0.05 and 0.0975, refinement would have been needed.

6. Experiments

We implemented our method in the prototypical tool ProHVer (probabilistic hybrid automata verifier). It combines a modified version of PHAVer [17] to obtain the abstract state space with a component to compute an upper probability bound for the reachability problem using value iteration (cf., Subsection 4.5) in the induced abstract probabilistic automaton.

PHAVer can handle continuous dynamics with constant bounds on the derivatives exactly. In order to be able to handle affine dynamics not in this class, it uses overapproximation. Here it splits locations (introducing new discrete locations) such that the continuous state space of the resulting locations are polyhedra of a predefined maximal width. It is possible to improve the precision of the overapproximation by reducing this maximal width. However, the resulting covering and location splits can look entirely different, which is also reflected on the probabilistic side. This phenomenon of PHAVer may induce situations, where that reduced width setting does not lead to tighter probability bounds. Usually, however, bounds do improve.

To show the applicability of our approach, we applied ProHVer to several case studies, which are small but diverse in the nature of their behaviour. In most of the examples considered, we focus on reachability probabilities with upper time bounds (obtained by using an additional clock to measure time), because these correspond to very natural verification problems for the settings considered. Notably, our method is not restricted to time-bounded reachability. Actually, time-unbounded problems are simpler (because no additional clock is needed). Experiments were run on a Pentium 4 with 2.16 GHz and 2 GB RAM.
Table 1. Bouncing ball performance figures. Where appropriate, we round probabilities to three decimal places and denote multiples of 1000 by “K”.

<table>
<thead>
<tr>
<th>Time bound</th>
<th>Interval length 0.15</th>
<th>Interval length 0.15, hull</th>
<th>Interval length 0.05, hull</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prob. Build (s) #states</td>
<td>Prob. Build (s) #states</td>
<td>Prob. Build (s) #states</td>
</tr>
<tr>
<td>1</td>
<td>0.25 1 38</td>
<td>0.25 2 59</td>
<td>0.25 9 185</td>
</tr>
<tr>
<td>2</td>
<td>0.5 13 408</td>
<td>0.5 4 124</td>
<td>0.5 22 347</td>
</tr>
<tr>
<td>3</td>
<td>0.5 70 11K</td>
<td>1 5 145</td>
<td>0.5 32 425</td>
</tr>
<tr>
<td>3.5</td>
<td>0.5 140 14K</td>
<td>1 6 150</td>
<td>0.5 37 436</td>
</tr>
<tr>
<td>3.7</td>
<td>0.5 214 18K</td>
<td>1 6 154</td>
<td>— — —</td>
</tr>
</tbody>
</table>

6.1. Bouncing Ball

We consider a bouncing ball assembled from different materials. Assume this ball consisting of three parts: 50 per cent of its surface consists of hard material, 25 per cent consists of a material of medium hardness and the rest consists of very soft material. We drop the ball from a height $h = 2$. Further, we assume the gravity constant to be $g = 1$. If the ball hits the floor and falls on the hard side, it will jump up again with one half of the speed it had before, and if it falls down on the medium side it will jump up with a quarter of the speed. However, if it hits the ground with its soft side, it won’t jump up again. We assume that the side with which the ball hits the ground is determined randomly, according to the amount of surface covered by each of the materials.

We study the probability that the ball falls on the soft side before a given time bound. Results are given in Table 1. We conducted three main analysis settings. With “interval length” we denote the splitting interval for the PHAVer-specific refinement technique, as described in the beginning of Section 6. In the left and medium part of the table, we used partitioning with interval length of 0.15 on the position and speed variables. For the medium part, we used the PHAVer-specific convex hull overapproximation [17]. For the right part of the table, we used an interval length of 0.05, and the convex hull overapproximation. Entries for which the analysis did not terminate within one hour are marked by “—”.

We ascertain here that, without the convex hull overapproximation, with an interval of length 0.15, we obtain non-trivial upper bounds. However, the analysis time as well as the number of states grows very fast with increasing time bound. The reason for this to happen is, that each time the ball hits the ground, there are three possibilities with which side it will hit the ground. Thus, the number of possibilities for the amount of energy the ball still has after a number of times $n$ the ground is hit is exponential in $n$. Also notice that there is some time bound up to which the ball has done an infinite number of jumps in all cases, as this case study features Zeno behaviour. This indicates that for time bounds near this value we have to use very small partitioning intervals to obtain realistic probability bounds. This leads to an even larger time and memory consumption in the analysis.

If we use the convex hull overapproximation and an interval length of 0.15, far less resources have to be used. But we only obtain non-trivial results for time bounds up to 3. Using an interval width of 0.05, we obtain a tighter probability bound, while using still less resources than the first configuration. However, for time bound $T = 3.7$ the third configuration does not terminate within one hour.

For a time bound of 3, we can compute the probability manually: the ball can hit the ground with its soft side when it hits the floor in first place with a probability of 0.25. It hits the ground initially with its hard side with a probability of 0.5. In this case, it won’t hit the ground again before $T = 3$. If it first hits the ground with its medium hard side, it will hit the ground a second time before $T = 3$. After this, there is no chance of touching the ground again before the time bound. Because of this, the overall probability of the property under consideration is $0.25 + 0.25 \cdot 0.25 = 0.3125$. This means that the result obtained by using an interval length of 0.05 and the convex hull overapproximation is exact.

6.2. Thermostat

We consider the thermostat example discussed in Section 5. For the safety property discussed there, ProHVer can verify this non-trivial system and property, and will answer that the system is safe, the upper bound computed is 0.097.

In Table 2, we give further probability bounds and performance statistics (time to build the abstraction—the value iteration time is negligible—and number of constructed abstract states) for different time bounds. For the left (right) part of the table, we set the interval length for the variable $x$ to 2 (respectively 10). The time needed for the analysis as well as the number of states of the abstract transition systems grows
Table 2. Thermostat performance figures.

<table>
<thead>
<tr>
<th>Time bound</th>
<th>Prob. Build (s)</th>
<th>#states</th>
<th>Interval length 2</th>
<th>Prob. Build (s)</th>
<th>#states</th>
<th>Interval length 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.05</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.097</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.370</td>
<td>20</td>
<td>916</td>
<td>1</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0.642</td>
<td>68</td>
<td>2207</td>
<td>0.512</td>
<td>30</td>
<td>609</td>
</tr>
<tr>
<td>80</td>
<td>0.884</td>
<td>134</td>
<td>4916</td>
<td>1</td>
<td>96</td>
<td>1717</td>
</tr>
<tr>
<td>120</td>
<td>0.940</td>
<td>159</td>
<td>4704</td>
<td>0.878</td>
<td>52</td>
<td>1502</td>
</tr>
<tr>
<td>160</td>
<td>0.986</td>
<td>322</td>
<td>10195</td>
<td>0.954</td>
<td>307</td>
<td>4260</td>
</tr>
<tr>
<td>180</td>
<td>0.986</td>
<td>398</td>
<td>10760</td>
<td>0.961</td>
<td>226</td>
<td>3768</td>
</tr>
<tr>
<td>600</td>
<td>1.0</td>
<td>1938</td>
<td>47609</td>
<td>1</td>
<td>1101</td>
<td>12617</td>
</tr>
</tbody>
</table>

about linearly in the time bound, though with oscillations. Comparing the left and the right side, we see that for the larger interval we need less resources, as was to be expected. Due to the way PHAVer splits locations along intervals, for some table entries, we see somewhat counter-intuitive behaviour. We observe that bounds do not necessarily improve with decreasing interval length. This is because PHAVer computes abstractions without taking into account probabilities, and hence does not guarantee abstractions with smaller intervals to be an improvement of the probability bound, though they are in most cases. Moreover, the abstractions we obtain from PHAVer also do not necessarily guarantee probability bounds to increase monotonically with the time bound. This is because a slightly increased time bound might induce an entirely different abstraction, leading to a tighter probability bound, and thus giving the impression of a decrease in probability, even though the actual maximal probability indeed stays the same or increases.

6.3. Water Level Control

We consider a model of a water level control system (extended from the one of Alur et al. [3]) which uses wireless sensors. Values submitted are thus subject to probabilistic delays, due to the unreliable transport medium.

A sketch of the model is given in Fig. 7. The water level y of a tank is controlled by a monitor. Its change is specified by a linear function. Initially, the water level is y = 1. When no pump is turned on (s0), the tank is filled by a constant stream of water (˙y). This water streams out of a larger water tank, which initially has a water level of W. We thus have W = −1 in all modes of the system, and stop the system as soon as W reaches 0. For clarity, we do not explicitly included the dynamics of W in Fig. 7. When a water level of y = 10 is seen by a sensor of the tank, the pump should be turned on. However, the pump features a certain delay, which results from submitting control data via a wireless network. With a probability of 0.95 this delay takes 2 time units (s1), but with a probability of 0.05 it takes 3 time units (s′). The delay is realised by the timer x. After the delay has passed, the water is pumped out with a higher speed than it is filled into the tank (˙y = −2 in s2). There is another sensor to check whether the water level is below 5. If this is the case, the pump must turn off again. For the system to work correctly, the water level must stay between 1 and 12.

We are interested in the probability that the pump system violates the property given above, that is either the water level falls below 1 or grows above 12, before the larger water tank from which the water tank is filled becomes empty. We model the system in ProHVer and reason about this property: performance statistics are given in Table 3. Without using partitioning, we were only able to obtain exact values for W up to 82 including. Notice that we did not use the convex hull overapproximation, like in the bouncing ball case study, nor another overapproximation. For W larger than this value, we always obtained a probability limit of 1. To get tighter results, we partitioned x by an interval of length 2. For W below 83 we obtain the exact value in both table parts, whereas for 83 we obtain a useful upper bound only when using partitioning.

A plot of probabilities for different W is given in Fig. 8. The graph has a staircase form where wide steps alternate with narrow ones. This form results, because each time the longer time bound is randomly chosen, the tank will overflow or underflow respectively, if there is enough time left. The wide steps corresponds to the chance of
overflow in the tank, the narrow ones to the chance of underflow.

6.4. Autonomous Lawn-Mower

We consider an autonomous lawn-mower that uses a probability bias to avoid patterns on lawns. This mower is started on a rectangular grassed area. When it reaches the border of the area, it changes its direction. To prevent the mower from following a simple cyclic pattern, this direction is randomly biased, such as to ensure that finally the entire area has been cut.

A sketch of the automaton is given in Fig. 9. There, \( l \) is the length and \( h \) the width of the area. The position of the mower on the area is given by \((x, y)\). With \((v_x, v_y)\) we denote the speed in \((x, y)\) direction, which the mower takes with a probability of 0.95 when reaching a border, whereas with \((v'_x, v'_y)\) denotes a variation of the speed that is taken with probability 0.05. Further, \((x_g, y_g)\) describes the mower’s initial position.

At the region with \( x \geq 90 \land x \leq 100 \land y \geq 170 \land y \leq 200 \) the owner of the lawn has left a tarpaulin. We are interested in the probability that within a time bound of \( t = 120 \) the mower hits the tarpaulin, thereby inevitably ripping it up.
6.5. Summary of Case Studies

We successfully applied our method on a number of case studies. However, it seems worthwhile to explore techniques more adapted to the generation of transition systems for probabilistic systems, especially by adjusting the splitting of states to a method better adapted to our needs. We may do so, for instance, by finding means to guarantee non-decreasing probability bounds when smaller intervals are used. Another possibility would be to do a more fine-grained splitting at places where this is useful to decrease the probability bound, while saving space by a coarser abstraction for parts of the model where this is sufficient to obtain a tight probability bound. Mostly, the upper bounds we could obtain were tight or exact (checked by manual inspection).

In Table 5, we give running times for the largest instances completed successfully:

<table>
<thead>
<tr>
<th></th>
<th>ball thermostat water mower</th>
</tr>
</thead>
<tbody>
<tr>
<td>214s ✓</td>
<td>398s ✓</td>
</tr>
</tbody>
</table>

Here, ✓ means success (exact or tight upper bounds) while ✗ means useful results for only some instances. Our implementation is effective in most cases considered, but there is still room for improvement in the ProHVer implementation to handle more complex case studies. PHAVer files of the case studies can be found on the homepage of ProHVer:

http://depend.cs.uni-sb.de/tools/prohver

7. Conclusions

In this paper we have discussed how to check safety properties for probabilistic hybrid automata. These models and properties are of central importance for the design and verification of emerging wireless and embedded real-time applications. Moreover, being based on arbitrary abstractions computed by tools for the analysis of non-probabilistic hybrid automata, improvements in effectiveness of such tools directly carry over to improvements in effectiveness of the technique we describe. The applicability of our approach has been demonstrated on a number of case studies, tackled using a prototypical implementation.

As future work we are investigating whether our approach can be adapted to the safety verification problem for more general probabilistic hybrid systems [9, 11], that is, systems with stochastic differential equations instead of ordinary differential equations.

Acknowledgement

The work of Lijun Zhang was partially supported by MT-LAB, a VKR Centre of Excellence. Part of this work was done while Lijun Zhang was at Saarland University and at the Computer Science, University of Oxford. The work of Zhikun She was partly supported by NSFC-61003021 and Beijing Nova Program, the one of Stefan Ratschan has been supported by GACR grant 201/08/J020, MŠMT grant OC10048, and by the institutional research plan AV0Z10030504. Holger Hermanns and Ernst Moritz Hahn were supported by the NWO-DFG bilateral project ROCKS, by the DFG as part of the Transregional Collaborative Research Centre SFB/TR 14 AVACS, and by the European Community’s Seventh Framework Programme under grant agreement no 214755. The work of Ernst Moritz Hahn has been partially supported by the Graduiertenkolleg “Leistungsgarantien für Rechnersysteme”.

We thank Josée Desharnais for pointing out an error in the definition of simulation relations in an earlier version of this paper.

References


Chapter 8

Providing a Basin of Attraction to a Target Region of Polynomial Systems by Computation of Lyapunov-like Functions

This chapter appeared as:

Stefan Ratschan and Zhikun She. “Providing a Basin of Attraction to a Target Region of Polynomial Systems by Computation of Lyapunov-Like Functions”. In: SIAM Journal on Control and Optimization 48.7 (2010), pp. 4377–4394. DOI: 10.1137/090749955 URL: http://link.aip.org/link/?SJC/48/4377/1
PROVIDING A BASIN OF ATTRACTION TO A TARGET REGION OF POLYNOMIAL SYSTEMS BY COMPUTATION OF LYAPUNOV-LIKE FUNCTIONS∗

STEFAN RATSCHAN† AND ZHIKUN SHE‡

Abstract. In this paper, we present a method for computing a basin of attraction to a target region for polynomial ordinary differential equations. This basin of attraction is ensured by a Lyapunov-like polynomial function that we compute using an interval based branch-and-relax algorithm. This algorithm relaxes the necessary conditions on the coefficients of the Lyapunov-like function to a system of linear interval inequalities that can then be solved exactly. It iteratively refines these relaxations in order to ensure that, whenever a nondegenerate solution exists, it will eventually be found by the algorithm. Application of an implementation to a range of benchmark problems shows the usefulness of the approach.

Key words. basin of attraction, stability, constraint solving, interval computation, algorithms

AMS subject classifications. 65G40, 68U99, 65P40

DOI. 10.1137/090749955

1. Introduction. A sufficient condition for verifying stability of ordinary differential equations (ODEs) is the existence of a Lyapunov function [16]. In cases where the differential equation is polynomial, due to decidability of the theory of real-closed fields [44], there is an algorithm that, for a given polynomial with parametric coefficients, decides whether there are instantiations of these parameters resulting in a Lyapunov function. However, all the existing decision procedures (e.g., implemented in the software packages QEPCAD [4] or REDLOG [11]), while able to solve impressively difficult examples, are not efficient enough to be able to solve this problem in practice.

Recently, a method based on sum of squares (SOS) decomposition [26, 27] has appeared that can compute Lyapunov functions for some realistic examples. However, being based on the classical stability, it does not allow reasoning about target regions (and hence it requires an equilibrium), and it results only in Lyapunov functions that characterize the behavior of the given system locally (around the equilibrium) or globally (on the whole state space).

Another option is to try to compute a Lyapunov function of a linearization of the nonlinear problem around a given equilibrium point, and compute a basin of attraction for this Lyapunov function with respect to the original, nonlinear problem [13, 10]. However, due to the information loss introduced by the linearization process, this basin of attraction will usually be very small.

In this paper we will provide an algorithm for computing a Lyapunov-like function for the original, nonlinear problem, which will provide us with a basin of attraction

∗Received by the editors February 17, 2009; accepted for publication (in revised form) April 20, 2010; published electronically July 1, 2010. This is an extended and revised version of an earlier conference paper [37].
†Institute of Computer Science, Czech Academy of Sciences, Prague, Czech Republic (stefan.ratschan@cs.cas.cz). This author was supported by GAČR grant 201/08/J020 and by the institutional research plan AV0Z100300504.
‡LMIB and School of Mathematics and Systems Science, Beihang University, Beijing, China (zhikun.she@buaa.edu.cn). This author was supported by grant NKBRPC-2005CB321902 and by the Beijing Nova Program.
to a given target region. This target region can, for example, be the smaller basin
of attraction obtained by linearization, hence ensuring attraction to the equilibrium.
This algorithm is also fit to deal with systems with small interval uncertainties—
a corresponding extension of the algorithms with the necessary interval arithmetic
computations is a simple homework exercise.

Our approach sets up a polynomial with parametric coefficients, substitutes this
polynomial into a constraint that formalizes Lyapunov style conditions, and then
solves this constraint for the parameters that form the coefficients of the polynomial.

The algorithm for solving this constraint employs a branch-and-relax scheme. It
relaxes the constraint to a system of linear interval inequalities that can then be solved
exactly [40]. Using a recursive decomposition of the state space into hyperrectangles,
it iteratively refines these relaxations. This ensures that, whenever a nondegenerate
solution exists, it will eventually be found by the algorithm.

We implemented our algorithm and tested our implementation on several exam-
pies.

The structure of the paper is as follows. In section 2 we show how to compute
a basin of attraction using an adapted version of the notion of a Lyapunov function;
in section 3 we describe our algorithm for computing such Lyapunov-like functions;
in section 4 we describe three improvements to the basic algorithm; in section 5 we
provide a rigorous formal efficiency comparison of the algorithm with earlier algo-
rithms, and prove its termination for nondegenerate inputs; in section 6 we discuss
our implementation; in section 7 we give eleven examples together with correspond-
ing computational results; in section 8 we discuss related work; and in section 9 we
conclude our paper.

2. Basins of attraction to a target region. In this section we will introduce
the basic mathematical notions used in this paper and show how attraction to a target
region can be ensured by certain Lyapunov-like functions. The techniques used in the
proofs are variations of techniques well known in the literature (see, e.g., [19]). But to
provide clear and self-contained insight into the theoretical background of our method
we provide our own proofs instead of deriving the results as corollaries of results from
the literature (whose proofs would then be highly technical and lack insight).

For an ODE \( \dot{x} = f(x) \), where \( x \in \mathbb{R}^n \), we denote by \( x(\cdot, x_0) : \mathbb{R}_{\geq 0} \to \mathbb{R}^n \) a
trajectory of the differential equation starting from \( x_0 \).

The definition of stability that we will use allows us to explicitly specify a target
region and the basin of attraction.

**Definition 2.1.** Given an \( n \)-dimensional differential equation \( \dot{x} = f(x) \), and sets
\( U \) and \( TR \) such that \( TR \subset U \subset \mathbb{R}^n \), the differential equation is stable with respect to
\( U \) and the target region \( TR \) if for every point \( x_0 \in U \), the trajectory \( x(\cdot, x_0) \)

- will always stay in \( U \) (for all \( t \in \mathbb{R}_{\geq 0} \), \( x(t, x_0) \in U \), and
- will eventually reach \( TR \) (there is a \( t_1 \in \mathbb{R}_{\geq 0} \) such that \( x(t_1, x_0) \in TR \).

By allowing an explicit parameter \( U \) in this definition, one can explicitly specify a
desired basin of attraction. This helps to avoid situations where a differential equation
is stable, but where the found Lyapunov-like function proves attraction only within a
tiny region.

By allowing a target region instead of a single equilibrium point, the method can
also be applied in cases where no equilibrium exists (e.g., when we want to study
attraction to a limit cycle; cf. the (weaker) notion of practical stability [19]). In cases
where an equilibrium point exists, one can use as a target region in our method a
small basin of attraction computed from a Lyapunov function of the linearization of
the differential equation \([13, 10, 6]\).

In order to ensure this stability notion, we use the following adaption of the notion of a Lyapunov function.

**Definition 2.2.** For a given differential equation \(\dot{x} = f(x)\) with sets \(B\) and \(TR\) such that \(TR \subset B\), a continuously differentiable function \(V(x)\) is called a set Lyapunov function in \(B\) with respect to \(TR\) if and only if the constraint

\[
\forall x \in B \left[ x \notin TR \Rightarrow \frac{d}{dt} V(x) < 0 \right]
\]

holds. Here \(\Rightarrow\) denotes an implication symbol, and \(\frac{d}{dt} V(x) < 0\) denotes the time-derivative of \(V\) along \(f\), that is, \(\frac{\partial V}{\partial x} T f(x)\).

What exactly is guaranteed by a set Lyapunov function \(V\)? First of all, it ensures that sublevel sets of \(V\) in \(B\) are never left. Here, given a closed set \(B\) and a set Lyapunov function \(V\), we define an \(s\)-sublevel set of \(V\) in \(B\) to be \(\{ x \in B : V(x) < s \}\) and denote this set by \(V^B_{<s}\).

**Theorem 2.3.** The existence of a set Lyapunov function \(V(x)\) in a closed set \(B\) with respect to a target region \(TR\) guarantees that for every \(s\), every trajectory starting in a connected component \(C\) of the \(s\)-sublevel set \(V^B_{<s}\) that does not intersect the boundary of \(B\) will not leave \(C\) without reaching \(TR\). Moreover, if the closure of \(TR\) does not intersect the boundary of \(C\), the trajectory will not leave \(C\) at all.

**Proof.** For every element of the boundary \(\partial C\) of \(C\) that is not in \(TR\), \(\frac{d}{dt} V(x) < 0\). Thus, trajectories can leave \(C\) only through an element of \(\partial C\) that is in \(TR\). Such an element does not exist if the closure of \(TR\) does not intersect the boundary of \(C\).

In Figure 2.1 we illustrate the only possible way a trajectory can leave \(C\): if the closure of \(TR\) intersects the boundary of \(C\), then a trajectory may enter \(TR\), and then leave \(C\) within \(TR\), since \(V\) can be positive in \(TR\).

![Fig. 2.1. Necessity of condition in Theorem 2.3.](image)

Moreover, a set Lyapunov function ensures that the target region is eventually reached.

**Theorem 2.4.** The existence of a set Lyapunov function \(V(x)\) in a closed set \(B\) with respect to an open target region \(TR\) guarantees that every trajectory starting in a connected component \(C\) of the \(s\)-sublevel set \(V^B_{<s}\) that does not intersect the boundary of \(B\) will enter \(TR\).
Proof. Since $B$ is bounded and $B \setminus TR$ is closed, due to the continuity of $V$ and $\frac{d}{dt}V$, we know that $V$ is bounded in $B \setminus TR$, and that $\frac{d}{dt}V$ has a maximum $\epsilon$ in $B \setminus TR$. Obviously, this maximum $\epsilon$ is a negative real number.

Let $x_0$ be an arbitrary but fixed point in $C$. It is sufficient to consider only the case $x_0 \in C \setminus TR$.

We assume that for all $t \in \mathbb{R}_{\geq 0}$, $x(t, x_0) \notin TR$, and we derive a contradiction. From Theorem 2.3 and our assumption, for all $t \in \mathbb{R}_{\geq 0}$, $x(t, x_0) \in C \setminus TR$. Thus, for all $t \in \mathbb{R}_{\geq 0}$, $\frac{d}{dt}V \leq \epsilon$. Since $\epsilon$ is negative, this implies that as $t$ goes to infinity, $V(x(t, x_0))$ goes to minus infinity, contradicting the fact that $V$ is bounded in $B \setminus TR$.

Thus, there exists a $t \in \mathbb{R}_{\geq 0}$ such that $x(t, x_0) \in TR$. Note that without the requirement that the target region $TR$ be open, the boundary of $TR$ could form a limit cycle. In such a case, trajectories could come arbitrarily close to the target region but not enter it.

So our stability notion can be proved by computation of set Lyapunov functions as follows.

**Corollary 2.5.** The existence of a set Lyapunov function $V(x)$ in a closed set $B$ with respect to an open set $TR$ guarantees stability with respect to a connected component $C$ of a given $s$-sublevel set $V^B_{\leq s}$ provided that

- the component $C$ does not intersect the boundary of $B$, and
- the closure of $TR$ does not intersect the boundary of $C$.

Does the existence of a set Lyapunov function $V(x)$ guarantee that every trajectory starting in such a sublevel set will reach and eventually stay in $TR$? No! The reason is that it will not prohibit a trajectory from infinitely often entering $TR$, staying within $TR$ for a period of time, and then leaving $TR$ again. This intuition is illustrated in the two-dimensional case in Figure 2.2.

![Fig. 2.2. An example with a cycle.](image)

However, since $TR$ will usually be small, one can usually proceed with other techniques based on local reasoning: On the one hand, one can use a Lyapunov function of the linearized system to show asymptotic stability [13, 10]. On the other hand, one can use techniques to prove that an invariant set of the system has been reached [3].

The question remains of how to find for a given set Lyapunov function $V(x)$ an $s$-sublevel set to which one can apply Corollary 2.5 to arrive at a basin of attraction. One approach follows an analogy of a classical approach for computing the basin of attraction based on Lyapunov functions [6]: Minimize $V(x)$ on the boundary $\partial B$ of
B and use the sublevel set \( V_{\min_{x \in \partial B}}^{B} \). Another approach is to guess a value for \( s \) and check whether there is a connected component of \( V_{\min_{x \in \partial B}}^{B} \) that does not intersect \( \partial B \) (in section 7 we will illustrate this in an example by using the constraint solver RSolver). If the resulting basin of attraction is too small, one can iteratively choose larger values for \( s \), thereby enlarging the basin of attraction.

Note that—in the case where both \( f \) and \( V \) are polynomials—constraint (2.1) is a formula in the predicate logical theory of the real numbers with addition and multiplication [31]. Hence, in theory, as for classical Lyapunov functions, one could compute set Lyapunov functions using decision procedures for the theory of real-closed fields [44]. However, the current methods are not efficient enough to solve this problem in practice. Another approach would be to use an interval arithmetic–based branch-and-bound or branch-and-prune scheme [32, 36]. However, one can do even better, as will be shown in the next section.

3. Algorithm. In this section, we present a method for finding a polynomial set Lyapunov function, provided that \( f \) is a polynomial. Let \( V \) be a polynomial with parametric coefficients. Let \( \frac{dV}{df} \) be the time-derivative of \( V \) along \( f \), represented as a polynomial whose coefficients are linear combinations of the parameters that form the coefficients of \( V \). We substitute the resulting time-derivative \( \frac{dV}{df} \) into constraint (2.1) and denote the result by \( \Lambda_{f,V} \). Then we solve \( \Lambda_{f,V} \) for the parameters that form the coefficients of \( V \).

For solving the constraint \( \Lambda_{f,V} \) we use the convention that an interval occurring in a constraint represents a fresh, universally quantified variable ranging over that interval (e.g., \([0, 1]a \leq 1\) represents that for all \( v \in [0, 1] \), \( va \leq 1 \), where \( v \) is a new variable). Also, we assume that \( V \) is of the form \( \sum_{j=1}^{m} a_{j} x^{a_{j}} \) (the powers occurring in the tuple \( a_{j} \) are applied elementwise to the tuple of variables \( x \)). We view the resulting time-derivative \( \frac{dV}{df} \) as having the form \( \sum_{j=1}^{m} g_{j}(a_{1}, \ldots, a_{m})x^{a_{j}} \), where the \( a_{1}, \ldots, a_{m} \) occur linearly in the \( g_{j}(a_{1}, \ldots, a_{m}) \). So the constraint \( \Lambda_{f,V} \), resulting from the substitution of \( \frac{dV}{df} \) into constraint (2.1), has the form

\[
\forall x \in B \left[ x \notin TR \Rightarrow \sum_{j=1}^{l} g_{j}(a_{1}, \ldots, a_{m})x^{a_{j}} < 0 \right].
\]

To solve this constraint for \( a_{1}, \ldots, a_{m} \), we compute a constraint that is a relaxation of \( \Lambda_{f,V} \) in the sense that

- every solution of the relaxation is also a solution of the original constraint \( \Lambda_{f,V} \), and
- the relaxation is easier to solve than the original constraint.

A first approach for arriving at this relaxation is as follows:

1. Drop the constraint on the left-hand side of the implication sign of \( \Lambda_{f,V} \).
2. Replace every monomial \( x^{a_{j}} \) by an interval bounding its range over \( B \) (the result is a constraint of the form \( \sum_{i=1}^{l} l_{i} g_{i}(a_{1}, \ldots, a_{m})I_{i} < 0 \), where the \( I_{i} \)’s are intervals).
3. Rewrite \( \sum_{i=1}^{l} l_{i} g_{i}(a_{1}, \ldots, a_{m})I_{i} \) as an expression of the form \( \sum_{j=1}^{m} I_{j} a_{j} \) by distributing each interval \( I_{j} \) over the linear combination \( g_{j}(a_{1}, \ldots, a_{m}) \), collecting the coefficients of each \( a_{j} \), and computing a single interval for each such coefficient.

The interval calculations of steps 2 and 3 can be done using interval arithmetic. The resulting constraint overapproximates the original constraint \( \Lambda_{f,V} \) (resulting in
Algorithm 1 Computing a set Lyapunov function.

Input: a polynomial differential system $\dot{x} = f(x)$, sets $B$ and $TR$
Output: if the algorithm terminates, a set Lyapunov function with respect to $B$ and $TR$

1: choose a polynomial $V$ with parametric coefficients $a_1, \ldots, a_m$
2: let $\phi \leftarrow \bigwedge_{f,V}$
3: while relax($\phi$) does not have a solution do
4: branch a universal quantifier in $\phi$
5: end while
6: return $V$ with the solution of relax($\phi$) substituted for $a_1, \ldots, a_m$

a smaller set of $a_1, \ldots, a_m$ on which it holds). Moreover, it is a linear inequality with interval coefficients (i.e., a linear interval inequality), which can be solved exactly (to be shown later). However, each of the above steps introduces some overapproximation. Our algorithm will iteratively reduce this overapproximation as follows.

Consider the overapproximation introduced by step 2. Even if the bounds on the monomials are exact, this step loses the dependency between the different monomials. For reducing this problem, we rewrite the universal quantifier $\forall x \in B \phi$ of $\Lambda_{f,V}$ to a conjunction of the form $\forall x \in B_1 \phi \land \forall x \in B_2 \phi$, where $B_1 \cup B_2 = B$, and $B_1$ and $B_2$ nonoverlapping. We can apply the above relaxation process to every branch of the resulting constraint, again arriving at a system of linear interval inequalities that can be solved exactly (see the discussion below). We continue with this branching process until a solution can be found.

For reducing the overapproximation introduced by step 1, we observe that the above branching process results in smaller bounds for the universal quantifiers. This allows us, for some branches of the form $\forall x \in B' \left[ x \notin TR \Rightarrow \frac{d}{dt}V(x) < 0 \right]$, to prove $\forall x \in B'[x \in TR]$, which also proves the full branch. Hence we can drop the branch from the conjunction.

For a constraint $\phi$, we denote the result of the relaxation process, as described until now by relax($\phi$), and arrive at the branch-and-relax Algorithm 1.

Since every step for arriving at the relaxation is an overapproximation, we have the following property.

Property 1. Every solution $(a_1, \ldots, a_k)$ of relax($\phi$) is a solution of $\phi$ but not vice versa.

The correctness of Algorithm 1 follows from this property and the fact that branching is an equivalence transformation.

The system of linear interval inequalities formed by relax($\phi$) can be solved as follows: First, replace each strict inequality $< 0$ by a nonstrict inequality $\leq -\varepsilon$, with $\varepsilon$ positive but small. This introduces some overapproximation, but this overapproximation can easily be made arbitrarily small.

Now we can proceed using a method due to Rohn and Kreslová [40]. We can replace each variable $a$ that we want to solve for by a difference of two positive variables $a^1 - a^2$, and replace expressions of the form $I(a^1 - a^2)$ by $Ia^1 - Ia^2$. Since $a^1$ and $a^2$ are positive, and $I$ represents a universally quantified variable in an inequality, we can replace the interval $I = [I^-, I^+]$ by its bounds to arrive at $I^+a^1 - I^-a^2$. The result is a system of linear inequalities that can be solved by linear programming.

According to a proof by Rohn and Kreslová this procedure does not introduce overapproximation [40]. In other words, this procedure does not lose solutions of
the original system. To see this, take an inequality of the original system, say
\( \sum_{i \in \{1,...,n\}} I_i x_i \leq b \), and denote the resulting linear inequality by \( \sum_{i \in \{1,...,n\}} I_i^+ a_i^+ - I^- a_i^- \leq b \). Let \( x_1, ..., x_n \) be an arbitrary, but fixed, solution of the original interval
inequality. We get a solution of the resulting linear inequality as follows: Let
\( a_1^+, ..., a_n^+ \) be such that \( a_i^+ \) is equal to \( x_i \) if \( x_i \) is positive, and zero otherwise. Let
\( a_1, ..., a_n \) be such that \( a_i^- \) is equal to \( -x_i \) if \( x_i \) is negative, and zero otherwise. Then
\( I_i^+ a_i^+ - I^- a_i^- = r_i x_i \), where \( r_i \) is equal to \( I^+ \) if \( x_i \) is positive, and equal to \( I^- \) oth-
erwise. Hence \( r_i \in I_i \). Since intervals represent universally quantified variables, we have \( \sum r_i x_i \leq b \). As a result, we also have \( \sum_{i \in \{1,...,n\}} I_i a_i^+ - I^- a_i^- \leq b \).

4. Improvements. In this section we describe three improvements to the basic
algorithm described in the previous section.

4.1. Initial partition. Especially in higher dimensions, branching needs a lot
of time to separate the target region from the rest of the state space. Hence we start
the algorithm with an initial partition that fulfills this separation. In all our examples
we use a target region of the form \( \{ x \in B : |x_i - x_i^a| < \delta, 1 \leq i \leq n \} \), where \( x_i^a \) is the
equilibrium, and so we use a partition that consists of \( 1 + 2n \) elements in one of the
two following forms:
- \( |x_i, \overline{x}_i| \times \cdots \times |x_i, x_i^a - \delta| \times |x_i^a + \delta, x_i^a + 1| \times \cdots \times |x_n^a - \delta, x_n^a + \delta|, \)
- \( |x_i, \overline{x}_i| \times \cdots \times |x_i^a + \delta, \overline{x}_i| \times |x_i^a - \delta, x_i^a + \delta| \times \cdots \times |x_n^a - \delta, x_n^a + \delta|, \)
where \( i \in \{0, ..., n\} \) (see Figure 4.1 for a two-dimensional example).

![Fig. 4.1. Initial partitioning.](image)

Note that in the case \( i = 0 \) both forms coincide. Moreover, in this case the branch
is completely contained in the target region, and hence the branch holds trivially.

4.2. Additional constraints. For reducing the overapproximation introduced
by step 3 in section 3, instead of distributing the intervals over the linear combinations
\( g_i(a_1, ..., a_m) \), we introduce for each linear combination \( g_i(a_1, ..., a_m) \) a new variable
\( a'_i \), and relate it to the variables \( a_1, ..., a_m \) by adding the equation \( a'_i = g_i(a_1, ..., a_m) \).
In fact, we rewrite this equation as two inequalities \( a_i' \leq g_i(a_1, ..., a_m) \) and \(-a_i' \leq -g_i(a_1, ..., a_m) \) to again arrive at linear interval inequalities. Note that the new
constraints do not contain the state space variables; hence they only have to be added
once—and not for every branch.

For a constraint \( \phi \), we denote by \( \text{relax}^{-}(\phi) \) the result of the relaxation process
that uses the construction of the previous paragraph in step 3. It can be used in
Algorithm 1 instead of \( \text{relax}(\phi) \). This maintains the correctness of the algorithm due
to the following property:
Property 2. Every solution \((a_1, \ldots, a_m)\) of \(\text{relax}^\wedge(\phi)\) is a solution of \(\phi\).

Moreover, the new algorithm does not need more branching steps than the old one due to the next property.

Property 3. Every solution \((a_1, \ldots, a_m)\) of \(\text{relax}(\phi)\) is a solution of \(\text{relax}^\wedge(\phi)\).

Both properties follow from the fact that the constraint \(\text{relax}^\wedge(\phi)\) is equivalent to the constraint resulting from the first two relaxation steps in the computation of \(\text{relax}(\phi)\). Since the second and third relaxation steps are not an equivalence transformation (the distribution of an interval \(I_i\) over the linear combination \(g_i(a_1, \ldots, a_m)\) creates several copies of \(I_i\); this loses the dependency between them), the reverse implication of Property 3 does not hold. Hence, for some cases, the algorithm will already terminate when using \(\text{relax}^\wedge(\phi)\) but will have to branch further using \(\text{relax}(\phi)\).

4.3. Pre- and postprocessing based on linear algebra. The relaxation \(\text{relax}^\wedge(\phi)\) consist of two parts: first, a set of linear interval inequalities in the variables \(a'_1, \ldots, a'_l\), and second, a set of \(l\) linear equations of the form \(a'_i = g_i(a_1, \ldots, a_m)\). Note that \(l > m\), and hence an attempt to substitute a fixed solution \(a'_1, \ldots, a'_l\) of the first part (the linear interval inequalities) into the second part would result in a linear system with more equations than variables which, in general, would not be solvable.

However, we can eliminate the \(a_1, \ldots, a_m\) from the equations. The result is a system of \(l - m\) linear equations in the variables \(a'_1, \ldots, a'_l\). We can use this system of linear equations instead of original linear equations \(a'_i = g_i(a_1, \ldots, a_m)\), where \(i = 1, \ldots, l\). Since \(l\) will usually be only slightly larger than \(m\), the resulting overall constraint system will in general have much fewer equations than in the original form \(\text{relax}^\wedge(\phi)\). After solving this constraint, we can compute the \(a_1, \ldots, a_m\) from the \(a'_1, \ldots, a'_l\) in a postprocessing step.

5. Theoretical comparison and termination analysis. In this section we prove that in a certain, formally defined, sense the algorithm introduced in this paper is more efficient than earlier algorithms using an interval arithmetic–based branch-and-bound or branch-and-prune scheme [32, 36]. This will allow us to prove that the algorithm successfully terminates for all inputs that are nondegenerate in a sense to be defined below.

We start with describing a specialization of earlier interval arithmetic–based branch-and-bound or branch-and-prune algorithms [32, 36] to the special case of the constraint \(\Lambda_{f,V}\). Here one would search for solutions by gridding a compact area. For a given grid point \((a_1, \ldots, a_m)\), representing a certain choice for the parameters occurring in constraint \(\Lambda_{f,V}\), one checks—using interval arithmetic—whether for all values of \(x \in B\) the implication holds. Here, interval arithmetic computes an interval \([\underline{x}, \bar{x}]\) overapproximating the set \(\{\sum_{i=1}^l g_i(a_1, \ldots, a_m)x^{\gamma_i} \mid x \in B\}\). Then the constraint is determined to hold on the grid point \((a_1, \ldots, a_m)\) if either \(B \subseteq TR\) or \(\bar{x} < 0\). In order to reduce the overapproximation introduced by interval arithmetic, one iteratively divides the box \(B\) into smaller pieces and checks whether the above test holds on all resulting pieces.

Theorem 5.1. The number of branchings (loop iterations) of Algorithm 1, when using the improved relaxation method \(\text{relax}^\wedge()\), is smaller than the smallest number of splittings the interval method needs for proving that constraint \(\Lambda_{f,V}\) holds on any grid point, provided the same heuristics are used for branching and splitting.

Proof. Let \((a_1, \ldots, a_m)\) be an arbitrary but fixed grid point. Assume that the interval method can prove constraint \(\Lambda_{f,V}\) for this grid point after a splitting of the original box \(B\) into subboxes \(B_1, \ldots, B_k\). So we know that for all \(r \in \{1, \ldots, k\},\)
interval arithmetic proves
\[ \forall x \in B_r \left[ x \notin TR \Rightarrow \sum_{i=1}^{l} g_i(a_1, \ldots, a_m)x^{\gamma_i} < 0 \right], \]
which we denote by \( \phi_r \). We prove that \( (a_1, \ldots, a_m) \) can be extended to a solution of \( \text{relax}^{-}(\bigwedge_{r \in \{1, \ldots, k\}} \phi_r) \). Observe that the latter constraint is a conjunction of two types of constraints:

- constraints of the form \( \sum_{i=1}^{l} a_i f_i < 0 \), and
- constraints of the form \( a_i' = g_i(a_1, \ldots, a_m) \).

Let \( (a_1', \ldots, a_l') \) be the unique solution of the second type of constraints for \( (a_1, \ldots, a_m) \). We prove that \( (a_1', \ldots, a_l') \) is a solution of every constraint of the first type. Let \( \psi \) be an arbitrary but fixed constraint of the first type. This constraint \( \psi \) is the result of relaxing a certain \( \phi_r \), where \( r \in \{1, \ldots, k\} \). Interval arithmetic proves this \( \phi_r \). We know that \( B_r \subseteq TR \) does not hold, because otherwise relaxation would have dropped the constraint. Hence, interval evaluation of \( \sum_{i=1}^{l} g_i(a_1, \ldots, a_m)x^{\gamma_i} \) on \( B_r \) results in an interval that is strictly less than zero. By the choice of \( (a_1', \ldots, a_l') \), this \( \sum_{i=1}^{l} g_i(a_1, \ldots, a_m)x^{\gamma_i} \) is equal to \( \sum_{i=1}^{l} a_i' x^{\gamma_i} \). By the overapproximation property of interval arithmetic, and the semantics of our interval notation (fresh universally quantified variable) this proves that \( (a_1', \ldots, a_l') \) is a solution of \( \psi \).

An essential question for such an algorithm is whether it can find a solution in all cases where a solution exists. For algorithms based on approximation, one usually does not consider degenerate cases and requires only that a solution be found for all problems that are robust in the following sense [33, 36].

**Definition 5.2.** The value \( x \in \mathbb{R}^n \) is a robust solution of a constraint \( \phi \) if and only if there is an \( \varepsilon > 0 \) such that \( x \) is a solution of every constraint \( \phi' \) that results from \( \phi \) by perturbing some constants by not more than \( \varepsilon \) (i.e., \( \phi \) are the same with the exception of constants, and for every constant \( c \) in \( \phi \) and corresponding constant \( c' \) in \( \phi' \), \( |c - c'| \leq \varepsilon \)).

Of course, the success of the algorithm depends on the branching strategy used in step 4 of the algorithm. Under the natural assumption that the employed branching strategy lets the width of the bounds of the universal quantifiers go to zero, we can now prove success for robust solutions.

**Theorem 5.3.** If

- the constraint \( \Lambda_{f,V} \) has a robust solution, and
- Algorithm 1 employs a branching strategy such that for every \( \varepsilon > 0 \), there is an integer \( n \) such that after \( n \) branching steps the width of the largest bound of the universal quantifier is smaller than \( \varepsilon \),

then Algorithm 1, using the improved relaxation method \( \text{relax}^{-}(\cdot) \), successfully terminates with a solution.

**Proof.** Due to Lemma 2 of [36], an interval arithmetic–based branch-and-bound process using such a branching strategy succeeds in finding a solution for which the degree of truth [33] is positive. This precisely corresponds to robustness of the solution [33]. Since, due to Theorem 5.1, our algorithm using the improved relaxation method \( \text{relax}^{-}(\cdot) \) needs fewer iterations, it terminates under the above assumptions. \( \square \)

Note that, to the best of our knowledge, all complete heuristics for interval-based branch-and-bound methods discussed in the literature (e.g., [20, 9, 8, 34]) fulfill the requirements of the second item of Theorem 5.3. Hence, the theorem encompasses all sensible implementations of the algorithm.
However, since actual implementations usually do not use the abstract form of the algorithm (e.g., using rational number computation) but instead some floating-point approximation, additionally one would have to demand sufficient float-point precision. However, experience with similar branch-and-relax algorithms and with our own implementation of the algorithm (see section 7) has shown that, usually, the overapproximation introduced by employing floating-point computation is small compared to the overapproximation introduced by relaxation. So, in practice, the termination property also applies to practical implementations using the common 64 bit floating arithmetic.

6. Implementation. We implemented the method within the framework of our constraint solver RSolver [35, 32] which allows solving of quantified constraints using a branch-and-prune algorithm. This solver has a branching loop of the same form as Algorithm 1, but instead of using the relaxation technique described in this paper, it deduces information from the input constraints using a generalization of interval arithmetic called interval constraint propagation [2].

We simply added our relaxation technique to the branching loop of RSolver. As a result, we have an algorithm that can, in some cases, infer slightly more information from the input than Algorithm 1 due to its use of interval constraint propagation. We solve the resulting linear programs using the GNU linear programming kit (GLPK). A user worried by resulting rounding errors could easily add verification techniques [23, 17] or use a linear programming implementation based on rational number arithmetic.

We use the following heuristic for branching: Choose the widest box and bisect it into two boxes along the variable that has not been split for the most amount of time. This is a widely used, simple, and robust strategy that fulfills the requirements of Theorem 5.3. Depending on certain application areas, one could try to come up with more sophisticated strategies based on previous work from the literature (e.g., [20, 9, 8, 34]).

7. Examples. In this section, eleven examples will be presented for which we computed set Lyapunov functions by using the version of Algorithm 1 that includes all improvements described in section 4. Here the target region $TR$ is the set $\{x \in B : |x_i - x_i^*| < \delta, 1 \leq i \leq n\}$, where $B$ is a given box containing the equilibrium $x^*$, and $\delta > 0$ is a constant.

Although we solve a different problem than that solved by methods based on SOS decomposition [26, 27], we took all the examples that we found in the corresponding literature (unfortunately without precise run-times) and derived our Examples 1, 5, and 8 from them.

Example 1. This is a simplified model of a chemical oscillator [26]:

$$\begin{align*}
\dot{x}_1 &= 0.5 - x_1 + x_1^2 x_2, \\
\dot{x}_2 &= 0.5 - x_1^2 x_2.
\end{align*}$$

The equilibrium is $(1,0.5)$. Let $V(x_1, x_2) = ax_1^2 + bx_1 + cx_1 x_2 + dx_2 + cx_2^2 + f$; then $\frac{\partial}{\partial x_1} V(x_1, x_2) = 2ax_1 x_2 + (c - 2c)x_1^2 x_2 - cx_1^2 x_2 + bx_1 x_2 - dx_1 x_2^2 - 2ax_1^2 - cx_1 x_2 + (a - b + 0.5c)x_1 + (0.5c + e)x_2 + 0.5b + 0.5d$.

Choosing $B = [0.8, 1.2] \times [0.3, 0.7]$, $\delta = 0.01$, and $\varepsilon = 0.0001$, we get a set Lyapunov function $V(x_1, x_2) = x_1^2 - 30.1033994112x_1 - 72.5637083605x_2 + 19.8772784884x_2^2$. 

Example 2. This is a simplified model of a chemical oscillator [26]:

...
Example 2. This is the well-known van der Pol equation:
\[
\begin{cases}
\dot{x}_1 = -x_2, \\
\dot{x}_2 = x_1 - (1 - x_1^2)x_2.
\end{cases}
\]

Let \( V(x_1, x_2) = ax_1^2 + bx_1x_2 + cx_2^2 \); then \( \frac{d}{dt} V(x_1, x_2) = (-2a - b + 2c)x_1x_2 + (b - 2c)x_1^2 + 2bx_1^2x_2 + 2cx_2^2. \)

Choosing \( B = [-0.8, 0.8] \times [-0.8, 0.8] \), \( \delta = 0.1 \), and \( \epsilon = 0.0001 \), we get a set Lyapunov function \( V(x_1, x_2) = x_1^2 - 0.344917218515x_1x_2 + 0.858976611479x_2^2 \).

Example 3. This is an example taken from a survey on the estimation of stability regions [13]:
\[
\begin{cases}
\dot{x}_1 = -x_1 + x_2, \\
\dot{x}_2 = 0.1x_1 - 2x_2 - x_1^2 - 0.1x_1^3.
\end{cases}
\]

Let \( V(x_1, x_2) = ax_1^2 + bx_2^2 \); then \( \frac{d}{dt} V(x_1, x_2) = -2ax_1^2 + (2a + 2b)x_1x_2 - 4bx_2^2 - 2bx_1^2x_2 - 0.2bx_1^2x_2 \).

If we choose \( B = [-0.8, 0.8] \times [-0.8, 0.8] \), \( \delta = 0.1 \), and \( \epsilon = 0.0001 \), we get a set Lyapunov function \( V(x_1, x_2) = x_1^2 + 1.6512955643x_2^2 \).

Example 4. This is an example taken from a Chinese textbook on ODEs:
\[
\begin{cases}
\dot{x} = -4x^3 + 6x^2 - 2x, \\
\dot{y} = -2y.
\end{cases}
\]

Let \( V(x, y) = ax^4 + bx^3 + cx^2 + dy^2 \); then \( \frac{d}{dt} V(x, y) = -16ax^6 + (24a - 12b)x^5 + (8a + 18b - 8c)x^4 + (b - 12c + 4d)x^3 - 4ex^2 - 4dy^2 \).

If we choose \( B = [-0.4, 0.4] \times [-0.4, 0.4] \), \( \delta = 0.1 \), and \( \epsilon = 0.00001 \), we get a set Lyapunov function \( V(x, y) = x^4 + 0.571428571429x^3 + 0.285714285714x^2 + 1.52556785714y^2 \).

Example 5. This is an example taken from a paper [27] on computing Lyapunov functions using SOS decomposition:
\[
\begin{cases}
\dot{x} = -x + (1 + x)y, \\
\dot{y} = -(1 + x)x.
\end{cases}
\]

Let \( V(x, y) = ax^2 + byx + cy^2 + dy^3 + ex^3 + fx^2y^2 + gy^4 \); then \( \frac{d}{dt} V(x, y) = (-2a - b)x^2 + (2a - 2c)x_1x_2 + by^2 - bx_1^2x_2 + (2a - 2c)x_1x_2 + (b - 3d)x_2y + (b - 3d)x_2y + (4e - 2f)x_3y + (4e - 2f)x_3y + (4e - 2f)x_3y + (4e - 2f)x_3y + (4e - 2f)x_3y + (4e - 2f)x_3y + (4e - 2f)x_3y + (4e - 2f)x_3y \).

If we choose \( B = [-0.7, 0.9] \times [-0.7, 0.9] \), \( \delta = 0.1 \), and \( \epsilon = 0.0001 \), we get a set Lyapunov function \( V(x, y) = x^2 - 0.160613397902x_1 + 1.08030669895y^2 - 0.0535377993005y^3 + 0.0401533494754x^4 + 0.0803066989508x^2y^2 + 0.0401533494754y^4 \).

Example 6. This is a two-dimensional example with an equilibrium and a limit cycle:
\[
\begin{cases}
\dot{x}_1 = x_2 + (1 - x_1^2 - x_2^2)x_1, \\
\dot{x}_2 = -x_1 + (1 - x_1^2 - x_2^2)x_2.
\end{cases}
\]

For this example, the equilibrium \((0, 0)\) is an unstable focus, and the curve \( x_1^2 + x_2^2 = 1 \) is a stable limit cycle. Taking \( \delta = 1.2 \), the target region contains both the unstable equilibrium and the stable limit cycle.
Let \( V(x_1, x_2) = \frac{(ax_1^2 + bx_2^2)}{2} \); then 
\[
\frac{d}{dt} V(x_1, x_2) = (a - b)x_1x_2 + ax_1^2 + bx_2^2 - ax_1^4 - bx_2^4 - (a + b)x_1^2x_2^2.
\]
Choosing \( B = [-2, 2] \times [-2, 2] \) and \( \varepsilon = 0.0001 \), we get a set Lyapunov function 
\[
V(x_1, x_2) = x_1^2 + 0.91874633005x_2^2.
\]

**Example 7.** This is a three-dimensional example taken from [42]:
\[
\begin{align*}
\dot{x}_1 &= -x_2, \\
\dot{x}_2 &= -x_3, \\
\dot{x}_3 &= -x_1 - 2x_2 - x_3 + x_1^2.
\end{align*}
\]

Let \( V(x_1, x_2, x_3) = ax_1^2 + bx_2^2 + cx_3^2 + dx_1x_2 + ex_1x_3 + fx_2x_3; \) then 
\[
\frac{d}{dt} V(x_1, x_2, x_3) = (d + c)x_1^2 - 2fx_1^2 + (2c - f)x_2^2 + (2a - 2e - f)x_1x_2 + (-2c - d - c)x_1x_3 + (-2b - e - f)x_2x_3 + 2cx_1x_3 + cx_1^2 + fx_1^2.
\]
Choosing \( B = [-0.2, 0.2] \times [-0.2, 0.2] \times [-0.2, 0.2], \) \( \delta = 0.1 \), and \( \varepsilon = 0.0001 \), we get a set Lyapunov function 
\[
V(x_1, x_2, x_3) = x_1^2 + 0.477681371524x_2^2 + 0.96415690989x_3^2 - 0.88367562827x_1x_2 - 1.04167056951x_1x_3 + 0.089274513903x_2x_3.
\]

**Example 8.** This is an example taken from a Chinese textbook on ODEs:
\[
\begin{align*}
\dot{x} &= -x - 3y + 2z + yz, \\
y &= 3x - y - z + xz, \\
z &= -2x + y - z + xy.
\end{align*}
\]

Let \( V(x, y, z) = ax^2 + by^2 + cz^2; \) then 
\[
\frac{d}{dt} V(x, y, z) = -2ax^2 - 2by^2 - 2cz^2 + (-6a + 6b)xy + (4a - 4c)xz + (-2b + 2c)yz + (2a + 2b + 2c)xyz.
\]
If we choose \( B = [-0.4, 0.4] \times [-0.4, 0.4] \times [-0.4, 0.4], \) \( \delta = 0.1 \), and \( \varepsilon = 0.0001 \), we get a set Lyapunov function 
\[
V(x, y, z) = x^2 + y^2 + z^2.
\]

**Example 9.** This is an example with three equilibria and infinitely many limit cycles:
\[
\begin{align*}
\dot{x}_1 &= x_2 - x_3 + (1 - x_1^2 - x_2^2 - x_3^2)x_1, \\
\dot{x}_2 &= -x_1 + x_3 + (1 - x_1^2 - x_2^2 - x_3^2)x_2, \\
\dot{x}_3 &= x_1 - x_2 + (1 - x_1^2 - x_2^2 - x_3^2)x_3.
\end{align*}
\]
Clearly, \((0, 0, 0)\) is an unstable equilibrium. Moreover, \(\pm(\sqrt{3}/3, \sqrt{3}/3, \sqrt{3}/3)\) are also equilibria. Further, for any arbitrary but fixed constant \(c \in (-\sqrt{3}, \sqrt{3})\), the intersection of \( x_1 + x_2 + x_3 = c \) and \( x_1^2 + x_2^2 + x_3^2 = 1 \) is a cycle.

Let \( V(x_1, x_2, x_3) = (ax_1^2 + bx_2^2 + cx_3^2)/2; \) then 
\[
\frac{d}{dt} V(x_1, x_2, x_3) = (a - b)x_1x_2 + (c - a)x_1x_3 + (b - c)x_2x_3 + ax_1^2 + bx_2^2 + cx_3^2 - ax_1^4 - bx_2^4 - cx_3^4 - (a + b)x_1^2x_2^2 - (a + c)x_1^2x_3^2 - (b + c)x_2^2x_3^2.
\]
Choosing \( B = [-2, 2] \times [-2, 2] \times [-2, 2], \) \( \delta = 1.3 \), and \( \varepsilon = 0.0001 \), we get a set Lyapunov function 
\[
V(x_1, x_2, x_3) = x_1^2 + 0.910410691005x_2^2 + 0.910410691005x_3^2.
\]

**Example 10.** This is a four-dimensional example with an unstable equilibrium:
\[
\begin{align*}
\dot{x}_1 &= x_2 - x_3 + x_4 + (1 - x_1^2 - x_2^2 - x_3^2 - x_4^2)x_1, \\
\dot{x}_2 &= x_1 + x_3 - x_4 + (1 - x_1^2 - x_2^2 - x_3^2 - x_4^2)x_2, \\
\dot{x}_3 &= x_1 - x_2 + x_4 + (1 - x_1^2 - x_2^2 - x_3^2 - x_4^2)x_3, \\
\dot{x}_4 &= x_1 + x_2 - x_3 + (1 - x_1^2 - x_2^2 - x_3^2 - x_4^2)x_4.
\end{align*}
\]
Clearly, \((0,0,0,0)\) is an unstable equilibrium and there are no other equilibria. Moreover, the system has several cycles on the surface \(x_1^2 + x_2^2 + x_3^2 + x_4^2 = 1\). Let \(V(x_1, x_2, x_3, x_4) = (a x_1^2 + b x_2^2 + c x_3^2 + d x_4^2)/2;\) then \(\frac{d}{dt} V(x_1, x_2, x_3) = (a - b)x_1 x_2 + (c - a)x_3 x_4 + (d - a)x_5 x_4 + (b - c)x_1 x_4 + (c - d)x_3 x_4 + a x_1^2 + b x_2^2 + c x_3^2 + d x_4^2 - a x_1^2 - b x_2^2 - c x_3^2 - d x_4^2 - (a + b)x_1 x_2 - (a + c)x_1 x_3 - (a + d)x_1 x_4 - (b + c)x_2 x_3 - (b + d)x_2 x_4 - (c + d)x_3 x_4.

Choosing \(B = [-2,2] \times [-2,2] \times [-2,2] \times [-2,2]\), \(\delta = 1.6\), and \(\varepsilon = 0.0001\), we get a set Lyapunov function \(V(x_1, x_2, x_3, x_4) = 0.00015943877551(x_1^2 + x_2^2 + x_3^2 + x_4^2)\).

**Example 11.** This is a six-dimensional system taken from [26]:

\[
\begin{align*}
\dot{x}_1 &= -x_1^3 + 4x_2^3 - 6x_3x_4, \\
\dot{x}_2 &= -x_1 - x_2 + x_3^2, \\
\dot{x}_3 &= x_1x_4 - x_3 + x_4x_6, \\
\dot{x}_4 &= x_1x_3 + x_3x_6 - x_4^3, \\
\dot{x}_5 &= -2x_5^3 - x_5 + x_6, \\
\dot{x}_6 &= -3x_3x_4 - x_5^3 - x_6.
\end{align*}
\]

Let \(V(x_1, \ldots, x_6) = a x_1^2 + b x_2^2 + c x_3^2 + d x_4^2 + e x_5^2 + f x_6^2;\) then \(\frac{d}{dt} V(x_1, \ldots, x_6) = -2ax_1^4 - 4bx_2^4 - 2cx_3^4 - 2dx_4^4 - 4ex_5^4 - 2fx_6^4 - (8a - b)x_1x_2^2 - (8a - c)x_1x_3^2 - (8a - d)x_1x_4^2 - (8a - e)x_1x_5^2 - (8a - f)x_1x_6^2 + (8a - b)x_2x_3^2 + (8a - c)x_2x_4^2 + (8a - d)x_2x_5^2 + (8a - e)x_2x_6^2 + (8a - b)x_3x_4^2 + (8a - c)x_3x_5^2 + (8a - d)x_3x_6^2 + (8a - e)x_4x_5^2 + (8a - f)x_4x_6^2 + (8a - e)x_5x_6^2 + (8a - f)x_5x_6^2 + (8a - d)x_5x_6^2 + (8a - f)x_5x_6^2.

Choosing \(B = [-0.8, 0.8] \times \cdots \times [-0.8, 0.8]\), \(\delta = 0.1\), and \(\varepsilon = 0.0001\), we can get a set Lyapunov function \(V(x_1, x_2, x_3, x_4, x_5, x_6) = x_1^2 + 2x_2^2 + 5.5x_3^2 + 0.5x_4^2 + x_5^3 + 2x_6^2\).

The computations were performed on an IBM notebook of Pentium IV, 1.70 GHz with 1 GB RAM, and they were cancelled in cases when computation did not terminate before 8 hours of computation time. The computing times and the number of branching steps are listed in Table 7.1 (basic algorithm, relaxation \(relax(\phi)\)), Table 7.2 (algorithm with initial partition), Table 7.3 (algorithm with initial partition and improved relaxation \(relax^- (\phi)\)), and Table 7.4 (algorithm with initial partition and linear algebra).

<table>
<thead>
<tr>
<th>Example</th>
<th>CPU time</th>
<th>Branching steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.92s</td>
<td>49</td>
</tr>
<tr>
<td>2</td>
<td>&gt; 8 hours</td>
<td>unknown</td>
</tr>
<tr>
<td>3</td>
<td>222.74s</td>
<td>1550</td>
</tr>
<tr>
<td>4</td>
<td>&gt; 8 hours</td>
<td>unknown</td>
</tr>
<tr>
<td>5</td>
<td>&gt; 8 hours</td>
<td>unknown</td>
</tr>
<tr>
<td>6</td>
<td>109.41s</td>
<td>824</td>
</tr>
<tr>
<td>7</td>
<td>97821.6s</td>
<td>4041</td>
</tr>
<tr>
<td>8</td>
<td>42.26s</td>
<td>507</td>
</tr>
<tr>
<td>9</td>
<td>&gt; 8 hours</td>
<td>unknown</td>
</tr>
<tr>
<td>10</td>
<td>&gt; 8 hours</td>
<td>unknown</td>
</tr>
<tr>
<td>11</td>
<td>&gt; 8 hours</td>
<td>unknown</td>
</tr>
</tbody>
</table>

The timings clearly show that, with minor anomalies due to incidental influences of the branching heuristics, the improvements of section 4 really improve the algorithm.

In order to illustrate how to arrive at a basin of attraction from a computed set Lyapunov function, we used RSolver to compute a verified inner and outer

BASIN OF ATTRACTION BY LYAPUNOV-LIKE FUNCTIONS
approximation of the level set corresponding to 0.03 for the set Lyapunov function computed for Example 4. The result can be seen in Figure 7.1, where the whole figure represents the box $B$. Since the outer approximation of the level set is a strict subset of the box $B$ used for computing the set Lyapunov function, all elements of the inner approximation are elements of the basin of attraction. Note that RSolver allows the user to arbitrarily decrease the difference between the inner and outer approximation.

8. Related work. We are aware of only one method that can compute closed-form Lyapunov functions of nonlinear ODEs in a completely automatic way. This method is based on SOS decomposition using relaxation to linear matrix inequalities.
However, it tries to prove classical stability. This has the advantage of being in correspondence with classical research and techniques in stability analysis. But it has two drawbacks compared to our method:

- The resulting Lyapunov function only characterizes the behavior of the given system locally (around the equilibrium) or globally (on the whole state space). However, in applications one usually wants a characterization of the behavior in a subset of the whole state space, as provided by the set $B$ in Definition 2.2 (note, however, that SOS can be used to compute a region of attraction with respect to a given Lyapunov function [27, section 7.3]).

- The method is not applicable in the case where no equilibrium point exists, for example, in the case of studying attraction to a limit cycle. The method studied in this paper is still applicable in such a case, since one can freely choose the target region provided by the set $TR$ in Definition 2.2.

Moreover, the efficiency of Algorithm 1 can be arbitrarily decreased and increased by changing the size of the sets $(B \setminus TR)$, and so it can also be made both arbitrarily slower and arbitrarily faster than any other method.

Hence the efficiency of our method is not directly comparable with the efficiency of SOS methods. Still, our results for benchmark examples that we took from the literature on SOS methods (Examples 1, 5, and 8) show that the set Lyapunov functions that result from our method usually have a form similar to the classical Lyapunov functions computed by SOS techniques.

To the best of our knowledge, all other techniques for computing Lyapunov functions for nonlinear systems either require manual intervention, and hence are not fully automatic, or they produce results that are correct only up to discretization errors.

Methods requiring manual intervention are as follows:

- A method that uses Gröbner bases to choose the parameters in Lyapunov functions in an optimal way [12]. This requires the computation of a Gröbner basis for an ideal with a large number of variables. The user has to manually distinguish critical points from optima.

- A method that computes Lyapunov functions in the form of continuous piecewise affine functions [15] based on user-provided bounds on second-order derivatives.
• Methods that compute piecewise linear Lyapunov function based on some user-provided polygonal system characterization [24, 25].

Methods that use approximate discretizations are as follows:
• A method based on approximation by radial basis functions [14].
• A method based on linear programming [18].

Methods for computing the region of attraction can be roughly divided into two classes: one class maximizes the size of a region of attraction for a certain given Lyapunov function [42]; another class uses approximation techniques that try to enlarge a small initial region of attraction [13].

9. Conclusion. In this paper we have provided a method for computing the basin of attraction to a target region. The method is based on computation of Lyapunov-like functions using a branch-and-relax constraint solving algorithm. It seems that similar constraints have to be solved in many other areas (e.g., proving the termination of term-write systems, computation of barrier certificates [29], invariant generation [41, 38, 22, 39, 30], and analysis of finite element methods (FEM) [43]), and it is interesting work to apply our algorithms in these areas.

The presentation in this paper is restricted to polynomial ODEs and polynomial Lyapunov-like functions. However, one of the tools used in the algorithm, interval arithmetic, also works for expressions that contain function symbols such as sin, cos, exp. In fact, in many such cases, our relaxation technique—when applied manually—also results in linear interval inequalities, allowing for an application of the overall algorithm. However, the classification of the cases for which this can be automatized depends on the (potentially very complicated) symbolic manipulation techniques one is willing to use in the algorithm. An exploration of this issue would require a significant amount of additional work.

We will further increase the efficiency of our method, for example, by improving the branching heuristics used, and we will generalize our results to the stability analysis of hybrid systems [28, 5].

REFERENCES


Chapter 9

Constraints for Continuous Reachability in the Verification of Hybrid Systems

This chapter appeared as:

Constraints for Continuous Reachability in the Verification of Hybrid Systems

Stefan Ratschan\textsuperscript{1} and Zhikun She\textsuperscript{2}

\textsuperscript{1} Institute of Computer Science, Czech Academy of Sciences, Prague, Czech Republic
\textsuperscript{2} Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. The method for verification of hybrid systems by constraint propagation based abstraction refinement that we introduced in an earlier paper is based on an over-approximation of continuous reachability information of ordinary differential equations using constraints that do not contain differentiation symbols. The method uses an interval constraint propagation based solver to solve these constraints. This has the advantage that—without complicated algorithmic changes—the method can be improved by just changing these constraints. In this paper, we discuss various possibilities of such changes, we prove some properties about the amount of over-approximations introduced by the new constraints, and provide some timings that document the resulting improvement.

1 Introduction

A hybrid system is a dynamical system that involves both continuous and discrete state and evolution. This can, for example, be used for modeling the behavior of an embedded (digital) computing device influencing its (continuous) environment. An important task is to verify that a given hybrid system is safe, that is, every trajectory starting from an initial state never reaches an unsafe state.

In this paper we study constraints that can be used for modeling the continuous flow in the safety verification of hybrid systems by constraint propagation based abstraction refinement \cite{14,16}. Especially, we exploit the fact that the underlying solver, which is based on interval constraint propagation as introduced within the field of artificial intelligence \cite{9,3,5,13}, allows the use of a rich language of constraints that includes function symbols such as \texttt{sin}, \texttt{cos}, \texttt{exp}. These symbols arise naturally as solutions of linear differential equations.

More specifically, in this paper we study two types of constraints that model the reachability problem: one, for linear differential equations, is related to the explicit solution of such equations; and the second, for general differential constraints, is based on the mean-value theorem. Both constraints are quite simple

\textsuperscript{*} This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS). See \url{www.avacs.org} for more information.
to derive and similar ones have been used before in the literature. However, here we study in detail, how they behave when using interval based constraint propagation techniques for solving them, and how this behavior reflects in our method of verification of hybrid systems using constraint propagation based abstraction refinement. Our verification software HSOLVER [15], allows the user to experiment with these, and additional, user-defined, constraints.

Regarding additional related work, the approach by Hickey and Wittenberg [7] puts the level of modelling even higher, by employing a constraint logic programming language [8] that directly can deal with differential equations. Internally it solves constraints by transforming them into polynomial constraints using Taylor expansion, and then solves these using a similar solver as ours. The approach does not provide a comparison of different formulations of these constraints, and does not employ abstraction refinement to concentrate on refining the solution which is relevant for a given safety verification problem.

Tiwari [19] derives simple polynomial constraints from linear differential equations by manually doing an over-approximating quantifier elimination on a similar constraint as one of the constraints employed here. In contrast to that, since our solver can handle function symbols such as sin, cos, and exp, we can directly work on the original constraint, and—using an abstraction refinement scheme—approximate it arbitrarily closely.

Anai and Weispfenning [1] provide a classification of the cases when the time variable can be symbolically eliminated from the solution of linear differential equations (which may contain transcendental function symbols).

Similar constraints as employed here, which are based on the mean-value theorem or Taylor expansion, are ubiquitous in the integration of ODE’s.

The content of the paper is as follows: In Section 2 we review our method of verification using abstraction refinement; in Section 3 we discuss constraints for modeling reachability of differential equations; in Section 4 we discuss how we solve these constraints; in Section 5 we compare the constraints theoretically; in Section 6 we study empirically, how the constraints behave within our method; and in Section 7 we conclude the paper.

2 Constraint Propagation Based Abstraction Refinement

In this section, we review our previous approach [16, 14] for verifying safety using constraint propagation based abstraction refinement. We fix a variable \( s \) ranging over a finite set of discrete modes \( \{s_1, \ldots, s_n\} \) and variables \( x_1, \ldots, x_k \) ranging over closed real intervals \( I_1, \ldots, I_k \). We denote by \( \Phi \) the resulting state space \( \{s_1, \ldots, s_n\} \times I_1 \times \cdots \times I_k \). In addition, for denoting the derivatives of \( x_1, \ldots, x_k \) we assume variables \( \dot{x}_1, \ldots, \dot{x}_k \), ranging over \( \mathbb{R} \) each, and for denoting the targets of jumps, variables \( s', x_1', \ldots, x'_k \) ranging over \( \{s_1, \ldots, s_n\} \) and \( I_1, \ldots, I_k \), correspondingly. In the following we will use boldface to denote vectors of variables.

For describing hybrid systems, we use a flow constraint \( Flow(s, x, \dot{x}) \), a jump constraint \( Jump(s, x, s', x') \), an initial constraint \( Init(s, x) \) and an unsafety constraint \( UnSafe(s, x) \). Now, assuming a given hybrid system \( H = (Flow, Jump, \ldots) \), we...
Init,Unsafe), our safety verification problem is to verify that $H$ is safe, that is, there is no piecewise continuous trajectory of $H$ that starts from an initial state (i.e., a state satisfying the constraint $\text{Init}(s, x)$ and reaches an unsafe state (i.e., a state satisfying the constraint $\text{UnSafe}(s, x)$). Here the continuous parts of the trajectory are required to fulfill the flow constraint $\text{Flow}(s, x, \dot{x})$, and the discontinuous jumps are required to fulfill $\text{Jump}(s, x, s', x')$, where $(s, x)$ refers to the state before the jump and $(s', x')$ refers to the state directly after the jump. For formal definitions refer to our previous publication [16].

Our approach decomposes the state space $\Phi$ using hyper-rectangles (boxes) into finitely many mode/box pairs and then computes a finite over-approximation of the hybrid systems (an abstraction). In detail, for each pair of boxes, it sets an abstract transition, only if it cannot prove the absence of trajectories between them; also, it marks boxes as initial (unsafe), if it cannot prove the absence of an initial (unsafe) element in the box. If the resulting finite abstraction is safe, the hybrid system is also safe, since the abstraction over-approximates the hybrid system. If it is not safe, we refine the abstraction by splitting boxes into pieces and recomputing the affected information in the abstraction.

Moreover, we have a mechanism for removing unreachable elements from boxes. For this observe that a point in a box $B$ is reachable only if it is reachable either from the initial set via a flow in $B$, from a jump via a flow in $B$, or from a neighboring box via a flow in $B$. So we formulate constraints corresponding to each of these conditions and then remove points from boxes that do not fulfill at least one of these constraints.

The approach can be used with any constraint describing that $y$ can be reachable from $x$ via a flow in $B$ and mode $s$, for example, the one introduced in our previous publication and the new ones that will be introduced in the latter sections. We denote the used constraint by $\text{Reachable}_B(s, x, y)$.

Thus the above three possibilities of reachability allow us to formulate the following theorem:

**Theorem 1.** For a set of abstract states $\mathcal{B}$ such that all boxes corresponding to the same mode are non-overlapping, a pair $(s', B') \in \mathcal{B}$ and a point $z \in B'$, if $(s', z)$ is reachable, then

$$\text{initflow}_{B'}(s', z) \lor \bigvee_{(s, B) \in \mathcal{B}} \text{jumpflow}_{B, B'}(s, s', z)$$

$$\lor \bigvee_{(s, B) \in \mathcal{B}, s = s', B \neq B'} \text{boundaryflow}_{B, B'}(s', z)$$

where $\text{initflow}_{B'}(s', z)$, $\text{jumpflow}_{B, B'}(s, s', z)$ and $\text{boundaryflow}_{B, B'}(s', z)$ denote the following three constraints, respectively:

- $\exists y \in B' [\text{Init}(s', y) \land \text{Reachable}_{B'}(s', y, z)]$,
- $\exists \bar{x} \in B \exists x' \in B' [\text{Jump}(s, x, s', x') \land \text{Reachable}_{B'}(s', x', z)]$,
- $\exists x \in B \cap B' [\forall F \text{ of } B [x \in F \Rightarrow \text{out}_{s', B, B'}(F, x)] \land \text{Reachable}_{B'}(s', x, z)]$.

Here, $\text{out}_{s', B, B'}(x)$ is one of the following constraints:
\[
\exists j \in \text{Flow}(s', x, (\dot{x}_1, \ldots, \dot{x}_k)) \land \dot{x}_j \leq 0, \text{ if } F \text{ is the } j\text{-th lower face of } B, \text{ and }
\exists j \in \text{Flow}(s', x, (\dot{x}_1, \ldots, \dot{x}_k)) \land \dot{x}_j \geq 0, \text{ if } F \text{ is the } j\text{-th upper face of } B.
\]

We denote the main constraint (i.e., the disjunction) by \(\text{reachable}_{B'}(s', z)\). If we can prove that a certain point does not fulfill this constraint, we know that it is not reachable. In Section 4 we describe pruning algorithm that takes such a constraint and an abstract state \((s', B')\), and returns a sub-box of \(B'\) that still contains all the solutions of the constraint in \(B'\).

Since the constraint \(\text{reachable}_{B'}(s', z)\) depends on all current abstract states, a change of \(B'\) might allow further pruning of other abstract states. So we can repeat pruning until a fixpoint is reached. This terminates since we use floating point computation here and there are only finitely many floating point numbers.

We remove the initial mark of an abstract state \((s', B')\) if we can disprove \(\text{initflow}_{B'}(s', z)\) in Theorem 1, and we remove the unsafe mark of an abstract state \((s', B')\) if we can disprove \(\exists x \in B' \text{ Unsafe}(s', x)\). Moreover, we remove a transition from \((s, B)\) to \((s', B')\) if we can disprove both \(\text{boundaryflow}_{B, B'}(s', z)\) and \(\text{jumpflow}_{B, B'}(s, s', z)\) from Theorem 1. It is easy to show that the resulting system is an abstraction of the original hybrid system.

### 3 Constraints for Reachability

Assume that the flow constraint contains a differential equation of the form \(\dot{x} = Ax\), where \(A \in \mathbb{Q}^{k \times k}\). Differential equations of the form \(\dot{x} = Ax + B\) can be reduced to that form by shifting the equilibrium, provided that the equation \(Ax + B = 0\) has solutions. Given an initial set \(\text{Init}\), we have the exact solution \(x(t) = e^{At}x_0\), where \(x_0 \in \text{Init}\) and \(e^{At}\) is defined by \(\sum_{k=0}^{\infty} \frac{A^k}{k!}\). Thus, if \(x\) is reachable, then the constraint \(\exists t \in \mathbb{R}_{\geq 0} \exists x_0 \in \mathbb{R}^k [x_0 \in \text{Init} \land x = e^{At}x_0]\) holds.

Since the matrix \(A\) appears in an exponent, it is difficult to directly solve this constraint. We use another constraint introduced by Tiwari [19], that over-approximates the reach set, that can be easily computed from the matrix \(A\), and that does not contain matrix exponentiation. For this we re-express the real and complex eigenvalues of \(A^T\) (the transpose of \(A\)) using the following two sets:

\[
A_1 = \{ \lambda \in \mathbb{R} | \exists c \in \mathbb{R}^k [c \neq 0 \land A^T c = \lambda c] \};
\]

\[
A_2 = \{ (a, b) \in \mathbb{R} \times \mathbb{R}_{>0} | \exists c \in \mathbb{R}^k [c \neq 0 \land ((A^T)^2 - 2aA^T + (a^2 + b^2)I)c = 0] \}.
\]

For every \(\lambda \in A_1\), let \(c(1, \lambda)\) be an orthonormal basis of \(\{c : A^T c = \lambda c\}\); for every \((a, b) \in A_2\), let \(c(2, (a, b))\) be an orthonormal basis of \(\{c : ((A^T)^2 - 2aA^T + (a^2 + b^2)I)c = 0\}\). Then we can describe an over-approximation of the set of reachable states as follows.

**Lemma 1.** For a differential equation \(\dot{x} = Ax\) and a box \(B \subseteq \mathbb{R}^k\), if there is a trajectory in \(B\) from a point \(x = (x_1, \ldots, x_k)^T \in B\) to a point \(y = (y_1, \ldots, y_k)^T \in B\) on which \(\dot{x} = Ax\) holds, then

\[
\exists t \in \mathbb{R}_{\geq 0}[\text{eigen}^*_{A, B}(t, x, y)],
\]
where \(eigen^*_{A,B}(t,x,y)\) denotes

\[
\left[ \bigwedge_{\lambda \in \Lambda_1} \left[ \bigwedge_{c \in c(1,\lambda)} c^T y = c^T xe^{\lambda t} \right] \right] \land \left[ \bigwedge_{(a,b) \in \Lambda_2} \exists \dot{x} \in \mathbb{R}^k \right]
\]

\[
\left[ \dot{x} = Ax \land \bigwedge_{c \in c(2,(a,b))} c^T y = e^{at}c^T \left( x \cos(bt) + \frac{\dot{x} - xa}{b} \sin(bt) \right) \right]
\]

This expression is a formula in the first-order predicate language over the real numbers—it does not contain higher-order expressions such as derivatives (\(\dot{x}\) does not denote the derivative of \(x\) but simply a new variable). However, the corresponding restriction to the first-order theory of the reals is still undecidable, since we have to deal with function symbols like \(\sin\). However, there are over-approximating constraint solvers, that can be used (see Section 4).

We denote the above Constraint 1 by \(eigen_{A,B}(x,y)\). Note that if \(A\) has \(k\) different real eigenvalues and \(B = \mathbb{R}^k\), then this constraint describes the exact solution of the differential equation.

Now we will describe a constraint describing the reachability not only for linear differential equations, but for much more general descriptions of continuous evolution. We assume that the continuous dynamics is defined by a differential constraint \(D(x, \dot{x})\) (or short, \(D\)) which can be an arbitrary first-order formula in the theory of the reals over the tuples of variables \(x\) and \(\dot{x}\). This includes explicit and implicit differential equations and inequalities, and even differential-algebraic equations and inequalities.

Earlier [16] we used a constraint \(flow_{D,B}(x,y)\) describing the reachability in boxes as follows:

**Lemma 2.** For a differential constraint \(D(x, \dot{x})\) and a box \(B \subseteq \mathbb{R}^k\), if there is a trajectory in \(B\) from a point \(x = (x_1,\ldots,x_k)^T \in B\) to a point \(y = (y_1,\ldots,y_k)^T \in B\) such that for every point \(u\) on the trajectory and its derivative \(\dot{u}\), the pair \((u, \dot{u})\) satisfies \(D(x, \dot{x})\), then

\[
\left\{ \begin{array}{l}
\exists a_1,\ldots,a_k, \dot{a}_1,\ldots,\dot{a}_k[(a_1,\ldots,a_k) \in B \land

D((a_1,\ldots,a_k),(\dot{a}_1,\ldots,\dot{a}_k)) \land \dot{a}_n \cdot (y_m - x_m) = \dot{a}_m \cdot (y_n - x_n)]
\end{array} \right.
\]

Observe that whenever a given pair of points \((x, y)\) fulfills the above constraint \(flow_{D,B}(x,y)\)—indicating that there is a possible flow from \(x\) to \(y\)—then also the flipped pair \((y, x)\) fulfills the constraint. That is, the constraint does not distinguish time flowing forward, and time flowing backward. In order to avoid this loss of information we use the mean value theorem to formulate the following constraint:

**Lemma 3.** For a differential constraint \(D(x, \dot{x})\) and a box \(B \subseteq \mathbb{R}^k\), if there is a trajectory in \(B\) from a point \(x = (x_1,\ldots,x_k)^T \in B\) to a point \(y = (y_1,\ldots,y_k)^T \in B\) such that for every point \(u\) on the trajectory and its derivative \(\dot{u}\), the pair \((u, \dot{u})\) satisfies \(D(x, \dot{x})\), then
\[ \exists t \in \mathbb{R}_{\geq 0} [\text{flow}^*_D,B(t, x, y)], \quad (2) \]

where \( \text{flow}^*_D,B(t, x, y) \) denotes

\[
\bigwedge_{1 \leq i \leq k} \exists a_1, \ldots, a_k, \dot{a}_1, \ldots, \dot{a}_k [(a_1, \ldots, a_k) \in B \\
\wedge D((a_1, \ldots, a_k), (\dot{a}_1, \ldots, \dot{a}_k)) \wedge y = x_i + \dot{a}_i \cdot t]
\]

We denote the above Constraint 2 by \( \text{flow}'_{D,B}(x, y) \). Although both the constraint \( \text{flow}_{D,B}(x, y) \) and the constraint \( \text{flow}'_{D,B}(x, y) \) are quite simple, it is not at all clear, whether \( \text{flow}'_{D,B}(x, y) \) really allows to derive tighter reach set information than \( \text{flow}_{D,B}(x, y) \).

4 Solving the Constraints

We solve the constraints using our constraint solver RSolver [12], which implements interval constraint propagation techniques [5, 3, 9, 13]. These techniques can, for a given constraint and intervals for all its variables, contract these intervals to smaller ones, without losing any solutions. We illustrate the idea on an example: Given the constraint \( x^2 - 1 \leq 0 \), and the interval \([-2, 2]\) for \( x \), the method first decomposes this constraint into a conjunction of so-called primitive constraints, arriving at \( x^2 = t_0 \wedge t_0 - 1 = t_1 \wedge t_1 \leq 0 \). Here \( t_0 \) and \( t_1 \) are new, auxiliary variables. Then it takes the interval \([-\infty, +\infty]\) for all auxiliary variables and tries to contract all intervals wrt. the primitive constraints: using \( x^2 = t_0 \) we can contract the interval \([-\infty, \infty]\) for \( t_0 \) to \([0, 4]\), using \( t_0 - 1 = t_1 \) contract \([-\infty, \infty]\) for \( t_1 \) to \([-1, 3]\), using \( t_1 \leq 0 \) contract \([-1, 3]\) for \( t_1 \) to \([-1, 0]\), using \( t_0 - 1 = t_1 \) contract \([0, 4]\) for \( t_0 \) to \([0, 1]\), and using \( x^2 = t_0 \) contract \([-2, 2]\) for \( x \) to \([-1, 1]\). This process continues until a fixpoint is reached, which will always happen eventually, due to the finiteness of floating point numbers. We call the resulting algorithm a pruning function and, given a constraint \( \phi \) and a box \( B \), we denote the result of applying this function to \( \phi \) and \( B \) by \( \text{Prune}(\phi, B) \). In computer implementations the resulting intervals will enclose the solution sets of the primitive constraints up to rounding to the next floating point number. For the theoretical analysis in this paper we will ignore this rounding, and assume that these intervals are the tightest possible enclosures using real-number endpoints.

**Definition 1.** Given a constraint \( \phi \), pruning is optimal for \( \phi \) iff for all boxes \( B \), \( \text{Prune}(\phi, B) \) is the smallest box containing all solutions of \( \phi \) in \( B \).

Although the pruning function will contract optimally for primitive constraints, this will in general not be the case for more complex constraints. However, due to a classical result of interval arithmetic, we have (ignoring floating-point rounding):

**Property 1.** For every constraint \( \phi \), that contains every variable just once, pruning is optimal.
There are special techniques for handling disjunctions and quantifiers [13]. Moreover, there are various optimizations and extensions of the techniques discussed above. Most of them spend additional time to deal with the problem that pruning is in general not optimal if some variables occur more than once. Since our constraints usually only have few occurrences of the same variables, and since in our abstraction refinement approach (see Section 2) we will do thousands of prunings, it does not seem promising to use such optimizations here.

5 Theoretical Evaluation of the Constraints

In this section we will do a theoretical comparison of the constraints $flow_{D,B}(x, y)$, $flow'_{D,B}(x, y)$ and $eigen_{A,B}(x, y)$. Moreover, based on the gained insight, we will introduce a new constraint that combines their advantages.

We start with comparing $flow'_{D,B}(x, y)$ with $flow_{D,B}(x, y)$. Let us first discuss the size of the produced constraints. For $k$ dimensions, $flow_{D,B}(x, y)$ has $k(k-1)/2$ conjuncts, whereas $flow'_{D,B}(x, y)$ has just $k$. So, for dimensions larger than 2, $flow'_{D,B}(x, y)$ is smaller, and its size increases only linearly instead of quadratically. Hence pruning will take less time on $flow'_{D,B}(x, y)$, especially for high dimensions.

Let us now compare the effectiveness of the two constraints. For the one-dimensional case, $flow_{D,B}(x, y)$ reduces to a conjunction with zero conjuncts, that is, to a constraint that is trivially true. So in that case, $flow'_{D,B}(x, y)$ is definitely better. For higher dimensions, the relationship between the two constraints is more complicated. Therefore, we will first study the relationship between $flow_{D,B}(x, y)$ and $flow'_{D,B}(x, y)$ themselves, and then between the result of applying the pruning function to them.

Here we will use the following notation: Given two constraints $\phi_{S,B}(x, y)$ and $\psi_{S,B}(x, y)$, where $S$ is a differential constraint $D$ or a matrix $A$, and $B$ is a box, we will write $\phi_{S,B} \leq \psi_{S,B}$ ($\phi_{S,B} \equiv \psi_{S,B}$) iff for all $S$ and all $B$, the solution set of $\phi_{S,B}$ in $B \times B$ is a subset of (equal to) the solution set of $\psi_{S,B}$ in $B \times B$. Analogously, we will write $\phi_{S,B} \leq_P \psi_{S,B}$ ($\phi_{S,B} \equiv_P \psi_{S,B}$) iff for all $S$, all $B$ and all $B_0$, where $B_0$ is a sub-box of $B$, $Prune(\phi_{S,B}(x, y), B_0 \times B)$ is a subset of (equal to) $Prune(\psi_{S,B}(x, y), B_0 \times B)$. Note that here we restrict $B_0$ to be a subset of $B$ because we use the constraints always in such a context.

There is no clearcut relationship between $flow_{D,B}(x, y)$ and $flow'_{D,B}(x, y)$:

Property 2. Neither $flow_{D,B} \leq flow'_{D,B}$ nor $flow'_{D,B} \leq flow_{D,B}$.

Proof. For showing the first part, we use a differential constraint $\dot{x}_1 = 0 \land \dot{x}_2 = 0$ and a box $B = [0, 2] \times [0, 2]$. Obviously, $((1, 1), (2, 2)) \in \{(x, y) : flow_{D,B}(x, y)\}$, but $((1, 1), (2, 2)) \notin \{(x, y) : flow'_{D,B}(x, y)\}$. The reason lies in the fact that the derivatives are zero for this example, and in such a case, the equality in $flow_{D,B}(x, y)$ reduces to the trivial equality $0 = 0$ that is true for all $x, y$.

For showing the second part, we use a differential constraint $\dot{x}_1 = x_1 + x_2 + 1 \land \dot{x}_2 = x_1 + x_2 + 1$ and a box $B = [0, 2] \times [0, 2]$. Obviously, $((0, 0), (1, \frac{1}{2})) \in \{(x, y) : flow'_{D,B}(x, y)\}$, but $((0, 0), (1, \frac{1}{2})) \notin \{(x, y) : flow_{D,B}(x, y)\}$. This is...
because we need $\dot{a}_m$ and $\dot{a}_n$ to be derivatives of $a_m$ and $a_n$ at the same point in $\text{flow}_{D,B}(x, y)$, but in $\text{flow}'_{D,B}(x, y)$, $\dot{a}_i$, can be the derivative of a different point for every $i$.

However, in our method, instead of computing exact solutions to these constraints, we only use over-approximations computed by the pruning function, and a tighter constraint does not necessarily give rise to a tighter pruning result.

So let us compare these over-approximations. We want to discuss the relation between $\text{Prune}(\text{flow}_{D,B}(x, y), B_0 \times B)$ and $\text{Prune}(\text{flow}'_{D,B}(x, y), B_0 \times B)$.

**Theorem 2.** $\text{flow}'_{D,B} \preceq_P \text{flow}_{D,B}$.

**Proof.** Given a set of variables $V$ and a box $B$ for which each component corresponds to a certain variable, we denote by $\pi_V(B)$ the projection of $B$ to the components corresponding to the variables in $V$.

We transform the constraints into a conjunction without existential quantifiers as follows: Rename the variables $a_1, \ldots, a_k$ and $\dot{a}_1, \ldots, \dot{a}_k$ to a different tuple of variables in each branch, and then drop all corresponding existential quantifiers. As a result, in addition to $x$ and $y$, $\text{flow}'_{D,B}$ has $2k^2 + 1$ free variables and $\text{flow}_{D,B}$ has $k^2(k - 1)$ free variables. Obviously, we have to prove

$$\pi_{x,y}(\text{Prune}(\text{flow}'_{D,B}(x, y), B_0 \times B \times \mathbb{R}_{\geq 0} \times \mathbb{R}^{2k^2})) \subseteq \pi_{x,y}(\text{Prune}(\text{flow}_{D,B}(x, y), B_0 \times B \times \mathbb{R}^{k^2(k - 1)})).$$

Let $\phi_B' = \bigwedge_{1 \leq i \leq k} y_i = x_i + \dot{a}_it$ and $\phi_B = \bigwedge_{1 \leq m \leq n \leq k} \dot{a}_n(y_m - x_m) = \dot{a}_m(y_n - x_n)$. Let $A$ be an arbitrary, but fixed, $k$-dimensional box. Let $P'$ be the exact solution set of $\phi_B'$ in $B_0 \times B \times \mathbb{R}_{\geq 0} \times A$ and $P' = \text{Prune}(\phi_B', B_0 \times B \times \mathbb{R}_{\geq 0} \times A)$. Also, let $P_s$ be the exact solution set of $\phi_B$ in $B_0 \times B \times A$ and $P = \text{Prune}(\phi_B, B_0 \times B \times A)$. Since $(a_1, \ldots, a_k) \in B \land D((a_1, \ldots, a_k), (\dot{a}_1, \ldots, \dot{a}_k))$ is shared by both $\text{flow}_{D,B}(x, y)$ and $\text{flow}'_{D,B}(x, y)$, it suffices to prove that $\pi_{x,y}(P') \subseteq \pi_{x,y}(P)$.

We will proceed by first proving that $\pi_{x,y}(P_s') \subseteq \pi_{x,y}(P_s)$ and then lifting this to $\pi_{x,y}(P') \subseteq \pi_{x,y}(P)$.

So let $(x, y) \in \pi_{x,y}(P_s')$ be arbitrary, but fixed. We will prove that $(x, y)$ is also in $\pi_{x,y}(P_s)$. Since $(x, y)$ is in the projection of $P_s'$ we know that there are $(t, \dot{a}_1, \ldots, \dot{a}_k)$ such that $(x, y, t, \dot{a}_1, \ldots, \dot{a}_k)$ satisfies the constraint $\phi_B'$. Choose a $t^*, \dot{a}_1^*, \ldots, \dot{a}_k^*$ with that property.

For proving that $(x, y) \in \pi_{x,y}(P_s)$, it suffices to prove that $(x, y, \dot{a}_1^*, \ldots, \dot{a}_k^*)$ satisfies $\bigwedge_{1 \leq m < n \leq k} \dot{a}_n^*(y_m - x_m) = \dot{a}_m^*(y_n - x_n)$. Letting $m, n$ be arbitrary, but fixed, such that $1 \leq m < n \leq k$, we prove that $(x, y, \dot{a}_1^*, \ldots, \dot{a}_k^*)$ is in the solution set of the corresponding conjunct. Here we have three cases:

- $\dot{a}_m^* \neq 0, \dot{a}_n^* \neq 0$: Then $\frac{y_n - x_n}{a_n^*} = t^*$ and $\frac{y_m - x_m}{a_m^*} = t^*$, so $\frac{y_n - x_n}{a_n^*} = \frac{y_m - x_m}{a_m^*}$, and hence $\dot{a}_n^*(y_m - x_m) = \dot{a}_m^*(y_n - x_n)$
- $\dot{a}_m^* = 0$: then $x_m = y_m$, and both sides of the equality $\dot{a}_n^*(y_m - x_m) = \dot{a}_m^*(y_n - x_n)$ are zero,
- $\dot{a}_n^* = 0$, analogous to previous case.
Hence $\pi_{x,y}(P_s') \subseteq \pi_{x,y}(P_s)$. For lifting this to the projected results of pruning, we first observe that each conjunct of $\phi_B'$ contains each variable just once. So, due to Property 1, pruning is optimal for each conjunct. Due to the fact that the conjuncts only share a single variable $t$ also pruning of $\phi_B'$ is optimal [3], and since projection commutes with the smallest-box relation, $\pi_{x,y}(P_s')$ is the smallest box containing $\pi_{x,y}(P_s')$. Moreover, since projection commutes with the subset relation, not only $P_s' \subseteq P_s$, but also $\pi_{x,y}(P_s') \subseteq \pi_{x,y}(P_s)$, and by transitivity $\pi_{x,y}(P_s') \subseteq \pi_{x,y}(P)$. Now, the following property implies the theorem: If $B$ is the smallest box containing a set $S$ and $S \subseteq S'$, then for every box $B'$ with $S' \subseteq B'$, $B \subseteq B'$.

So, we can prune at least as tightly using $\text{Prune}(\text{flow}', A, B, 0 \times B)$ as using $\text{Prune}(\text{flow}, D, B, \{0 \times 0, 0 \times 3, 0 \times 3\})$. In fact, we can prune strictly tighter!

**Property 3.** Not $\text{flow}'_{D,B} \equiv_P \text{flow}_{D,B}$.

**Proof.** Take a differential constraint $\dot{x}_1 = 1 \land \dot{x}_2 = 1$ and a box $B = [0, 2] \times [0, 2]$. Let $B_0 = \{(1, 1)\}$, then $\text{Prune}(\text{flow'}_{D,B}(x, y), B_0 \times B) = B_0 \times [1, 2] \times [1, 2]$ and $\text{Prune}(\text{flow}_{D,B}(x, y), B_0 \times B) = B_0 \times B$.

To sum up, the theoretical evaluation shows that although none of the two constraints always has a smaller solution set than the other, $\text{flow'}_{D,B}(x, y)$ has a definite advantage in size and in pruning power. Hence we only use the latter from now on.

Next, we will compare $\text{flow'}_{A, B}(x, y)$ (or short: $\text{flow}_{A, B}(x, y)$) with the constraint $eigen_{A, B}(x, y)$ for linear differential equations $\dot{x} = Ax$.

If $A$ has $k$ different real eigenvalues and $B = \mathbb{R}^k$, $eigen_{A, B}(x, y)$ describes the exact solutions of the differential equations. But, $\text{flow'}_{A, B}(x, y)$ employs the first-order Taylor expansion to over-approximate the exact solutions. Thus, $eigen_{A, B} \preceq \text{flow'}_{A, B}$? No! Only in cases where all trajectories leave the box, and do not enter it again. Otherwise, $eigen_{A, B}(x, y)$ also includes the part of the trajectory that enters the box again, but $\text{flow'}_{A, B}(x, y)$ does not necessarily.

Now we compare the two constraints wrt. pruning. For linear differential equations, pruning is optimal for $\text{flow'}_{A, B}$. This follows from an analysis of the proof of Theorem 2, and the fact that the differential equation constrains each derivative $\dot{x}$ using a constraint $\dot{x} = Ax$, where each equation of this constraint contains only one component of $\dot{x}$, and contains each component of $x$ only once. However, this is in general not the case for $eigen_{A, B}$ due to multiple occurrences of variables.

There is no clearcut relationship between $\text{Prune}(\text{flow}_{A, B}(x, y), B_0 \times B)$ and $\text{Prune}(eigen_{A, B}(x, y), B_0 \times B)$:

**Property 4.** Neither $\text{flow}_{A, B} \preceq_P eigen_{A, B}$ nor $eigen_{A, B} \preceq_P \text{flow}_{A, B}$.

**Proof.** This can be directly proven using only one example with a differential equation $\dot{x}_1 = -x_1 - x_2$, $\dot{x}_2 = x_1 - x_2$ and a box $B = [0, 4] \times [0, 4]$. If we set $B_0 = [2.5, 3] \times [0, 0]$, then $\text{Prune}(\text{flow}_{A, B}(x, y), B_0 \times B) = B_0 \times [0, 3] \times [0, 4]$ and $\text{Prune}(eigen_{A, B}(x, y), B_0 \times B) = B_0 \times [0, 3.5] \times [0, 3]$.
Moreover, there are even some cases, where pruning \( \text{flow}'_{A,B} \) returns a strict subset of pruning \( \text{eigen}_{A,B} \) and vice versa: The former happens for a differential equation \((x_1, x_2) = (x_1 - x_2, x_1 + x_2)\) and a box \( B = [0, 2] \times [0, 4]\). If we set \( B_0 = [2, 2] \times [2, 4] \), then \( \text{Prune}(\text{flow}'_{A,B}, B_0 \times B) = B_0 \times [0, 2] \times [2, 4] \) and \( \text{Prune}(\text{eigen}_{A,B}, B_0 \times B) = B_0 \times B \). This is because the left-hand side of \( \text{eigen}_{A,B} \), \( c^T y \), evaluates to zero on some element in \( B \). Hence every solution of \( c^T(x \cos(bt) + \frac{2-x_0}{2} \sin(bt)) = 0 \) fulfills the constraint. There is such a solution, and since \( \sin \) and \( \cos \) are periodic, the solution set is not bounded for \( t \), and interval \([0, +\infty]\) for \( t \) will not be pruned. Since \( a \) is positive, the interval derived for the term \( e^{at} \) will also stay unbounded, and no intervals will be pruned. But, \( t \in [0, 1] \) in \( \text{flow}'_{A,B} \), which does provide some pruning.

The latter happens for a differential equation \((\dot{x}_1, \dot{x}_2) = (x_1, x_2)\) and a box \( B = [0, 2] \times [0, 2] \). If we set \( B_0 = \{(0, 0)\}, \text{Prune}(\text{flow}'_{A,B}, B_0 \times B) = B_0 \times B \) and \( \text{Prune}(\text{eigen}_{A,B}, B_0 \times B) = B_0 \times B_0 \). This is because \( \text{eigen}_{A,B} \) here describes the exact solution starting from the initial point. But, since \( \dot{x}_1 \) and \( \dot{x}_2 \) can be zero, pruning \( \text{flow}'_{A,B} \) results in \( t \in [0, \infty] \). Thus, also the intervals for \( y_1 \) and \( y_2 \) cannot be pruned.

Since there is no clearcut relationship between \( \text{Prune}(\text{flow}'_{A,B}, B_0 \times B) \) and \( \text{Prune}(\text{eigen}_{A,B}, B_0 \times B) \), we strengthen both constraints by combining them. Thus, by sharing the same time variable we allow timing information to be propagated between them as follows:

**Lemma 4.** For a linear differential equation \( \dot{x} = Ax \) and a box \( B \), if there is a trajectory in \( B \) from a point \( x = (x_1, \ldots, x_k)^T \in B \) to a point \( y = (y_1, \ldots, y_k)^T \in B \) on which \( \dot{x} = Ax \) holds, then

\[
\exists t \in \mathbb{R}_{\geq 0} [\text{flow}^*_{A,B}(t, x, y) \wedge \text{eigen}^*_{A,B}(t, x, y)]
\]

We denote the above new Constraint 3 by \( \text{comb}_{A,B}(x, y) \). Clearly, this constraint implies \( \text{flow}'_{A,B}(x, y) \), and also implies \( \text{eigen}_{A,B}(x, y) \). That is, \( \text{comb}_{A,B} \preceq \text{flow}'_{A,B} \) and \( \text{comb}_{A,B} \preceq \text{eigen}_{A,B} \). Moreover, we have:

**Theorem 3.** \( \text{comb}_{A,B} \preceq_P \text{flow}'_{A,B} \) and \( \text{comb}_{A,B} \preceq_P \text{eigen}_{A,B} \).

So, the combination constraint is at least as good as \( \text{flow}'_{A,B}(x, y) \) and \( \text{eigen}_{A,B}(x, y) \). But, in fact, it is better!

**Property 5.** Neither \( \text{comb}_{A,B} \equiv_P \text{flow}'_{A,B} \) nor \( \text{comb}_{A,B} \equiv_P \text{eigen}_{A,B} \).

**Proof.** This can be seen on an example with a differential equation \((\dot{x}_1, \dot{x}_2) = (-x_1 - x_2, x_1 - x_2)\) and a box \( B = [0, 4] \times [0, 4] \). If we set \( B_0 = [2.5, 3] \times [0, 0] \), then \( \text{Prune}(\text{flow}'_{A,B}, B_0 \times B) = B_0 \times [0, 3] \times [0, 4], \text{Prune}(\text{eigen}_{A,B}, B_0 \times B) = B_0 \times [0, 3.5] \times [0, 3] \) and \( \text{Prune}(\text{comb}_{A,B}, B_0 \times B) = B_0 \times [0, 3] \times [0, 3] \). \( \square \)

However, the combination constraint is bigger than both \( \text{flow}'_{A,B}(x, y) \) and \( \text{eigen}_{B}(x, y) \). Thus, pruning will take more time on it.
6 Empirical Evaluation

In this section we evaluate the constraints empirically by using them in the verification method introduced in Section 2. That is, we replace all occurrences of Reachable_B(s, x, y) introduced in Theorem 1 by flow_{D,B} (or, flow_{A,B}), flow'_{D,B} (or, flow'A_B), eigen_{A,B} and comb_{A,B}, respectively. We illustrate the behavior of our implementation on a few benchmark problems. Note that in the literature on the verification of hybrid systems the habit prevails to test new methods only on 2-3, or even less examples. We do not follow this tradition and do more extensive benchmarking. We will publish the corresponding HSolver input files on its web-page [15].

Note that we use the following splitting strategy here: In each mode, we choose the box with the biggest side-length, and then bisect each choice along the variable along which the box has not been split the longest time (i.e., we use a round-robin strategy to choose the variable). The computations were performed on a Pentium IV, 2.60GHz with 1 GB RAM, and they were canceled when computation did not terminate before 5 hours of computation time.

We used the following benchmark problems for comparing flow_{D,B} and flow'_{D,B} with computation results shown in Table 1:

Example 1. The flow constraints are constructed by setting all the parameters in the two tanks problem [18] to 1.

Flow: (\dot{x}_1, \dot{x}_2) = (-x_1 - x_2, x_1 - x_2), empty jump relation
Init: 2.5 ≤ x_1 ≤ 3 ∧ x_2 = 0, Unsafe: x_1 > 3 ∨ x_2 > 3
The state space: [0.4] × [0.4]

Example 2. Flow: (\dot{x}_1, \dot{x}_2) = (x_1 - x_2, x_1 + x_2), empty jump relation
Init: 2.5 ≤ x_1 ≤ 3 ∧ x_2 = 0, Unsafe: x_1 ≤ 2
The state space: [0.4] × [0.4]

Example 3. The flow constraints are constructed by setting all the parameters in the two tanks problem [18] to 1.

Flow: (s = 1 → (\dot{x}_1) = (\frac{1}{x_1} - \sqrt{x_1})) ∧ (s = 2 → (\dot{x}_1) = (\frac{1}{x_1} - \sqrt{x_1}))
Jump: (s = 1 ∧ 0.99 ≤ x_2 ≤ 1) → (s' = 2 ∧ x_1 = x_1 ∧ x_2 = 1)
Init: s = 1 ∧ (x_1 - 5.5)^2 + (x_2 - 0.25)^2 ≤ 0.0625
Unsafe: (s = 1 ∧ (x_1 - 4.25)^2 + (x_2 - 0.25)^2 < 0.0625)
The state space: (1, [4, 6] × [0, 1]) ∪ (2, [4, 6] × [1, 2])

Example 4. This is a predator-prey example.

Flow: (s = 1 → (\dot{x}_1) = (-x_1 + x_1 x_2) ∧ (s = 2 → (\dot{x}_1) = (x_2 - x_1 x_2))
Jump: \((s = 1 ∧ 0.875 ≤ x_2 ≤ 0.9) → (s' = 2 ∧ (x'_1 - 1.2)^2 + (x'_2 - 1.8)^2 ≤ 0.01)) ∨ ((s = 2 ∧ 1.1 ≤ x_2 ≤ 1.125)) → (s' = 1 ∧ (x'_1 - 0.7)^2 + (x'_2 - 0.7)^2 ≤ 0.01))
Init: s = 1 ∧ (x_1 - 0.8)^2 + (x_2 - 0.2)^2 ≤ 0.01
Unsafe: (s = 1 ∧ x_1 > 0.8 ∧ x_2 > 0.8 ∧ x_1 ≤ 0.9 ∧ x_2 ≤ 0.9)
State space: (1, [0.1, 0.9] × [0.1, 0.9]) ∪ (2, [1.1, 1.9] × [1.1, 1.9])
Example 5. This is a simple example with a clock variable.
Flow: \((\dot{x}, \dot{y}, \dot{t}) = (-5.5y + y^2, 6x - x^2, 1)\), empty jump relation
Init: \(4 \leq x \leq 4.5 \land y = 1 \land t = 0\)
Unsafe: \((1 \leq x < 2 \land 2 < y < 3 \land 2 \leq t \leq 4)\)
The state space: \([1, 5] \times [1, 5] \times [0, 4]\)

Example 6. A three-dimensional and nonlinear example about a simple controller that steers a car along a straight road [4]. The three continuous variables are the position \(x\), the heading angle \(\gamma\) and the internal timer \(c\). Since we cannot prove the safety property described in the original paper, in this paper the unsafe space is reset to be \(x \leq -4\).

Example 7. A linear collision avoidance example from a part of the car convoy control from a paper by A. Puri and P. Varaiya [11]. Let \(gap, v_r, v_l\) and \(a_r\) respectively represent the distance between the two cars \((d_{i-1} - d_i)\) in the original paper), the velocity of the rear car \(d_i\), the velocity of the leading car \(d_{i-1}\) and the acceleration of the rear car \(d_{i-1}\). By using these variables and restricting \(v_l\) by \(-2 \leq v_l \leq -0.5\) we transformed the original higher-order differential equation into a four-dimensional differential (in)equation of order one.

We set the state space to \([0, 4] \times [0, 2] \times [0, 2] \times [-2, -0.5]\), and we want to verify that \(gap > 0\) when starting from \(gap = 1, v_r = 2, v_l = 2\) and \(a_r = -0.5\).

Example 8. A four-dimensional and nonlinear example about a mixing-tank system from a paper by O. Stursberg, S. Kowalewski and S. Engell [17]. In the original paper, the system is simplified to a two-dimensional system. In this paper, we keep the differential equations \((\dot{V}_1, \dot{V}_2) = (0.008, 0.015)\) in the flow constraint, where \(V_1\) and \(V_2\) are two inlet streams. Then, initially, \(V_1(0) = 1, V_2(0) = 1\), and \((h(0), c(0)) \in [1.32, 1.5] \times [1.2, 1.32]\), where \(h\) is liquid height and \(c\) is concentration. We want to verify that the state \(\{(V_1, V_2, h, c) : h \in [1.1, 1.3] \land c \in [1.68, 1.80]\}\) is unreachable.

Example 9. A two-dimensional and nonlinear example about a tunnel-diode oscillator circuit [6]. It models the voltage drop \(V\) and the current \(I\). The original problem was to prove that all trajectories eventually reach a certain set and stay there. We transformed it to a reachability problem, using the state space \([-0.1, 0.6] \times [-0.002, 0.002]\) and the unsafety constraint \(V < -0.04 \lor V > 0.54 \lor I < -0.0015 \lor I > 0.00175\).

Example 10. A linear, three-dimensional model of a mutant of V. \textit{fischeri} [2]. Let \(x_1, x_2\) and \(x_3\) respectively represent the protein \(\text{LuxI}\), the autoinducer \(A\) and the complex \(Co\) described in the original paper. The model has two modes with dynamics in the form \(\dot{x} = Ax + b_i, \ i = 1, 2\), where \(x = (x_1, x_2, x_3)^T\) and

\[
A = \begin{pmatrix}
-1/3600 & 0 & 0 \\
7.5e - 5 & -1(1/3600 + 7.5e - 9) & 1.5e - 9 \\
0 & 0.005 & -1/3600 - 0.01
\end{pmatrix}
\]

and \(b_1 = (0.00375, 0, 0)^T\) and \(b_2 = (3.75375, 0, 0)^T\).
We set the state space to be \([0, 30000] \times [0, 60000] \times [0, 30000]\) and the switches occur when the plane \(x_3 = 1000\) is reachable and \(x_2 \in [1000, 45000]\). We want to verify that \(x_1 \geq 27500 \lor x_2 \geq 50000 \lor x_3 \geq 25000\) cannot be reachable when starting from \([17500, 20000] \times [40000, 45000] \times [5000, 7500]\) in mode 1.

### Table 1. Computation results for \(flow_{D,B}\) and \(flow'_{D,B}\)

<table>
<thead>
<tr>
<th>Example</th>
<th>(flow_{D,B})</th>
<th>(flow'_{D,B})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU time</td>
<td>Splitting steps</td>
</tr>
<tr>
<td>1</td>
<td>0.041s</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>0.34s</td>
<td>71</td>
</tr>
<tr>
<td>3</td>
<td>0.18s</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>0.57s</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>2.59s</td>
<td>93</td>
</tr>
<tr>
<td>6</td>
<td>0.35s</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>187.96s</td>
<td>369</td>
</tr>
<tr>
<td>8</td>
<td>7.68s</td>
<td>54</td>
</tr>
<tr>
<td>9</td>
<td>13.209s</td>
<td>165</td>
</tr>
<tr>
<td>10</td>
<td>1876s</td>
<td>1889</td>
</tr>
</tbody>
</table>

The results show that the new constraint improves the number of pruning steps for all examples except for Example 8, which we will discuss below. As expected, this also decreases the run-time of the method except for 2-dimensional examples, where \(flow'_{D,B}\) has more conjuncts than \(flow_{D,B}\).

We analyzed the anomaly in the behavior on Example 8 in more detail. After applying the pruning algorithm for the first time to \(reachable_{s'}(s', z)\), using \(flow'_{D,B}\) we can prune the box \([0, 2] \times [0, 2] \times [0.5, 1.5] \times [1.2, 1.8]\) to a new box \([1, 1.53333333333] \times [1, 2] \times [1.22034017148, 1.5] \times [1.2, 1.8]\); but, after we apply the pruning algorithm to \(reachable_{s'}(s', z)\) using \(flow_{D,B}\), we can only prune the box \([0, 2] \times [0, 2] \times [0.5, 1.5] \times [1.2, 1.8]\) to a new box \([0.46666666667, 1.53333333333] \times [0, 2] \times [1.22034017148, 1.5] \times [1.2, 1.8]\). So, in fact, the new method is better at the beginning! However, it seems that this improvement at the beginning turns out to be bad luck later since our method is very sensitive to splitting heuristics, and the improved pruning results in different choices of boxes for splitting during the algorithm. This suggests that a detailed study of splitting heuristics, will be able to significantly improve the method further.

In addition to the linear examples from above (Examples 1, 2 and 10), we used the following benchmarks for comparing \(flow'_{A,B}\), \(eigen_{A,B}\) and \(comb_{A,B}\) with results shown in Table 2:

**Example 11.** A linear, three-dimensional example.
Flow: \((\dot{x}_1, \dot{x}_2, \dot{x}_3) = (0.80x_2 + 0.6x_3 - 1.8, 0.8x_1 + 0.7x_3 - 15.2, 0.6x_1 + 0.7x_2 - 1.8);\)
Empty jump relation; Init: \(19 \leq x_1 \leq 20 \land 19 \leq x_2 \leq 20 \land 19 \leq x_3 \leq 20;\)
Unsafe: \(x_1 \leq 21 \land x_2 \leq 20 \land x_3 \geq 22.5;\)
The state space: \([15, 24] \times [15, 24] \times [15, 24].\)
Table 2. Computation results for \( \text{flow}'_{A,B} \), \( \text{eigen}_{A,B} \) and \( \text{comb}_{A,B} \)

<table>
<thead>
<tr>
<th>Example</th>
<th>( \text{flow}'_{A,B} )</th>
<th>( \text{eigen}_{A,B} )</th>
<th>( \text{comb}_{A,B} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU time</td>
<td>Splitting steps</td>
<td>Pruning number</td>
</tr>
<tr>
<td>1</td>
<td>0.020s</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>0.38s</td>
<td>58</td>
<td>1203</td>
</tr>
<tr>
<td>10</td>
<td>686s</td>
<td>1417</td>
<td>270078</td>
</tr>
<tr>
<td>11</td>
<td>unknown</td>
<td>unknown</td>
<td>0.319s</td>
</tr>
<tr>
<td>12</td>
<td>unknown</td>
<td>0.465s</td>
<td>0</td>
</tr>
</tbody>
</table>

Example 12. A linear collision avoidance example similar to Example 7. We restrict \( v_t \) by \( \dot{v}_t = 0 \), and reset the state space to be \([0, 10] \times [0, 30] \times [0, 30] \times [-2, 5] \) and the initial set to be \(-0.8522v_r - 0.1478v_l - 0.3177a_r + \text{gap} > 10 \).

The results show that the combination decreases the size of the abstraction and the number of calls to the constraint solver. However, as expected, this will for some cases increase the run-time of the method, due to the bigger size of this constraint. This phenomenon is reflected by Examples 1, 2 and 12. But, for hard (and thus realistic) problems (e.g., Example 10), the improvement due to the first phenomenon will always dominate: in such cases the time spent on constraint solving will always be dominated by computations on the abstraction, and hence it is essential to keep the abstraction small.

For some cases, the safety property cannot be verified using \( \text{eigen}_{A,B} \) in our method. For Examples 1 and 2 this can be explained using an observation already discussed in Section 5: the eigenvalues are complex with non-zero imaginary parts, and in such a case, since \( t \) occurs several times in the term \( c^T (x \cos(bt) + \frac{\dot{x} - ax}{b} \sin(bt)) \), we will get an over-approximating interval for this term.

On Examples 10 and 11 it can be seen nicely that the combined constraint can be stronger than either \( \text{flow}'_{A,B} \) or \( \text{eigen}_{A,B} \) in isolation.

Note that we did not use Examples 11 and 12 in Table 1 because their safety properties cannot be verified using either \( \text{flow}_{A,B} \) or \( \text{flow}'_{A,B} \).

7 Conclusion

We have provided a detailed study of two types of constraints in the verification of hybrid systems. The overall approach, to formulate reach set computation as a constraint solving problem, and to apply an efficiently over-approximating constraint solver to it, can be extended to various new types of constraints. Specifically we will study the use of higher order Taylor approximations instead of the constraint based on the mean value theorem. Our software is publically available [15], and includes an interface that allows the incorporation of and experimentation with new, user-defined constraints. Based on the gained experience and user feedback, we will optimize the constraint solver especially for the most useful ones.
References


Chapter 10

Efficient Solving of Quantified Inequality Constraints over the Real Numbers

This chapter appeared as:

Stefan Ratschan. “Efficient Solving of Quantified Inequality Constraints over the Real Numbers”. In: ACM Transactions on Computational Logic 7.4 (2006), pp. 723–748
Efficient Solving of Quantified Inequality Constraints over the Real Numbers

STEFAN RATSCHAN
Max-Planck-Institut für Informatik, Saarbrücken, Germany

Let a quantified inequality constraint over the reals be a formula in the first-order predicate language over the structure of the real numbers, where the allowed predicate symbols are $\leq$ and $<$. Solving such constraints is an undecidable problem when allowing function symbols such as $\sin$ or $\cos$.

In this article, we give an algorithm that terminates with a solution for all, except for very special, pathological inputs. We ensure the practical efficiency of this algorithm by employing constraint programming techniques.

Categories and Subject Descriptors: F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic—Logic and constraint programming; G.1.0 [Numerical Analysis]: General—Interval arithmetic, stability (and instability); I.2.3 [Artificial Intelligence]: Deduction and Theorem Proving

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Constraint solving, decision procedures, numerical constraints

1. INTRODUCTION

The problem of solving quantified constraints over the reals has numerous applications (we have created a web-page that lists more than fifty references [Ratschan 2001a]). However, it is undecidable, when allowing function symbols such as $\sin$ or $\cos$ [Tarski 1951], and highly complex when restricting oneselfs to addition and multiplication [Davenport and Heintz 1988; Weispfenning 1988].

In this article we give an algorithm that nevertheless always terminates for inputs that are stable in the sense that their truth value (in the case where
all variables are quantified) does not change under small perturbations of the occurring constants. For example, the constraint \( \forall x \ x^2 + 1 \geq 0 \) is stable, whereas \( \forall x \ x^2 \geq 0 \) is not.

Furthermore, we ensure the practical efficiency of the algorithm by basing it on techniques from the field of constraint programming [Benhamou et al. 1994; Benhamou and Older 1997; Sam-Haroud and Faltings 1996; Van Hentenryck et al. 1997]. The basic idea of these techniques is to reduce the average runtime of algorithms for computationally hard problems by replacing expensive exhaustive search as much as possible by methods for pruning elements from the search space for which it is easy to show that they do not contain solutions.

In this article, we extend this idea to quantified inequality constraints for which all (free and bound) variables are bounded to a closed interval: We try to prune elements from these bounds for which it is easy to show that they do not contain solutions. When we cannot easily prune more elements, we do branching by splitting a bound into pieces (for quantified variables this means replacing sub-constraints of the form \( \forall x \in I \ \phi \) by \( \forall x \in I_1 \ \phi \land \forall x \in I_2 \ \phi \) where \( I = I_1 \cup I_2 \), or the corresponding existential case). This gives us new possibilities for pruning. We repeat the two steps until we have pruned all elements (or disproved the constraint). For computing elements of the bounds that do contain solutions, we take the negation of the input constraint and again apply the above branch-and-prune approach.

In this article, we formalize this approach, study its properties in detail, improve it for an implementation, and do timings that show its efficiency.

As a side-effect, this article even improves the current methods for numerical constraint satisfaction problems in the case where the solution set does not consist of finitely many, isolated solutions, which—up to now—was essential for their efficiency. For example, the book describing the system Numerica [Van Hentenryck et al. 1997] explicitly states that for inputs not fulfilling that property the method creates a huge number of boxes.

In order to be able to reuse existing theory, algorithms and software for solving atomic inequality constraints (i.e., constraints of the form \( t \geq 0 \) or \( t > 0 \), where \( t \) is a term), and to be able to benefit from further progress in this area, the article employs a parametric approach: It takes as input theory and algorithms from constraint programming, and provides as output corresponding new theory and algorithms for solving quantified constraints. More specifically, building upon the notion of a narrowing operator [Van Hentenryck et al. 1997a; Benhamou et al. 1999; Hong and Stahl 1994], it takes as input: a specification describing a consistency notion for atomic inequality constraints (e.g., box-consistency [Benhamou et al. 1994]), and a narrowing operator that implements this specification. It provides as output: a specification describing a corresponding consistency notion for quantified constraints, a narrowing operator that implements this specification, and an algorithm for computing approximate solutions of quantified constraints over the reals that uses this narrowing operator for pruning. These outputs are accompanied with proofs of their usefulness/optimality.

For the special case where the only allowed function symbols are addition and multiplication, up to recently, all algorithms have been based on computer
algebra methods [Collins 1975; Caviness and Johnson 1998], which resulted in certain drawbacks (e.g., low practical efficiency, restriction to polynomials, unwieldy output expressions). In an earlier paper [Ratschan 2002a], the author of this article proposed a scheme for solving quantified constraints approximately that followed the idea of quantifier elimination by cylindrical algebraic decomposition [Collins 1975; Caviness and Johnson 1998], but decomposed space into floating-point boxes instead of semi-algebraic cells. This approach was successful in showing that one can efficiently compute approximate solutions of quantified constraints using interval methods. However, it still had several drawbacks. Especially, it was not clear when and how to optimize box splitting, because the algorithm was not separated into (inherently exponential) search, and pruning. This article provides a solution to this, and other, problems of the older approach.

The following special cases of the general problem have been studied using interval or constraint satisfaction methods:

— The case of expressions of the form $\forall \overrightarrow{p} \ \phi(\overrightarrow{p}, \overrightarrow{q})$, where $\phi(\overrightarrow{p}, \overrightarrow{q})$ is a system of strict inequalities [Jaulin and Walter 1996; Malan et al. 1997; Jaulin et al. 2001] using methods that repeatedly bisect the free-variable space, and test after each bisection, whether the system of inequalities holds everywhere on the resulting box.

— The case of expressions of the form $\forall p \ \phi(p, \overrightarrow{q})$, where $\phi(p, \overrightarrow{q})$ is a system of strict inequalities [Benhamou and Goualard 2000], using methods that correspond to our case for universal quantification, conjunction and atomic constraints, but without branching in the universally quantified variables.

— The case of quantified systems of equations where certain variables occur only once, and the quantifiers obey certain orderings (various results by S. Shary, see e.g., [Shary 2002]).

— The case of disjunctive constraints. This has been done in the discrete case [Jourdan and Sola 1993; Van Hentenryck et al. 1995], and in the continuous case for disjunctive constraints occurring in interactive graphical applications [Marriott et al. 2001], and in a similar way as in this article for speeding up solving of factorizable constraints [Granvilliers 1998].

See also an overview on methods for solving quantified inequalities in control [Dorato 2000]. For improving box splitting strategies for inequality constraints, Silaghi et al. [2001] use information from the negation of the input constraints. A similar problem is the problem of extending the bounds of universal quantifiers, for which a method based on constraint propagation [Collavizza et al. 1999] is provided for the special case of a system of inequalities for which all variables are universally quantified. Also for discrete domains there is a lot of recent interest solving constraints with quantifiers [Bordeaux and Monfroy 2002; Fargier et al. 1996], or related stochastic constraints [Walsh 2002]. Some of the above [Collavizza et al. 1999; Benhamou and Goualard 2000; Silaghi et al. 2001] take a similar approach of using the negation of the input to compute positive information. However, they do not address the question of being able to compute answers for all except unstable inputs.
The content of this article is as follows: Section 2 gives various preliminaries; Section 3 introduces a framework for reusable pruning based on the notion of narrowing operator; Section 4 describes an according notion of consistency for quantified constraint that allows us to specify the pruning power of narrowing operators; Section 5 gives a generic algorithm for pruning that implements a narrowing operator; Section 6 bases an according branch-and-prune solver for quantified constraints on this pruning algorithm. Section 7 discusses how to arrive at an efficient implementation of such a solver; Section 8 presents timings of such an implementation; Section 9 discusses the relation of the results to symbolic quantifier elimination algorithms; and Section 10 concludes this article.

2. PRELIMINARIES

We fix a set $V$ of variables. A quantified constraint (or short: constraint) is a formula in the first-order predicate language over the reals with predicate and function symbols interpreted as suitable relations and functions, and with variables in $V$. We take over a large part of the according predicate-logical terminology without explicit definitions.

In this article, we restrict ourselves to the predicate symbols $<, >, \le, \ge$, and assume that equalities are expressed by inequalities on the residual (i.e., $f = 0$ as $|f| \le \varepsilon$ or $f^2 \le \varepsilon$, where $\varepsilon$ is a small positive real constant\(^1\)). Furthermore we only deal with constraints without negation symbols because one can easily eliminate negation symbols from quantified constraints by pushing them down, and replacing atomic constraints of the form $\neg(f \le g)$ by $f > g$, and $\neg(f < g)$ by $f \ge g$, respectively. For any quantified constraint $\phi$, let $\bar{\phi}$ (the opposite of $\phi$) be the quantified constraints that results from $\neg\phi$ by eliminating the negation by pushing it down to the predicates.

We require that every quantifier be bounded by an associated quantifier bound, using expressions of the form $\exists x \in I$ or $\forall x \in I$, where $I$ is a closed interval.

A variable assignment is a function from the set of variables $V$ to $\mathbb{R}$. We denote the semantics of a constraint $\phi$, the set of variable assignments that make $\phi$ true, by $[\! [\phi] \! ]$. For example, $[\! [x^2 + y^2 \leq 1] \! ]$ is the set of variable assignments that assign values within the unit disc to $x$ and $y$.

For any variable assignment $d$, any variable $v \in V$ and any real number $r$, we denote by $d_r^v$ the variable assignment that is equal to $d$ except that it assigns $r$ to $v$.

Let $I$ be the set of closed real intervals. We denote by $I_1 \sqcup I_2$ the smallest interval containing both intervals $I_1$ and $I_2$. A box assignment is a set of variable assignments that can be represented by functions from $V$ to $I$; that is, it contains all the variable assignments that assign elements within a certain interval to a variable. For example, for $V = \{x, y\}$, the set of variable assignments that assign an element of $[-1, 1]$ to both $x$ and $y$ is a box assignment—this set of variable assignments can be represented by the function that assigns $[-1, 1]$ to both $x$ and $y$.

\(^1\)The constant $\varepsilon$ needs to be non-zero because otherwise solutions would vanish under small perturbations of $\varepsilon$, resulting in an unstable [Ratschan 2002d] constraint.
From now on, we will use a box assignment and its interval function representation interchangeably. For any box assignment $B$, any variable $v \in V$, and any interval $I$, we denote by $B \upharpoonright v$ the box assignment that is equal to $B$ except that it assigns $I$ to $v$. In the context of closed constraints, we denote by the Boolean value $F$ the empty box assignment and by the Boolean value $T$ the box assignment that assigns the set of real numbers $\mathbb{R}$ to each variable, and allow the usual Boolean operation on them. We denote by \{ $x \mapsto [-1, 1], y \mapsto -$ \} a box assignment that assigns the interval $[-1, 1]$ to the variable $x$ and an arbitrary interval to the variable $y$.

Traditionally, constraint programming techniques [Benhamou et al. 1994; Benhamou and Older 1997; Sam-Haroud and Faltings 1996; Van Hentenryck et al. 1997a] use boxes (i.e., Cartesian products of intervals) instead of box assignments. However, when working with predicate logic, the additional flexibility of box assignments is very convenient in dealing with the scoping of variables. For efficiency reasons, an actual implementation might represent box assignments by boxes.

3. A FRAMEWORK FOR REUSABLE PRUNING

Remember that our approach will be to solve quantified constraints by a branch and prune algorithm. Fortunately, there is already a lot of work done on how to do pruning on atomic constraints, and their conjunctions. The formal framework for this is the notion of a narrowing operator [Benhamou 1995; Benhamou and Older 1997], which specifies some properties required of such an algorithm without regard to the concrete algorithm. In this article, we generalize this notion to quantified constraints. This will allow us to reuse existing theory, algorithms, and software implementing such narrowing operators. Readers who are only interested in a concrete pruning algorithm and not in a formal framework for reasoning about its properties can directly jump to Section 5.

The essential difference between our approach and the classical one is that quantified constraints also store information about the range of variables within the constraint, and so we allow a narrowing operator to modify constraints also. For this, we use pairs $(\phi, B)$, where $\phi$ is a quantified constraint and $B$ is a box assignment. We call such pairs bounded constraints, and the second element of a bounded constraint its free-variable bound.

**Definition 1.** A narrowing operator is a function $N$ on bounded constraints such that for bounded constraints $(\phi, B)$, and $(\phi', B')$, and for $N(\phi, B) = (\phi_N, B_N)$, and $N(\phi', B') = (\phi'_N, B'_N)$,

- $B \supseteq B_N$ (contractance),
- $[[\phi_N]] \cap B_N = [[\phi]] \cap B_N$ (soundness),
- $B' \subseteq B$ implies $B'_N \subseteq B_N$ (monotonicity), and
- $N(N(\phi, B)) = N(\phi, B)$ (idempotence).

Note that we use a soundness condition here, instead of a correctness condition: We require that the solution set of the resulting constraint be the same only on the resulting box, but not necessarily on the initial box. We will ensure full correctness by the next definition.
Constraint programming techniques for continuous domains traditionally compute outer approximations of the solution set. However, here we also want to compute inner approximations for three reasons: First, we need to compute such inner approximations for proving closed constraints to be true. Second, the solution set of quantified constraints with inequality predicates usually does not consist of singular points, but of sets that can have volume, and for many applications it is important to find points that are guaranteed to be within this solution set. And third, available inner approximations can speed up the computation of outer approximations and vice-versa, because any element known to be within the solution set, or known to be not in the solution set, does not need to be inspected further.

So we will allow two kinds of narrowing operators—one that only removes elements not in the solution set, and one that only removes elements in the solution set.

**Definition 2.** An outer narrowing operator is a narrowing operator $N$ such that for every bounded constraint $(\phi, B)$, for the free-variable bound $B_N$ of $N(\phi, B)$, $B_N \supseteq B \cap \|\phi\|$. An inner narrowing operator is a narrowing operator $N$ such that for every bounded constraint $(\phi, B)$, for the free-variable bound $B_N$ of $\overline{N}(\phi, B)$, $B_N \supseteq B \setminus \|\phi\|^2$.

As discussed in the introduction, we get an inner narrowing operator from an outer narrowing operator by working on the opposite of the input:

**Theorem 1.** Let $\overline{N}$ be a function on bounded constraints and let $\overline{N}(\phi, B) := (\forall \neg \phi_N, B_N)$ where $(\phi_N, B_N) = \overline{N}(\forall \neg \phi, B)$. Then, $N$ is an outer narrowing operator iff $\overline{N}$ is an inner narrowing operator.

**Proof.** Obviously $\overline{N}$ is a narrowing operator iff $\overline{N}$ is a narrowing operator. $N$ is outer narrowing iff $\overline{N}$ is inner narrowing because for any bounded constraint $(\phi, B)$, $B \cap \|\neg \phi\| = B \setminus \|\phi\|$, and $B \setminus \|\neg \phi\| = B \cap \|\phi\|$. □

A similar observation has already been used for the special case of quantified constraints with one universal quantifier [Benhamou and Goualard 2000]. The above theorem allows us to concentrate on outer narrowing operators from now on. We get the corresponding inner narrowing operator for free by applying the outer narrowing operator on the opposite of the input.

### 4. Consistency of Quantified Constraints

In constraint programming, the notion of consistency is used to specify the pruning power of narrowing operators. In this section, we generalize this approach to quantified constraints. Again, readers who are only interested in a concrete algorithm and not in formal reasoning about its properties, can skip this section.

---

2This definition of inner narrowing operator differs from the one used by Benhamou and Goualard [2000].
Clearly, it is not possible to prune the empty set further. So we require the following from a predicate on bounded constraints that we use for such specification purposes:

**Definition 3.** A *consistency property* is a predicate $C$ on bounded constraints such that for every constraint $\phi$, $C(\phi, \emptyset)$.

**Example 1.** For an atomic bounded constraint $(\phi, B)$, $BC(\phi, B)$ holds iff there is no (canonical-interval-wide) face of the hyperrectangle described by $B$ for which interval evaluation [Moore 1966; Neumaier 1990; Jaulin et al. 2001] will prove that it contains no element of $[\phi]$. In this case, we say that $\phi$ is *box-consistent* with respect to $B$ [Benhamou et al. 1994; Van Hentenryck et al. 1997a].

The strongest form of consistency achievable using floating-point numbers is:

**Example 2.** For a bounded constraint $(\phi, B)$, $HC(\phi, B)$ holds iff there is no box assignment $B'$ with floating-point endpoints such that $B' \subset B$ and $[\phi] \cap B' = [\phi] \cap B$. In this case, we say that $\phi$ is *hull-consistent* with respect to $B$ [Benhamou and Older 1997].

Note that in the constraint programming literature [Cleary 1987; Benhamou 1995; Benhamou et al. 1994], the definition of hull-consistency usually assumes, that the according constraints have been decomposed into so-called *primitive* constraints. We do not follow this approach here, because that would blur the borderline between consistency properties and symbolic preprocessing of constraints.

The following is the strongest form of consistency that does not result in loss of information, that is, for which an outer narrowing operator exists.

**Definition 4.** For a bounded constraint $(\phi, B)$, $TC(\phi, B)$ holds iff there is no box assignment $B'$ such that $B' \subset B$ and $[\phi] \cap B' = [\phi] \cap B$. In this case, we say that $\phi$ is *tightly consistent* with respect to $B$.

Now we can use consistency properties as specifications for the effectiveness of narrowing operators:

**Definition 5.** Given a consistency property $C$, a narrowing operator $\mathcal{N}$ ensures $C$ iff for all bounded constraints $(\phi, B)$, $C(\mathcal{N}(\phi, B))$ holds.

Now we assume a certain consistency property on literals (i.e., atomic constraints and their negations) and lift it to a corresponding consistency property on quantified constraints.

**Definition 6.** Given a quantified constraint $\phi$ and a consistency property $C$ on literals, let $\hat{C}$ be the following predicate:

- if $\phi$ is a literal, then $\hat{C}(\phi, B)$ iff $C(\phi, B)$
- if $\phi$ is of the form $\phi_1 \land \phi_2$, then $\hat{C}(\phi, B)$ iff $\hat{C}(\phi_1, B)$ and $\hat{C}(\phi_2, B)$.
- if $\phi$ is of the form $\phi_1 \lor \phi_2$, then $\hat{C}(\phi, B)$ iff $B = B_1 \uplus B_2$, where $\hat{C}(\phi_1, B_1)$ and $\hat{C}(\phi_2, B_2)$.
—if $\phi$ is of the form $Qy \in I' \phi'$, where $Q$ is a quantifier, then $\hat{C}(\phi, B)$ iff $\hat{C}(\phi', B \cup y)$. 

**Theorem 2.** For a consistency property $C$, $\hat{C}$ is also a consistency property.

If for a bounded constraint $(\phi, B)$, $\hat{C}(\phi, B)$ holds, we say that $(\phi, B)$ is structurally $C$-consistent.

Note that, in the above definition, recursion for quantification puts the quantifier bound into the free-variable bound of the quantified constraint. This means that a narrowing operator will also have to modify the quantifier bounds in order to achieve structural consistency.

**Example 3.** The bounded constraint $(\exists y \in [0, 1][x^2 + y^2 \leq 1 \land y \geq 0), (x \mapsto [-1, 1], y \mapsto)]$ is structurally tightly consistent, and it will be the result of applying an according narrowing operator to an input such as $(\exists y \in [-2, 2][x^2 + y^2 \leq 1 \land y \geq 0), (x \mapsto [-2, 2], y \mapsto)]$.

Note that Definition 6 is compatible with the usual consistency notions for sets of constraints [Benhamou et al. 1994, 1999]. For example, a set of atomic constraints $\{\phi_1, \ldots, \phi_n\}$ is box-consistent with respect to a box assignment $B$ iff $(\phi_1 \land \cdots \land \phi_n, B)$ is $B\hat{C}$-consistent. In addition, the method for solving constraints with one universally quantified variable by Benhamou and Goualard [2000] computes a special case of Definition 6.

In the following sense, our definition of $\hat{C}$-consistency is optimal (remember that tight consistency is the strongest possible consistency property).

**Theorem 3.** A $\hat{T}C$-consistent bounded constraint $(\phi, B)$, where $\phi$ contains neither conjunctions nor universal quantifiers, is $T'C$-consistent.

**Proof.** We proceed by induction over the structure of constraints. The atomic case trivially holds. Now assume constraints of the following types:

—For a $T'C$-consistent bounded constraint of the form $(\phi_1 \lor \phi_2, B)$, by definition, we have $B = B_1 \cup B_2$ where $T'C(\phi_1, B_1)$ and $T'C(\phi_2, B_2)$. By the induction hypothesis both $(\phi_1, B_1)$ and $(\phi_2, B_2)$ are tightly consistent. So for no box assignment $B'_1 \subset B_1$, we have $B'_1 \supseteq B_1 \cap \llbracket \phi_1 \rrbracket$, and for no box assignment $B'_2 \subset B_2$, we have $B'_2 \supseteq B_2 \cap \llbracket \phi_2 \rrbracket$. Thus also for no box assignment $B' \subset B_1 \cup B_2$, we have $B' \supseteq B \cap (\llbracket \phi_1 \rrbracket \cup \llbracket \phi_2 \rrbracket) = B \cap \llbracket \phi_1 \lor \phi_2 \rrbracket$.

—For a $\hat{T}C$-consistent bounded constraint of the form $(\exists y \in I' \phi', B)$, by definition $T\hat{C}(\phi', B \cup y)$. Thus, by the induction hypothesis $(\phi', B \cup y)$ is tightly consistent. So for no box assignment $B'_p \subset B \cup y$, $B'_p \supseteq B \cup y \cap \llbracket \phi' \rrbracket$. As a consequence also for no box assignment $B_p \subset B$, $B_p \supseteq B \cap \llbracket \exists y \in I' \phi' \rrbracket$, and so $(\exists y \in I' \phi', B)$ is tightly consistent. $\square$

The fact that the above theorem does not hold for constraints with conjunctions is well known [Benhamou and Older 1997]. It is illustrated in Figure 1, where both $\phi_1$ and $\phi_2$ are tightly consistent with respect to the box $B$ (i.e., the larger box encloses the ellipses tightly), but $\phi_1 \land \phi_2$ is only tightly consistent with respect to the smaller box $B'$ (i.e., the smaller, but not the larger box, encloses the intersection of the ellipses tightly).
Fig. 1. Conjunction—Structural tight consistency.

Fig. 2. Universal quantification—Structural tight consistency.

For universal quantification, there is a similar problem: In Figure 2, $\phi$ is tightly consistent with respect to the box $B$, but when considering $\forall y \in I \phi$ one can still narrow $B$ horizontally. So any stronger consistency notion would have to treat universal quantification differently from existential quantification.

5. PRUNING ALGORITHM

In this section, we give an algorithm for pruning quantified constraints that can use an arbitrary algorithm for pruning atomic constraints. This algorithm fulfills the formal properties introduced in Sections 3 and 4.

The algorithm proceeds recursively according to the structure of constraints. For conjunctions, this means the usual: We prune with respect to the individual subconstraints, until this does not result in any further improvements, that is, until we reach a fix-point. For disjunctions, we prune the individual subconstraints and combine the result by taking the smallest box assignment containing the union.

For existentially quantified bounded constraints of the form $(\exists x \in I \phi, B)$, we proceed as shown in Figure 3, where the horizontal axis represents the free-variable bound $B$ (ignoring the component corresponding to the variable $x$) and the vertical axis the quantifier bound $I$. Below and to the left of these axes, we show the changes on the corresponding elements. We recursively prune the bounded subconstraint $(\phi, B^{x \leftarrow I})$ consisting of the subconstraint $\phi$, and the box assignment that is the same as $B$ except that it assigns $I$ to the variable $x$. We use the result to remove these elements from the free-variable bound $B$ and the quantifier bound $I$ for which recursive pruning showed that they do not contain any solution. This results in the new free-variable bound $B'$ and quantifier bound $B'(x)$.

For universally quantified bounded constraints of the form $(\forall x \in I \phi, B)$, if pruning of the subconstraints removes elements from $I$, the whole constraint is false, and we can replace $B$ by the empty set (Figure 4). If no such elements are removed, then we just prune $B$ accordingly (Figure 5).
Fig. 3. Existential pruning.

Fig. 4. Universal pruning.

Fig. 5. Universal pruning.
To formalize the above, we let \( \text{fix} \) be a partial function such that, for a set of functions \([f_1, \ldots, f_n]\), for all \( a \), \( \text{fix}([f_1, \ldots, f_n](a)) \) is a fixpoint of applying \([f_1, \ldots, f_n]\) to \( a \), if such a fixpoint exists, and is undefined, otherwise. Now we have:

**Definition 7.**

— For atomic \( \phi, N_A(\phi, B) = \overline{A}(\phi, B) \),

— \( N_A(\phi_1 \land \phi_2, B) = \text{fix}(N_1, N_2)(\phi_1 \land \phi_2, B) \),

where \( N_1(\phi_1 \land \phi_2, B) = (\phi'_1 \land \phi_2, B') \), where \( (\phi'_1, B') = N_A(\phi_1, B) \), and \( N_2(\phi_1 \land \phi_2, B) = (\phi_1 \land \phi'_2, B') \), where \( (\phi'_2, B') = N_A(\phi_2, B) \).

— \( N_A(\exists x \in I \ \phi, B) = (\exists x \in B' (x) \phi', B') \),

where \( (\phi', B') = N_A(\phi, B'') \)

— \( N_A(\forall x \in I \ \phi, B) = (\forall x \in I \ \phi', D) \),

where \( (\phi', B') = N_A(\phi, B'') \)

and \( d \in D \) iff for all \( r \in I, d \leq B' \).

**Example 4.** For the input \( (\exists y \in [-2, 2] [x^2 + y^2 \leq 1 \land y \geq 0], \{x \mapsto [-2, 2], y \mapsto \}) \) already used in Example 3, a narrowing operator based on tight consistency applies itself recursively to \((x^2 + y^2 \leq 1 \land y \geq 1, \{x \mapsto [-2, 2], y \mapsto [-2, 2]\})\). Repeated applications of the atomic narrowing operator—until a fixpoint is reached—will create the constraint \((x^2 + y^2 \leq 1 \land y \geq 0, \{x \mapsto [-1, 1], y \mapsto [0, 1]\})\). As a final result, we get \((\exists y \in [0, 1] [x^2 + y^2 \leq 1 \land y \geq 0], \{x \mapsto [-1, 1], y \mapsto \})\).

**Example 5.** For the input \( (\forall x \in [-2, 2] x \geq 0, \{x \mapsto \}) \), the algorithm will first narrow \((x \geq 0, \{x \mapsto [-2, 2]\})\) to \((x \geq 0, \{x \mapsto [0, 2]\})\) and then create \((\forall x \in [-2, 2] x \geq 0, \emptyset)\), indicating that the constraint is false.

In the rest of this section, we will use the results of Section 3 and 4 for studying the properties of the introduced pruning algorithm. Readers not interested in these formal properties can directly jump to Section 6 for an according solver.

Note that the fixed-point operator \( \text{fix} \) could result in a partial function, that is, the algorithm could fail to terminate. For ensuring termination, we require that pruning atomic constraints eventually terminates even when intermingled with shrinking of the free-variable bound by other operations:

**Definition 8.** A narrowing operator \( N \) is finitely contracting iff there is no infinite sequence \((\phi_1, B_1), (\phi_2, B_2), \ldots \) for which for all \( k \in \mathbb{N} \), for \( (\phi', B') = N(\phi_k, B_k), \phi_{k+1} = \phi' \) and \( B_{k+1} \) is a strict subset of \( B' \).

This property usually holds for practical implementations, because of the finiteness of floating point numbers.

**Lemma 1.** If \( \overline{A} \) is a finitely contracting narrowing operator, then \( N_A \) is a total function.
Proof. We assume that $\mathcal{A}$ is finitely contracting but $N_A$ is not total. This can only happen if fix($\{N_1, N_2\}$,$\phi_1 \land \phi_2$, $B$) is undefined. Consider the sequence $(\phi_1 \land \phi_2, B_1)$, $(\phi_1 \land \phi_2, B_2)$, ... of bounded constraints created by repeated applications of $N_1$ and $N_2$. Here, $(\phi_1, B_1), (\phi_1, B_2), ...$ is an infinite sequence as in Definition 8. So $N_A$ is not finitely contracting, and by induction also $\mathcal{A}$ is not finitely contracting—a contradiction.

The algorithm fulfills the properties needed by the formal framework of narrowing operators and gives a unique result (despite the nonunique definition of fixpoint operator):

Theorem 4. For every finitely contracting (atomic) outer narrowing operator $\mathcal{A}$, $N_A$ is a unique outer narrowing operator.

Proof. Contractance and idempotence hold by easy induction. For proving the ground case of atomic constraints holds by definition. Now we have:

—Obviously the composition of two narrowing operators is also a narrowing operator. So, for constraints of the form $\phi_1 \land \phi_2$ we just need to show that both $N_1$ and $N_2$ are outer narrowing. For $(\phi_1', B_1') = N(\phi_1, B)$ and $(\phi_2', B_2') = N(\phi_2, B)$, by the induction hypothesis $B_1' \supseteq B \cap \|\phi_1\|$ and $B_2' \supseteq B \cap \|\phi_2\|$. Thus also $B_1' \cap B_2' \supseteq B \cap \|\phi_1\| \cap \|\phi_2\| = B \cap \|\phi_1 \land \phi_2\|$. The induction step for soundness and monotonicity is easy.

—For constraints of the form $\phi_1 \lor \phi_2$, for $(\phi_1', B_1') = N(\phi_1, B)$ and $(\phi_2', B_2') = N(\phi_2, B)$, by the induction hypothesis $B_1' \supseteq B \cap \|\phi_1\|$ and $B_2' \supseteq B \cap \|\phi_2\|$. Thus also $B_1' \cup B_2' \supseteq B_1' \cup B_2' \supseteq B \cap (\|\phi_1\| \cup \|\phi_2\|) = B \cap (\|\phi_1 \lor \phi_2\|)$. The induction step for soundness and monotonicity is easy.

—For constraints of the form $\exists x \in I \phi$, the induction step for outer narrowing is easy. For soundness, we have to prove that $\|\exists x \in I \phi\| \cap B' = \|\exists x \in B'(x) \phi'\| \cap B'$, where $B' = N_A(\phi, B^\bot_x)$. Now, by the outer narrowing property $B' \supseteq B \cap \|\phi\|$, and so $\|\exists x \in I \phi\| \cap B' = \|\exists x \in B'(x) \phi\| \cap B'$. This is equal to $\|\exists x \in B'(x) \phi'\| \cap B$, because by the induction hypothesis $\|\phi'\| \cap B' = \|\phi\| \cap B'$. For constraints of the form $\forall x \in I \phi$, for outer narrowing we have to prove that $D \supseteq B \cap \|\forall x \in I \phi\|$, where $D$ is defined as in the corresponding rule of Definition 7. So we assume a variable assignment $d$ that is both in $B$ and $\|\forall x \in I \phi\|$, and prove that $d \in D$. This means that we have to prove that for all $r \in I, d^r_x \in B'$, where $(\phi', B') = N_A(\phi, B^\bot_x)$. This is clearly the case by the semantics of universal quantification and the induction hypothesis.

For soundness, we have to prove that $\|\forall x \in I \phi\| \cap D = \|\forall x \in I \phi\| \cap \{d\}$, where $D$ is as above. By the quantifier semantics it suffices to prove that $\|\phi\| \cap D^\bot_x = \|\phi'\| \cap D^\bot_x$. This holds, because for all variable assignments $d \in D$, for all $r \in I, d^r_x \in B'$, and moreover, by the induction hypothesis $\|\phi'\| \cap B' = \|\phi\| \cap B'$.

The uniqueness of the fixpoint operator follows from contractance and monotonicity of narrowing operators [Cousot and Cousot 1977; Apt 1999].
By easy induction, we also get:

**Theorem 5.** $N_A$ ensures $\hat{A}$-consistency.

By applying Theorem 1, we get a corresponding inner narrowing operator $N_A$ from $\hat{N}_A$. Note, however, that $\hat{N}_A$ and $N_A$ do not commute, and $\hat{N}_A \circ N_A$ is not idempotent.

As for the classical conjunctive case, the complexity of the algorithm in a floating-point implementation is polynomial in the problem dimension (the number of floating point numbers that one can remove from the quantification bounds is polynomial). So, as desired, pruning is efficient compared to expensive exhaustive search, and even more so compared to the doubly exponential complexity of symbolic solvers [Collins 1975; Caviness and Johnson 1998].

Note however, that the cardinality of the usual float-point representations is so high that, in addition, one should take care that the worst-case complexity is not reached in practice.

6. SOLVER

Now a branch-and-prune algorithm can do pruning according to Definition 7, and branching means replacing subconstraints of the form $\forall x \in I \phi$ by $\forall x \in I_1 \phi \land \forall x \in I_2 \phi$, or subconstraints of the form $\exists x \in I \phi$ by $\exists x \in I_1 \phi \lor \exists x \in I_2 \phi$, where $I = I_1 \cup I_2$. We assume branching to be fair, in the sense that every bound will eventually be split (finding such a strategy is easy, but finding an optimal branching strategy is highly nontrivial).

For disproving a closed constraint $\phi$ we repeatedly branch and prune the input constraint $(\phi, T)$ until $T$ is pruned to $F$ (remember that $F$ is an abbreviation for the empty box assignment). For proving, we do the same on the opposite of $\phi$. For computing, the truth-value we do both things in parallel.

A solver for open quantified constraints could for example use the following specification:

Given:
— A quantified constraint $\phi$ with $n$ free variables,
— $B \subseteq \mathbb{R}^n$,
— $\epsilon \in \mathbb{R}^+$

Find: Sets of boxes $Y$, $N$ such that
— all elements of $Y$ are in the solution set of $\phi$,
— all elements of $N$ are not in the solution set of $\phi$,
— $\text{Vol}(B \setminus Y \setminus N) \leq \epsilon$

This specification allows the user to decide on the trade-off between run-time and precision. When choosing a large $\epsilon$, only a small part of the solution set of $\phi$ within $B$ will be characterized by $Y$ and $N$, when choosing an $\epsilon$ close to zero, almost the whole set will be characterized.

An according solver is an easy extension of the closed case that would record the boxes that narrowing of the input constraint proved to be not in the solution set, and narrowing of the opposite of the input constraint proved to be in the solution set. Furthermore, in addition to branching the quantified variables it
also has to branch the free-variable bound. We call the resulting algorithm, a parallel branch-and-prune solver.

For discussing termination of such a branch-and-prune solver, it is important to see, that the problem of computing truth-values is undecidable. So it is impossible to find an algorithm that terminates always. A solution to this problem is to require termination for all, except very special cases.

For this, we observe that the truth-value/solution set of a quantified constraint can be numerically unstable [Ratschan 2002d]. An example is the quantified constraint \( \exists x \in [-1, 1] \ - x^2 \geq 0 \) which is true, but becomes false under arbitrarily small positive perturbations of the constant 0. As a consequence, it is not possible to design an algorithm based on approximation that will always terminate (with a correct result). Note that this situation is similar for most computational problems of continuous mathematics (e.g., solving linear equations, solving differential equations). However, as in these cases, most inputs are still numerically stable (in fact, in a certain, realistic model this is the case with probability one [Ratschan 2001b]). One can even argue that, philosophically speaking, the stable problems are exactly the problems that model real-life problems in a meaningful way.

It is beyond the scope of this article to present all the formal details for characterizing stable quantified constraints and we will introduce the necessary concepts in a semi-formal way. For this, we replace the discrete notion of truth of a quantified constraint by a continuous notion [Ratschan 2002d]. We interpret universal quantifiers as infimum operators, existential quantifiers as supremum operators, conjunction as minimum, disjunction as maximum, atomic constraints of the form \( f > g \) or \( f \geq g \) as the function denoted by \( f - g \), and atomic constraints of the form \( f < g \) or \( f \leq g \) as the function denoted by \( g - f \). We call the result the degree of truth of a quantified constraint and denote it by \( \| \phi \| \) for any constraint \( \phi \). This function assigns to every variable assignment a real value that is independent of the variables that are not free in \( \phi \). The idea is that the degree of truth is greater or equal zero for variable assignments that make \( \phi \) true, and less or equal zero for variable assignments that make \( \phi \) false. One can prove [Ratschan 2002d] that the problem of computing the truth value of a closed quantified constraint is numerically stable (or: stable) iff its degree of truth is nonzero.

We assume that the given narrowing operator for atomic constraints eventually succeeds for stable inputs:

**Definition 9.** An outer narrowing operator \( \overline{A} \) is converging iff for all atomic constraints \( \phi \) and sequences \( B^0 \supseteq B^1 \supseteq \cdots \) such that

- for all \( i \in \mathbb{N} \), \( B^i \supseteq B^{i+1} \),
- and \( \bigcap_{k \in \mathbb{N}} B^k = \{ d \} \), where the degree of truth of \( \phi \) at the variable assignment \( d \) is negative,

there is a \( k \), such that for all \( l \geq k \), the free-variable bound of \( \overline{A}(\phi, B^l) \) is empty.

Note that this trivially holds for tight consistency. For hull and box consistency it is necessary that the number of bits used in floating point computation

Solving Quantified Inequality Constraints over Real Numbers • 737
is high enough for a given sequence of boxes. For box consistency, in addition, interval evaluation has to converge for all atomic constraints. This is the case if they only contain continuous functions such as +, ×, exp, and sin, on which we can implement interval evaluation, see Theorem 2.2 [Jaulin et al. 2001].

However, the above property is in general impossible to fulfill for any narrowing operator based on fixed-precision floating-point arithmetic. Still, one can easily overcome this difficulty, by using sufficient precision [Revol and Rouillier 2005] (see Theorem 2.1.5 [Neumaier 1990]). Moreover, the application of our method to real-life problems has shown that usual double-precision floating-point arithmetic almost always suffices in practice.

**Lemma 2.** Let $\mathcal{A}$ be a converging outer narrowing operator and let the sequence $(\phi_1^1, B^1), (\phi_2^2, B^2), \ldots$ be such that

- for all $i \in \mathbb{N}$, $B^i \supseteq B^{i+1}$, and
- $\bigcap_{i \in \mathbb{N}} B^i = \{ d \}$ such that the degree of truth of $\phi$ at the variable assignment $d$ is negative,
- for all $i \in \mathbb{N}$, $\phi^{i+1}$ results from $\phi^i$ by branching, and
- for all $\varepsilon > 0$, there is a $k$ such that for all $l \geq k$, the volume of all quantification sets in $\phi^i$ is less or equal $\varepsilon$.

Then there is a $k$ such that for all $l \geq k$, the free-variable bound of $\mathcal{N}_A(\phi^i, B^i)$ is empty.

**Proof.** We proceed by induction over the structure of the constraint $\phi^1$. For atomic constraints the lemma holds because $\mathcal{A}$ is converging. Now consider the following cases:

- For constraints of the form $\phi_1 \land \phi_2$, $\| \phi_1 \land \phi_2 \|^\phi(d) = \min(\| \phi_1 \|^\phi(d), \| \phi_2 \|^\phi(d))$ being negative implies that either $\| \phi_1 \|^\phi(d)$ is negative or $\| \phi_2 \|^\phi(d)$ is negative. Therefore, at least one of the sequences $(\phi_1^1, B^1), (\phi_2^2, B^2), \ldots$ and $(\phi_2^2, B^2), (\phi_2^2, B^2), \ldots$, where $\phi_1^1$ is the subconstraint of $\phi^i$ corresponding to $\phi_1$ and $\phi_2^2$ is the subconstraint of $\phi^i$ corresponding to $\phi_2$, fulfills the preconditions of the induction hypothesis. Let $r \in \{1, 2\}$ be the number of this sequence. Then, there is a $k$, such that for all $k \geq l$, the free-variable bound of $\mathcal{N}_A(\phi^i, B^i)$ is empty. Thus, by definition of $\mathcal{N}_A$, the corresponding free-variable bound is also empty in the original sequence.

- For constraints of the form $\phi_1 \lor \phi_2$, $\| \phi_1 \lor \phi_2 \|^\phi(d) = \max(\| \phi_1 \|^\phi(d), \| \phi_2 \|^\phi(d))$ being negative implies that both $\| \phi_1 \|^\phi(d)$ and $\| \phi_2 \|^\phi(d)$ are negative. Therefore both sequences $(\phi_1^1, B^1), (\phi_2^2, B^2), \ldots$ and $(\phi_2^2, B^2), (\phi_2^2, B^2), \ldots$, where $\phi_1^1$ is the subconstraint of $\phi^i$ corresponding to $\phi_1$ and $\phi_2^2$ is the subconstraint of $\phi^i$ corresponding to $\phi_2$, fulfill the preconditions of the induction hypothesis. As a consequence, there is a $k_1$, such that for all $l \geq k_1$, the free-variable bound of $\mathcal{N}_A(\phi^i, B^i)$ is empty, and there is a $k_2$, such that for all $l \geq k_2$, the free-variable bound of $\mathcal{N}_A(\phi^i, B^i)$ is empty. Thus, by definition of $\mathcal{N}_A$, for all

---

3This item formalizes the notion of a fair branching strategy.
\[ l \geq \max\{k_1, k_2\} \] the free-variable bound of the \( l \)th element in the original sequence is empty.

—Constraints of the form \( \forall x \in I \, \phi' \), are replaced by branching into the form
\[ \forall x \in I_1 \, \phi' \land \ldots \land \forall x \in I_k \, \phi' \]. Since the degree of truth of \( \forall x \in I \, \phi' \) at \( d \) is negative, by definition of infimum, there is a \( b \in I \) for which the degree of truth of \( \phi' \) at \( d \times b \) is negative. Consider the sequence for which the \( i \)th element consists of the branch of \( \phi' \) that contains \( d \times b \), and of \( B' \). This sequence fulfills the preconditions of the induction hypothesis, and as a consequence there is a \( k \), such that for all \( k \geq l \), the \( k \)th free-variable bound in this sequence is empty. Thus, by definition of \( N_A \), the corresponding free-variable bound is also empty in the original sequence.

—Constraints of the form \( \exists x \in I \, \phi' \), are replaced by branching into the form
\[ \exists x \in I_1 \, \phi' \lor \ldots \lor \exists x \in I_k \, \phi' \]. Since the degree of truth of \( \exists x \in I \, \phi' \) at \( d \) is negative, by definition of supremum, for all \( b \in I \) the degree of truth of \( \phi' \) at \( d \times b \) is negative. This means that each sequence for which the \( i \)th element consists of a branch of \( \phi' \) and of \( B' \) fulfills the preconditions of the induction hypothesis, and as a consequence there is a \( k \), such that for all \( k \geq l \), the \( k \)th free-variable bound in this sequence is empty. Thus, by definition of \( N_A \), the corresponding free-variable bound is also empty in the original sequence. 

This implies:

**Lemma 3.** A branch-and-prune algorithm for disproving a closed constraint succeeds iff the degree of truth of the input is negative.

From Lemma 3 and its dual version, we get:

**Theorem 6.** For stable inputs, a parallel branch-and-prune solver eventually computes the truth value for closed inputs, and fulfills the above solver specification for open inputs.

### 7. EFFICIENT IMPLEMENTATION

In this section, we show how to extend the basic solver for allowing an efficient implementation.

#### 7.1 Connectives with Arbitrary Arity

The first step for arriving at an efficient implementation, is to treat conjunctions and disjunctions not as binary operators, but as operators with arbitrary arity (see Definition 7). It is easy to adapt the according algorithms and proofs.

#### 7.2 Quantifier Blocks

Treat quantifiers of the same kind in blocks. That is, quantify over a whole vector of variables at once, using quantifier bounds that are boxes instead of intervals. This allows more flexibility for branching.

#### 7.3 Removing Empty Disjunctive Branches

Pruning might show that one of the branches of a disjunction has an empty solution set. Currently (see Definition 7), this information is forgotten. In order
to prevent this, we simply remove the corresponding subconstraint in such as case.

7.4 Combination with Negated Constraint

The parallel branch-and-prune solver developed in Section 6 independently works on proving and disproving the input constraint. For proving, it employs a branch-and-prune procedure on the negation of the input, for disproving, on the input itself. Working on the input and its negation separately has the disadvantage that information computed for one is not used for the other. In order to improve this, we do both on the same constraint, repeatedly negating it in between. The result is Algorithm 1 for closed constraints and Algorithm 2 for open constraints. Here “Branch” and “Prune” do the obvious, except that in the second algorithm “Branch” takes a bounded constraint and returns a set of bounded constraints that either

—contains two elements whose union of bounds is equal to the input bound, or
—contains one element with a quantifier split into a conjunction (in the case of a universal quantifier), or a disjunction (in the case of an existential quantifier).

Algorithm 1 Combined Solver for Closed Constraints

\[ \text{Input: a closed quantified constraint } \phi \]
\[ \text{Output: the truth-value of } \phi \]

\begin{verbatim}
unknown ← T
while unknown do
    neg ← T
    φ′ ← F
    (φ, unknown) ← Prune(φ, unknown)
    while unknown and φ ≠ φ′ do
        φ′ ← φ
        neg ← not neg
        if neg then
            (¬φ, unknown) ← Prune(¬φ, unknown)
        else
            (φ, unknown) ← Prune(φ, unknown)
        end if
    end while
    if unknown then
        φ ← Branch(φ)
    end if
end while
return neg
\end{verbatim}

THEOREM 7. Algorithms 1 and 2 are correct and terminate for stable inputs.

PROOF. We just show the proof for Algorithm 1, the other case is similar.

If neg is false at the return statement, then pruning succeeded for φ, and so it is false. Otherwise, neg is true and so pruning succeeded for ¬φ and φ is true.

The innermost while loop always terminates because there are only finitely many floating point numbers, and so the narrowing operator can only do finitely
many changes after which $\phi = \phi'$. Termination of the overall loop again is a consequence of Lemma 2. □

Algorithm 2 Combined Solver for Open Constraints

| Input:  | a quantified constraint $\phi$, a box $B$, and a positive real number $\varepsilon$ |
|-----------------------------------------------|
| Output: | $Y$, a list of boxes on which $\phi$ is true, and $N$, a list of boxes on which $\phi$ is false, such that the volume of $B \setminus \bigcup Y \setminus \bigcup N$ is less than $\varepsilon$. |

$U \leftarrow \{(\phi, B)\}$
$Y \leftarrow \emptyset$
$N \leftarrow \emptyset$

while $\text{Vol}(\bigcup U) \geq \varepsilon$ do
  choose and remove a bounded constraint $(\phi_U, B_U)$ from $U$
  neg $\leftarrow \text{T}$
  $(\phi_U', B_U') \leftarrow (\phi_U, B_U)$
  $(\phi_U', B_U') \leftarrow \text{Prune}(\phi_U, B_U)$
  $Y \leftarrow Y \cup B_U' \setminus B_U$

  while $B_U$ is non-empty and $(\phi_U, B_U) \neq (\phi_U', B_U')$ do
    $(\phi_U', B_U') \leftarrow (\phi_U, B_U)$
    neg $\leftarrow \text{not neg}$
    if neg then
      $(\neg \phi, B_U) \leftarrow \text{Prune}(\neg \phi, B_U)$
      $Y \leftarrow Y \cup B_U' \setminus B_U$
    else
      $(\phi, B_U) \leftarrow \text{Prune}(\phi, B_U)$
      $N \leftarrow N \cup B_U' \setminus B_U$
    end if
  end while

  if $\text{Vol}(\bigcup U) \geq \varepsilon$ then
    $U \leftarrow U \cup \text{Branch}(\phi, B_U)$
  end if
end while

7.5 Branching Strategy

For arriving at an implementation, a good strategy for choosing a (free-variable or quantification) bound for branching is crucial. We did not yet try to arrive at a theoretically well founded or even optimal strategy. However, the following approach seems to work well in practice:

In Algorithm 2, for every element of the set $U$ we store the level of the last splitting done (viewing constraints as trees), and for each conjunction or disjunction (of the original constraint, not the ones created by branching) the last branched subconstraint. We choose the bounded constraint $(\phi_U, B_U)$ with largest $B_U$ and branch a subconstraint

— that is one level below the last splitting, or (if this is impossible) on the highest level,
— with the maximum volume of the quantifier bound for conjunctions or disjunctions created by branching, which is
— the next subconstraint for all other conjunctions and disjunctions.
7.6 Incremental Disjunctive Pruning

The disjunctive case in Definition 7, has the disadvantage that it stores the intermediate results of narrowing all subconstraints until computing the box union $\cup$ of all of them. We can avoid this by using an incremental algorithm instead that intermingles the recursive calls with taking the box union of the result.

7.7 Shortcut for Disjunctions

When doing incremental pruning of disjunctions, we might detect that the result will be the input box, before inspecting all subconstraints. We can leave the according loop already at this point.

7.8 Reusing Dual Information

Sometimes an atomic narrowing operator computes the information that narrowing the negation will fail. For example, assume that a bound $[-2, 2]$ of a univariate atomic constraint has been pruned to $[-1, 1]$. For many narrowing operators, this implies that the constraint does not hold on $-1$ and $1$. Therefore, we cannot use the opposite constraint to prune $[-1, 1]$ further.

In the case of box consistency, which we use in our implementation, one works on atomic constraints one variable after the other. For each atomic subconstraint/variable pair, we can get the information that pruning or pruning on the opposite will not succeed, that is that this subconstraint or its opposite is consistent. We store this information by assigning to each atomic subconstraint two sets of variables (the mark and the opposite mark). Furthermore, in order to reflect the situation for box consistency, we assume that consistency also takes into account variables.

Definition 10. We call a bounded constraint $(\phi, B)$ correctly marked iff for every atomic subconstraint $\phi'$ of $\phi$, every box assignment $B'$ that results from $B$ by replacing all variables of $B$ bounded by $\phi$ by these bounds, and every variable $v$ in the mark of $\phi'$, $(\phi', B', v)$ is consistent.

Sometimes we change the bound of a variable. In this case, some of the marks might not stay valid. For example, take a constraint of the form $\exists x \in I_x, \exists y \in I_y \ [\phi_1 \land \phi_2]$, where $\phi_1$ is an atomic constraint that contains both variables $x$ and $y$, but $\phi_2$ is an atomic constraint that only contains $y$. After pruning, the marks and opposite marks of both $\phi_1$ and $\phi_2$ can be set to $\{x, y\}$, indicating that no further pruning is possible. After branching the quantifier of $y$, we have to remove the marks of both copies of $\phi_1$, but we can keep the marks of the copies of $\phi_2$ since $\phi_2$ contains no variable affected by the branching. Therefore, further pruning will know that calls to the atomic narrowing operator of $\phi_2$ are not necessary.

So denote by Notify($\phi, V$) the result of replacing all marks of subconstraints that contain variables in $V$ by the empty set.

Lemma 4. For every bounded constraint $(\phi, B)$ that is correctly, and for every box $B'$ such that for all $v \notin V$, $B'(v) = B(v)$, Notify($\phi, V$), $B'$ is correctly marked.
PROOF. All subconstraints of Notify(φ, V) that contain V have empty marks and Definition 10 requires nothing from them. For all bounded subconstraints that do not contain V, the corresponding bound inherited from B is the same as the one from B', and therefore one does not have to change the corresponding marks.

Definition 11. A narrowing operator preserves marks iff after applying it to a bounded constraint that is correctly marked the result is again correctly marked.

We assume that we have an atomic narrowing operator that preserves marks and opposite marks. Of course, this can be easily done by always setting the marks to the empty set. But, in our implementation, we will try to set them as large as possible.

Now, whenever applying the narrowing operator on atomic constraint/variable pairs we check the marks before. For more complicated constraints, we have to adapt the narrowing operator of Definition 7 such that it updates the marks accordingly. This means that for disjunctions, if the box union is different from the boxes resulting from narrowing an individual constraint, then we have to do notification on it. In a similar way, for conjunctions, when computing the fixpoint, every time narrowing succeeds for a subconstraint, we have to do notification for all other subconstraints. Clearly, by Lemma 4, the adapted narrowing operator preserves marks and opposite marks.

Also for branching we have to do according notification for the changed variables. When using adapted branching and pruning in Algorithms 1 and 2, we set all marks to the empty set at the beginning and preserve marks and opposite marks throughout.

Clearly, the resulting solver does less calls to the narrowing operator for projection constraints than the original one. Furthermore, this also gives an improvement for the case of unquantified constraints for which the solution set does not consist of finitely many, isolated solutions.

8. TIMINGS

We have implemented the algorithm described in this paper using as the atomic narrowing operator an algorithm [Hong and Stahl 1994] that computes something similar to box consistency. In Table I, we compare this implementation described in this paper with an implementation of the older algorithm using “cylindrical box decomposition” [Ratschan 2002a] which also used the same atomic narrowing operator. Here the heading “0.1 old” refers to running the older algorithm until it leaves not more than a fraction of 1/10 of the solution space unknown, the heading “first” refers to running the current algorithm till it finds the first true box, and the heading “0.1” refers to running the current algorithm in the same way as the older one.

Columns headed by “Time” list the time in seconds needed to solve the problem where ε means less than a second and ∞ more than 10 minutes; columns headed by “Hits” list the number of calls to the atomic narrowing operator for the new algorithms (this is a good efficiency measure because it ignores
Table I. Comparison with Cylindrical Box Decomposition

<table>
<thead>
<tr>
<th>Example</th>
<th>Time</th>
<th>Boxes</th>
<th>Time</th>
<th>Hits</th>
<th>Boxes</th>
<th>Time</th>
<th>Hits</th>
<th>Boxes</th>
</tr>
</thead>
<tbody>
<tr>
<td>McCallum_2_1</td>
<td>ε</td>
<td></td>
<td>ε</td>
<td>22</td>
<td></td>
<td>ε</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>McCallum_2_2</td>
<td>ε</td>
<td></td>
<td>ε</td>
<td>157</td>
<td></td>
<td>ε</td>
<td>157</td>
<td></td>
</tr>
<tr>
<td>McCallum_2_3</td>
<td>ε</td>
<td></td>
<td>ε</td>
<td>2854</td>
<td></td>
<td>ε</td>
<td>2854</td>
<td></td>
</tr>
<tr>
<td>McCallum_2_4</td>
<td>ε</td>
<td></td>
<td>ε</td>
<td>153</td>
<td></td>
<td>ε</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>anderson2</td>
<td>ε</td>
<td>391</td>
<td>10.95</td>
<td>150343</td>
<td>575</td>
<td>ε</td>
<td>3372</td>
<td>250</td>
</tr>
<tr>
<td>anderson2_proj</td>
<td>16.81</td>
<td>3466</td>
<td>ε</td>
<td>1129</td>
<td>21</td>
<td>2.69</td>
<td>21058</td>
<td>620</td>
</tr>
<tr>
<td>circle</td>
<td>ε</td>
<td>42</td>
<td>ε</td>
<td>19</td>
<td>10</td>
<td>ε</td>
<td>60</td>
<td>26</td>
</tr>
<tr>
<td>robust-1</td>
<td>ε</td>
<td>23</td>
<td>ε</td>
<td>13</td>
<td>3</td>
<td>ε</td>
<td>121</td>
<td>16</td>
</tr>
<tr>
<td>robust-2</td>
<td>∞</td>
<td>90.74</td>
<td>663011</td>
<td>1266</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>robust-3</td>
<td>∞</td>
<td>ε</td>
<td>344</td>
<td>5</td>
<td>ε</td>
<td>936</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>robust-5</td>
<td>∞</td>
<td>ε</td>
<td>376</td>
<td>11</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>robust-6</td>
<td>5.4</td>
<td>112</td>
<td>ε</td>
<td>34</td>
<td>3</td>
<td>256.14</td>
<td>564326</td>
<td>4309</td>
</tr>
<tr>
<td>silaghi1</td>
<td>ε</td>
<td>19</td>
<td>ε</td>
<td>25</td>
<td>6</td>
<td>ε</td>
<td>25</td>
<td>6</td>
</tr>
<tr>
<td>termination</td>
<td>∞</td>
<td>ε</td>
<td>90</td>
<td>ε</td>
<td>90</td>
<td>ε</td>
<td>90</td>
<td>ε</td>
</tr>
</tbody>
</table>

Implementation details, and because atomic narrowing takes the largest part of the overall runtime; and columns headed by “Boxes” list the total number of boxes created (true, false, and unknown boxes).

Examples starting with McCallum [1993] are from computational geometry, asking whether a solution to a given system of inequalities exists. The example “circle” simply computes a description of the unit disc $x^2 + y^2 \leq 1$, “silaghi” is a system of inequalities describing the geometry of a simple mechanical problem [Silaghi et al. 2001], the example “anderson2” is a system of polynomial inequalities from control engineering [Anderson et al. 1975; Abdallah et al. 1996; Garloff and Graf 1999], the example “anderson2_proj” computes the projection of the former into two-dimensional space, the example “termination” proves the termination of a certain term-rewrite system [Collins and Hong 1991]. All examples starting with “robust” are taken from robust control [Dorato 2000; Fiorio et al. 1993; Malan et al. 1997; Dorato et al. 1997; Jaulin and Walter 1996; Jaulin et al. 2002] and laid out on a website [Ratschan 2001a].

For all examples, we push quantifiers inside as much as possible by transforming $\forall (\phi_1 \land \phi_2)$ to $(\forall \phi_1) \land (\forall \phi_2)$ and the dual for existential quantification. Furthermore, in the few cases, where no quantification bounds were available, we introduced new, very large ones.

As one can see, except for the example “robust-6”, the number of generated boxes is much smaller for the new algorithm. For the examples where it is possible to make a clear comparison of the run-times, the new algorithm is also faster (again with the exception of “robust-6”). An analysis of the behavior for the outlier “robust-6” shows that in this case our branching heuristics do not work very well—alternative heuristics show a much better behavior. This suggests that a detailed study of such heuristics—expanding results for a simpler branch-and-bound approach [Ratschan 2002e] can still result in large improvements of the method.

In Table II, we show the results of some of the algorithm improvements introduced in Section 7. We chose these that need a nontrivial implementation
Table II. Comparison of Improvements

<table>
<thead>
<tr>
<th>Example</th>
<th>No Reuse/No Sh.cut</th>
<th>No reuse/Sh.cut</th>
<th>Reuse/No Sh.cut</th>
<th>Reuse/Sh.cut</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time Hits</td>
<td>Time Hits</td>
<td>Time Hits</td>
<td>Time Hits</td>
</tr>
<tr>
<td>McCallum_2.1</td>
<td>ε 22</td>
<td>ε 22</td>
<td>ε 22</td>
<td>ε 22</td>
</tr>
<tr>
<td>McCallum_2.2</td>
<td>ε 182</td>
<td>ε 169</td>
<td>ε 150</td>
<td>ε 157</td>
</tr>
<tr>
<td>McCallum_2.3</td>
<td>ε 10059</td>
<td>ε 3005</td>
<td>ε 3780</td>
<td>ε 2854</td>
</tr>
<tr>
<td>McCallum_2.4</td>
<td>ε 163</td>
<td>ε 165</td>
<td>ε 138</td>
<td>ε 153</td>
</tr>
<tr>
<td>anderson2</td>
<td>ε 4034</td>
<td>ε 4253</td>
<td>ε 2964</td>
<td>ε 3372</td>
</tr>
<tr>
<td>anderson2.proj</td>
<td>8.49 78985</td>
<td>8.54 65401</td>
<td>7.57 55443</td>
<td>7.54 55244</td>
</tr>
<tr>
<td>circle</td>
<td>ε 95</td>
<td>ε 95</td>
<td>ε 60</td>
<td>ε 60</td>
</tr>
<tr>
<td>robust-1</td>
<td>ε 138</td>
<td>ε 139</td>
<td>ε 116</td>
<td>ε 121</td>
</tr>
<tr>
<td>robust-2</td>
<td>∞ ∞</td>
<td>∞ ∞</td>
<td>∞ ∞</td>
<td>∞ ∞</td>
</tr>
<tr>
<td>robust-3</td>
<td>ε 1623</td>
<td>ε 1115</td>
<td>ε 1003</td>
<td>ε 936</td>
</tr>
<tr>
<td>robust-5</td>
<td>∞ ∞</td>
<td>∞ ∞</td>
<td>∞ ∞</td>
<td>∞ ∞</td>
</tr>
<tr>
<td>robust-6</td>
<td>147.89 637216</td>
<td>285.52 702734</td>
<td>69.8 255163</td>
<td>256.14 564326</td>
</tr>
<tr>
<td>silaghi1</td>
<td>ε 31</td>
<td>ε 27</td>
<td>ε 27</td>
<td>ε 25</td>
</tr>
<tr>
<td>termination</td>
<td>ε 282</td>
<td>ε 102</td>
<td>ε 156</td>
<td>ε 90</td>
</tr>
</tbody>
</table>

effort or these for which it is not totally clear that they improve the efficiency of the algorithm. These are the ones described in Section 7.7 and Section 7.8.

One can conclude that reusing dual information (Section 7.8) always improves the algorithm, sometimes significantly. This phenomenon also occurs for examples that do not contain any quantifiers. On the other hand, for taking shortcuts for disjunctions (Section 7.7), the influence on efficiency is inconclusive, especially when combined with the former improvement.

Note that often one can get even better run-times by symbolically eliminating linearly quantified variables before [Weispfenning 1988; Loos and Weispfenning 1993]. Unfortunately, in some cases the result can be very large, destroying the positive effect of the eliminated variable. Future work will investigate this behavior in detail.

9. RELATION TO CLASSICAL ALGORITHMS

Tarski [1951] showed that quantified constraints over the reals with equality and inequality predicates, multiplication and addition admit quantifier elimination. Adding additional function symbols (e.g., sin, tan), usually removes this property [Richardson 1968; van den Dries 1988; Macintyre and Wilkie 1996]. Using the method in this article one can still compute useful information for these cases, provided that the input is numerically stable.

The complexity bound supplied by Tarski's method has been improved several times [Collins 1975; Renegar 1992; Basu et al. 1994]—but the problem is inherently doubly exponential [Davenport and Heintz 1988; Weispfenning 1988] in the number of quantifier alternations, and exponential in the number of variables.

The only general algorithm for which a practically useful implementation exists, is the method of quantifier elimination by cylindrical algebraic decomposition [Collins 1975]. This algorithm employs similar branching as the algorithm presented in this article. However, its branching operation is much more complicated because it branches into a finite set of truth-invariant cells, that
is, into pieces whose value can be computed by evaluation on a single sample point. For being able to do this, its quantifier bounds can depend on the free variables, and branching is done based on information from projection polynomials. For implementing these operations one needs expensive real algebraic number computations.

Instead of branching, quantifier elimination by partial cylindrical algebraic decomposition [Collins and Hong 1991] employs pruning in a similar sense as described in this article. However, it still decomposes into truth-invariant cells, which again needs expensive computation of projection polynomials, and real algebraic numbers.

In contrast to this, the narrowing operator provided in this paper is cheap, and can do pruning in polynomial time. As a result, we have a clear separation between polynomial time pruning, and exponential branching. So we have a way of working around the high worst-case complexity of the problem, whenever a small amount of branching is necessary.

For inputs with free variables, all of these algorithms produce symbolic output that is equivalent to the input, but quantifier-free. This output can be huge. In many applications, such output is only considered a transformation of the problem, but not a solution. In contrast to this, our algorithm produces explicit numerical output, that one can directly visualize for dimensions less than three.

10. CONCLUSION

In this article, we have provided an algorithm for solving quantified inequality constraints over the reals. Although this is an undecidable problem, the algorithm terminates for all, except pathological (i.e., unstable) inputs.

The result has several advantages over earlier approaches: Compared to symbolic approaches [Collins 1975; Caviness and Johnson 1998] it is not restricted to polynomials, and avoids complicated and inefficient computation with real algebraic numbers. Furthermore, it decreases the necessity for expensive space decomposition by extracting information using fast consistency techniques. Compared to earlier interval approaches that could deal with quantifiers of some form, it provably terminates for all except unstable inputs, and can either handle a more general case [Jaulin and Walter 1996; Malan et al. 1997; Benhamou and Goualard 2000; Shary 1999], or provides a much cleaner, more elegant, and efficient framework [Ratschan 2002a].

As a side-effect, this article even improves the current methods for (unquantified) numerical constraint satisfaction problems in the case where the solution set does not consist of finitely many, isolated solutions.

In future work, we will explore optimal branching strategies [Ratschan 2002e; Kreinovich and Csendes 2001; Csendes and Ratz 1997], exploit continuity information for efficiently dealing with equalities [Ratschan 2002c], exploit the structure of quantified constraints in special problem domains, and provide an implementation that allows the flexible exchange of different atomic narrowing operators.
REFERENCES


Solving Quantified Inequality Constraints over Real Numbers


Received November 2002; revised January 2004 and January 2005; accepted January 2005
Chapter 11

Efficient Solution of a Class of Quantified Constraints with Quantifier Prefix Exists-Forall

This chapter appeared as:

Efficient Solution of a Class of Quantified Constraints with Quantifier Prefix Exists-Forall

Milan Hladík · Stefan Ratschan

Received: 14 December 2013 / Revised: 12 May 2014 / Accepted: 31 May 2014 / Published online: 29 July 2014
© Springer Basel 2014

Abstract In various applications the search for certificates for certain properties (e.g., stability of dynamical systems, program termination) can be formulated as a quantified constraint solving problem with quantifier prefix exists-forall. In this paper, we present an algorithm for solving a certain class of such problems based on interval techniques in combination with conservative linear programming approximation. In comparison with previous work, the method is more general—allowing general Boolean structure in the input constraint, and more efficient—using splitting heuristics that learn from the success of previous linear programming approximations.

Keywords Constraint solving · Decision procedures · Interval computation

Mathematics Subject Classification 65G20 · 65H20

1 Problem Description

We study the problem of finding $x_1, \ldots, x_r$ such that

$$\bigwedge_{i=1}^n \forall y_1, \ldots, y_s \in B_i \phi_i(x_1, \ldots, x_r, y_1, \ldots, y_s)$$

where each $B_i$ is a box (i.e., Cartesian product of closed intervals) in $\mathbb{R}^s$ and each of the $\phi_1, \ldots, \phi_n$ is a Boolean combination of inequalities where for each $i \in \{1, \ldots, n\}$ only one of those inequalities contains the variables $x_1, \ldots, x_r$ and this one inequality contains those variables only linearly. If no such $x_1, \ldots, x_r$ exist, we want to detect this. Here is an illustrating example:

$$\forall y_1 \in [0, 1], y_2 \in [-1, 1] \ [y_1 \geq y_2 \lor x_1 \sin(y_1)y_2 + x_2y_1^2y_2 \leq 0]$$

$$\forall y_1 \in [0, 1], y_2 \in [-1, 1] \ [y_1 < y_2 \lor x_1 \cos(y_1)y_2 + x_2y_1y_2^2 \leq 0].$$
We also study the extension of this problem to the case where the conjunction may—in addition to constraints of the form
\[ \forall y_1, \ldots, y_t, \exists \phi_i(x_1, \ldots, x_r, y_1, \ldots, y_t) \]
also contain linear equalities in the variables \( x_1, \ldots, x_r \) (the equalities can be viewed as a conjunction of inequalities, but this violates the condition that only one inequality contains \( x_1, \ldots, x_r \)).

In an earlier paper [22], we showed how to solve a special case, with restricted Boolean structure. The contributions of this paper are:

- The extension of the approach to arbitrary Boolean structure.
- The design of splitting heuristics that improves the performance of the algorithm by orders of magnitude.

Constraints of this type occur in various applications. Especially, they are useful in finding certificates for certain global system properties. For example, a Lyapunov function [15] represents a certificate for the stability of dynamical systems. In the case of global stability, such a function has to fulfill certain properties in the whole state space except for the original/equilibrium. After using an ansatz (often also called template) of the Lyapunov function as a polynomial with parametric coefficients, one can find the Lyapunov function by solving a universally quantified problem for those parameters. The fact that polynomials are linear in their coefficients corresponds to linearity of our variables \( x_1, \ldots, x_r \). A similar situation occurs, for example in termination analysis of computer programs [4,18], or in the termination analysis of term-rewrite systems [7].

However, usually further work is usually necessary to apply the method studied in this paper to such problems: In the case of Lyapunov functions, one has to exempt one single point (the equilibrium) from the property which cannot be directly expressed by the boxes \( B_1, \ldots, B_n \). In the case of termination analysis, such constraints have to be solved for the whole real space instead of boxes \( B_1, \ldots, B_n \). In an earlier paper, we solved this problem for Lyapunov functions [10,22]. However, in other areas this is an open area for further research.

In the polynomial case, constraints over the domain of real numbers with quantifiers can always be solved due to Tarski’s classical result that the first-order theory of the real numbers allows quantifier elimination [28]. The area of computer algebra has developed impressive software packages for quantifier elimination [2,8]. However, those still have problems with scalability in the number of involved variables and, in general, they cannot solve non-polynomial problems. This was the motivation for several approaches to use interval based techniques for such problems, for special cases [1,11,14] and for arbitrary quantifier structure [21].

Alternative approaches for finding Lyapunov function certificates are based on techniques from real algebraic geometry [19,25]. However, those methods cannot solve general constraints of the form discussed in this paper (no general Boolean structure, no non-polynomial constraints).

In termination analysis of term-rewrite systems constraints with quantifier-prefix \( \exists \forall \) are usually solved by first eliminating the universally quantified variables using conservative approximation [13,16], and then solving the remaining, existentially quantified problem. Again, this technique cannot solve general constraints of the form discussed in this paper (no general Boolean structure, no non-polynomial constraints).

The structure of the paper is as follows: In the next section, we will introduce the basic algorithm for solving the constraints. In Sect. 3 we will introduce splitting heuristics for the algorithm. In Sect. 4 we will extend the algorithm with equality constraints. In Sect. 5 we will prove termination of the algorithm for all non-degenerate cases. In Sect. 6 we discuss how, for a given box to split, choose the variable of that box to split. In Sect. 7 we will provide the results of computational experiments with the algorithm. And in Sect. 8 we will conclude the paper.

Throughout the paper, boldface variables denote objects that are intervals or contain intervals (e.g., interval vectors or matrices).

2 Basic Algorithm

We use the following algorithm (which generalizes an algorithm [22] that solves constraints of a more specific form arising in the analysis of ordinary differential equations):
1. For each \( i \in \{1, \ldots, n\} \), substitute the intervals of \( B_i \) corresponding to \( y_1, \ldots, y_i \) into \( \phi_i \), and evaluate using interval arithmetic. As a result, all the inequalities that do not contain \( x_1, \ldots, x_r \) are simplified to an inequality of the form \( I \leq 0 \), where \( I \) is an interval, and every inequality (for each \( i \in \{1, \ldots, n\} \)) that does contain \( x_1, \ldots, x_r \) to an inequality of the form

\[
p_1 x_1 + \ldots + p_r x_r \leq q,
\]

where the \( p_1, \ldots, p_r, q \) are intervals.

2. Replace any inequality of the form \( I \leq 0 \) with a (not necessarily strictly) negative upper bound of \( I \) by the Boolean constant \( T \) (for “true”).

3. Replace any inequality of the form \( I \leq 0 \) with a (strictly) positive lower bound of \( I \) by the Boolean constant \( F \) (for “false”).

4. Simplify the constraint further using basic reasoning with the Boolean constants \( T \) and \( F \) (e.g., simplify \( T \lor \phi \) to \( T \), \( F \lor \phi \) to \( \phi \)).

5. If the resulting constraint is a Boolean constant, we are done (if the result is the Boolean constant \( T \), every \( x_1, \ldots, x_r \) is a solution, if the result is \( F \), no solution exists).

6. If the constraint is an interval linear system of inequalities \( P x \leq q \) (i.e., all disjunctions \( \lor \) in the formula have been removed by the simplifications in Step 4), we reduce the interval linear system to a linear system \( Az \leq b \) using the method of Rohn and Kreslová \([12,24]\), and solve this system using linear programming. If the result is not yet an interval linear system, we continue with the next step.

7. If the previous step resulted in a solvable linear program, we have a solution to the original problem. If no, we choose an \( i \in \{1, \ldots, n\} \), split the box \( B_i \) into pieces \( B^1_i, B^2_i \), replace the original constraint by

\[
\land_{j \in \{i-1,i+1,\ldots,n\}} \left[ \forall y_1, \ldots, y_s \in B_j \phi_j(x_1, \ldots, x_r, y_1, \ldots, y_s) \right] \land
\left[ \forall y_1, y_2 \in B^1_i \phi_1(x_1, \ldots, x_r, y_1, \ldots, y_s) \right] \land
\left[ \forall y_1, y_2 \in B^2_i \phi_2(x_1, \ldots, x_r, y_1, \ldots, y_s) \right]
\]

and iterate from Step 1 of the algorithm.

Note that this algorithm only splits boxes with bounds pertaining to the variables \( y_1, \ldots, y_s \), but not wrt. variables \( x_1, \ldots, x_r \). This is the main advantage of such an algorithm over a naive algorithm that substitutes sample points for the free variables \( x_1, \ldots, x_n \). Completeness can be preserved due to completeness of the Rohn/Kreslová algorithm which we now describe in more detail:

The basic idea is, to replace each variable \( x_i \) by two non-negative variables \( x^1_i \) and \( x^2_i \), and then to rewrite each term \( [p_l, p_u]x_i \) of the interval system of inequalities to \( [p_l, p_u](x^1_i - x^2_i) \) which is equal to \( [p_l, p_u]x^1_i - [p_l, p_u]x^2_i \).

Based on the fact that the inequalities should hold for all elements of the intervals, we can now exploit the fact that the \( x^1_i \) and \( x^2_i \) are non-negative. Hence, we can now replace the interval coefficients of \( x^1_i \) with their upper endpoint, and the interval coefficients of \( x^2_i \) with their lower endpoint, resulting in \( p_u x^1_i - p_l x^2_i \). The result is an interval system of inequalities of the form \( p_u x^1_i - p_l x^2_i \leq q \). \( x_1 \geq 0, x_2 \geq 0 \).

3 An Informed Splitting Strategy

The major building block of the algorithm that we left open is the splitting strategy: Which box to choose for splitting in Step 7 and which variable to split it. In this section we will develop such a strategy. We will first describe the basic idea for the linear system \( Az \leq b \) created in Step 6 of the algorithm (Sect. 3.1), then we will study how to take into account the fact that the linear system was created from an interval linear system \( P x \leq q \) by the algorithm of Rohn and Kreslová (Sect. 3.2), then we will study how to ensure convergence of the strategy (Sect 3.3), take into account interval evaluation from Step 1 of the main algorithm (Sect. 3.4), and summarize the result into a sub-algorithm of our main algorithm (Sect. 3.5).
3.1 Basic Idea

Our goal is to have a strategy that is

- complete: if the problem has a solution, we will eventually find it\(^1\)
- efficient: the algorithm converges to a solution as fast as possible.

It is not too difficult to ensure completeness of the algorithm: Just ensure that the width of all boxes goes to zero [21, 22]. However, the result can be highly inefficient: each split increases the size of the constraint to solve, slowing down the algorithm. Hence it is essential to concentrate on splits that bring the constraint closer to solvability.

Since splitting heuristics for classical interval branch-and-bound (or branch-and-prune) algorithms are well-studied [6, 9, 20] we assume that in Step 6 the algorithm already arrived at an interval linear system of inequalities.

We will try to come up with splits that bring the next linear program closer to solvability. For achieving this, we need some measure of what it means for an infeasible system of inequalities \(Az \leq b\) (as created by Step 6 of the algorithm) to be close to solvability, which will lead us to a method for determining how it can be brought closer to solvability by splitting.

The overall approach is to

1. use the minimum of the residual \(\max_{i \in \{1, \ldots, n\}} (Az - b)_i\), that is
   \[
   \min_z \max_{i \in \{1, \ldots, n\}} (Az - b)_i
   \]
   as a measure of closeness to feasability (here the index \(i\) denotes the \(i\)-th entry of the vector \(Az - b\)),
2. to compute the corresponding minimizer, and then to
3. use those splits that promise to improve the residual for this minimizer the most.

For computing the minimum of the residual, we reformulate

\[
\min_z \max_{i \in \{1, \ldots, n\}} (Az - b)_i
\]

as the constrained optimization problem

\[
\min_{z, \rho} \rho \text{ subject to } \rho = \max_{i \in \{1, \ldots, n\}} (Az - b)_i
\]

which is

\[
\min_{z, \rho} \text{ subject to } \rho \geq (Az - b)_1, \ldots, \rho \geq (Az - b)_n,
\]

from where we arrive at the linear program

\[
\min_{z, \rho} \text{ subject to } Az - b \leq [1, \ldots, 1]^T \rho.
\]

Let \(z^*, \rho^*\) be the resulting minimizer. If the residual \(\rho^* \leq 0\) then we know that the system \(Az \leq b\) is solvable. If not, then the constraint violation vector \(Az^* - b\) provides information on how much the individual constraints contribute to non-solvability.

We try to decrease the constraint violation of the row of \(A\) for which \(Az^* - b\) is maximal. Denote this row by \(i\). The constraint corresponding to this row is of the form \(a_1 z^*_1 + \ldots + a_{2r} z^*_{2r} \leq b\) where each coefficient \(a_j, j \in \{1, \ldots, 2r\}\) results from an endpoint of some interval in the interval system \(Px \leq q\). Now we want to choose a \(j \in \{1, \ldots, 2r\}\) such that splitting will aim at changing the coefficient \(a_j\) as much as possible. We assume that the change that we can expect for coefficient \(a_j\) if using such a split, is given by some real number \(\delta_j\). Under this assumption, the inequality will change to

\[
a_1 z^*_1 + \ldots + a_{j-1} z^*_{j-1} + (a_j + \delta_j) z^*_j + a_{j+1} z^*_{j+1} + \ldots + a_{2r} z^*_{2r} \leq b
\]

\(^1\) With the exception of degenerate cases, see Sect. 5.
which is
\[ a_1 z^*_1 + \ldots + a_{2r} z^*_r \leq b - \delta_j z^*_j, \]
resulting in an improvement \( -\delta_j z^*_j \).

Hence we can expect the maximal improvement of the residual by choosing \( j \) as

\[ \arg \max_{j \in \{1, \ldots, 2r\}} -\delta_j z^*_j \]

For analyzing how \( \delta_j \) should look like, we have to analyze the precise form of the system \( Az - b \) which we will do in the next sub-section.

### 3.2 Exploiting Structure

Now observe that the linear program that we used in the previous sub-section is not arbitrary, but is the result of the Rohn/Kreslová transformation of an interval system of linear inequalities of the form

\[ \min \rho \text{ subject to } \overrightarrow{P} x^1 - P x^2 - b \leq [1, \ldots, 1]^T \rho, x^1 \geq 0, x^2 \geq 0. \]

Observe that the entries of the underlying interval linear system of inequalities \( P x \leq q \) are created by interval evaluation (Step 1 of the main algorithm). Assuming that splitting shrinks large entries of the interval matrix \( P \) more than small intervals, the change \( \delta \) that we can expect for \( a_j \) from splitting is proportional to the width \( w(p_{v(j)}) \) of the corresponding interval \( p_{v(j)} \) in the \( i \)-th row of \((p_1, \ldots, p_n)\) of \( P \). However, since splitting results in a sub-interval \( P'_{v(j)} \subseteq p_{v(j)} \), the expected change for lower bounds of intervals is positive, and for upper bounds of intervals is negative.

Analyzing the left-hand side \( \overrightarrow{P} x^1 - P x^2 - b \) of the linear program resulting from the Rohn/Kreslová transformation, we observe that the \( x^1 \) have coefficient \( \overrightarrow{P} \), that is, the sign of upper bounds is positive, and the \( x^2 \) have coefficient \( -P x^2 \) that is, the sign of lower bounds is negative. Combining this with the fact that lower bounds will be increased and upper bounds be decreased by splitting, the expected change \( \delta_j = -w(p_{v(j)}) \), resulting in

\[ \arg \max_{j \in \{1, \ldots, 2r\}} -\delta_j z_j = \arg \max_{j \in \{1, \ldots, 2r\}} w(p_{v(j)}) z^*_j. \]

Now observe furthermore, that the coefficients of the linear program come in pairs that refer to the two bounds the same intervals, and hence also their width is the same. So, instead of

\[ \arg \max_{j \in \{1, \ldots, 2r\}} w(p_{v(j)}) z^*_j \]

we can directly refer to the interval matrix:

\[ \arg \max_{j \in \{1, \ldots, r\}} [w(p_j) \max(x^1_j, x^2_j)] \]

where \( x^1_j \) and \( x^2_j \) refer to the individual entries of the vectors of variables as introduced by the Rohn/Kreslová transformation.

We also note the following:

**Lemma 1** Let \( \overrightarrow{P} \) and \( P \) be real matrices in \( \mathbb{R}^{n \times r} \) such that for every \( i \in \{1, \ldots, n\}, \ j \in \{1, \ldots, r\}, \ P_{i,j} < \overrightarrow{P}_{i,j}. \) Let \( b \) a real vector in \( \mathbb{R}^n \). Then for every solution \( x^1, \ x^2 \) of the linear program

\[ \min \rho \text{ subject to } \overrightarrow{P} x^1 - P x^2 - b \leq [1, \ldots, 1]^T \rho, x^1 \geq 0, x^2 \geq 0 \]

for every \( j \in \{1, \ldots, r\}, \) either \( x^1_j \) or \( x^2_j \) is zero.
Proof Let $P^c$ be the center of the interval matrix $P$, that is the matrix that contains the midpoint of the corresponding intervals of $P$. Let $P^\Delta$ be the matrix that contains for every entry the width of the corresponding interval of $P$. Then the above linear program is equivalent to
\[
\min \rho, \ P^c (x^1 - x^2) + P^\Delta (x^1 + x^2) - b \leq [1, \ldots, 1]^T \rho, \ x^1 \geq 0, \ x^2 \geq 0
\]

Let $j \in \{1, \ldots, r\}$ be arbitrary, but fixed, and assume that both $x^1_j$ and $x^2_j$ are non-zero. Then, we can replace $x^1_j$ by $x^1_j - \varepsilon$ and $x^2_j$ by $x^2_j + \varepsilon$, where $\varepsilon > 0$. As a result, the value of the first term $P^c (x^1 - x^2)$ stays unchanged, while the value of the second term $P^\Delta (x^1 + x^2)$ has decreased. Hence we can decrease the minimum of the linear program, which is a contradiction to the assumption that the original values $x^1_j$ or $x^2_j$ were a solution of the linear program. \qed

3.3 Ensuring Convergence

The basic idea, as described in the previous section, does not result in a converging method. We will demonstrate this on a concrete example, taking into account the precise form of how the system of inequalities is created by the method of Rohn and Kreslová. For this, assume the interval inequality
\[
[-1, 3] x_1 + [-3, 1] x_2 \leq -2
\]
and the corresponding inequality
\[
3 x^1_1 + x^2_1 + x^1_2 + 3 x^2_2 \leq -2, \ x^1_1 \geq 0, \ x^2_1 \geq 0, \ x^1_2 \geq 0, \ x^2_2 \geq 0.
\]
The resulting linear program
\[
\min_{x^1_1, x^1_2, x^2_1, x^2_2} \rho \quad \text{subject to} \quad 3 x^1_1 + x^2_1 + x^1_2 + 3 x^2_2 + 2 \leq [1, \ldots, 1]^T \rho, \ x^1_1 \geq 0, \ x^2_1 \geq 0, \ x^1_2 \geq 0, \ x^2_2 \geq 0
\]
has the solution $\rho = 2, \ x^1_1 = 0, \ x^2_1 = 0, \ x^1_2 = 0, \ x^2_2 = 0$ which corresponds to the values $x_1 = 0, x_2 = 0$ of the original interval inequality. Evaluating our heuristics, we get
\[
\arg \max_{j \in \{1, 2\}} \bigl[ w(p_j) \max \{x^1_j, x^2_j\} \bigr] = \arg \max_{j \in \{1, 2\}} 0 = 0
\]
Hence our heuristics already compute the value 0 for each coefficient, suggesting that no shrinking of interval coefficients is necessary anymore (Theorem 1 in Sect. 5 will provide a more general characterization of such behavior). Still, we have not yet found a solution of $[-1, 3] x + [-3, 1] y \leq -2$, and the residual value $\rho > 0$ correctly indicates this. Moreover, a shrinking of the first interval, for example, resulting in
\[
[2, 3] x + [-3, 1] y \leq -2
\]
leads to a solvable system.

Analyzing the problem, we see that for the solution of $[2, 3] x + [-3, 1] y \leq -2$, $x \neq 0$! So the original heuristics was misleading, since it mistakenly assumed $x = 0$. In other words, while the minimizer $z^*$ of the linear program $\min_{z^*} \rho, \ Az - b \leq [1, \ldots, 1]^T \rho$ gives some orientation on which coefficients to shrink, it need not necessarily be a solution of the original input constraint, and hence may be misleading.

To fix the problem, we assume that the minimizer $x^1, x^2$ to the linear program only approximates the final solution of the input constraint that we are looking for. For each $j \in \{1, \ldots, r\}$, the final solution might instead be located in an interval $[x^1_j - x^2_j - \varepsilon, x^1_j - x^2_j + \varepsilon]$ around the corresponding solution $x^1_j - x^2_j$ of the interval system of linear inequalities.

We will now analyze the corresponding changed value of the term $\max \{x^1_j, x^2_j\}$ used in the computation of the heuristic value
\[
\arg \max_{j \in \{1, \ldots, r\}} \bigl[ w(p_j) \max \{x^1_j, x^2_j\} \bigr].
\]
In the case where \( x_j^1 - x_j^2 \geq 0 \), by Lemma 1, \( x_j^2 = 0 \). Hence the original value of the term \( \max\{x_j^1, x_j^2\} \) is \( x_j^1 \), and the corresponding changed value is \( x_j^1 + \varepsilon \). In the case where \( x_j^1 - x_j^2 < 0 \), the original value is \( x_j^2 \), and the changed value is \( x_j^2 + \varepsilon \). Putting those cases together, the changed value is \( \max\{x_j^1, x_j^2\} + \varepsilon \).

Hence the corresponding heuristic value can be up to
\[
\arg \max_{j \in \{1, \ldots, r\}} \left[ w(p_j) \left( \max\{x_j^1, x_j^2\} + \varepsilon \right) \right]
\]
which we will use, for a user-provided constant \( \varepsilon > 0 \).

We will see later (Sect. 5), that even if the constant \( \varepsilon \) does not correctly estimate the difference between the solution of the current linear program and a solution of the original constraint \( \phi \), for constraints that have a non-degenerate solution, the resulting method always converges to such a solution, if \( \varepsilon > 0 \).

In general, we will use heuristics of the form
\[
\arg \max_{j \in \{1, \ldots, r\}} h(p_j, x_j^1, x_j^2)
\]
and show convergence under certain conditions of this function \( h \).

3.4 Interval Evaluation

Up to now, we know which row(s) of \( P \) to split, that is, for which \( i \in \{1, \ldots, n\} \) to split the box \( B_i \). We also know, which bound of which interval in that row of \( P \) we want to decrease, but we still do not know which coordinate of \( B_i \) result in the biggest decrease of that bound. For determining this, observe that each entry of the interval matrix \( P \) results from interval evaluation of a certain expression on the box \( B_i \). Hence we need to infer, for a given arithmetical expression and an interval for each variable in that expression, which split of an interval results in the biggest decrease of the given resulting (lower or upper) bound of interval evaluation. There are many possible choices for this. Hence our approach will be parametric in the concrete method used. We will assume a function \( \text{splitheur} \) such that for a given arithmetical expression \( t \), box \( B \), and sign \( s \in \{-, +\} \), \( \text{splitheur}(t, B, s) \)

- returns a variable of \( B \) to split for improving the lower/upper bound (depending on \( s \in \{-, +\} \)) of the interval evaluation of \( t \) on \( B \), and for which
- repeated splitting according to this function converges, that is, for the sequence \( B_1, B_2, \ldots \) created by splitting according to this function, for every \( \varepsilon > 0 \) there is a \( k \) such that for all \( i \geq k \) the width of \( t(B_i) \) is smaller than \( \varepsilon \).

Right now, we use this function for only the coefficient chosen by our heuristics. It might also make sense to try it on all coefficients, and choose the best one.

3.5 Algorithm

The resulting algorithm is called from the main algorithm in Step 7 in the case where in Step 6 we arrived at an (unsolvable) interval linear system. The algorithm has the following form (where, for an arithmetical expression \( t \) and a box \( B \), \( \text{eval}(t, B) \) denotes interval evaluation of \( t \) on \( B \)):

**Input:** expressions \( t_{i,j}, i \in \{1, \ldots, n\}, j \in \{1, \ldots, r\} \) s.t. \( t_{i,j} \) is the coefficient of \( x_j \) in \( \phi_i \), boxes \( B_i, i \in \{1, \ldots, n\} \)

**Output:** \( i \in \{1, \ldots, n\}, k \in \{1, \ldots, s\} \) suggesting to split box \( B_i \) at its \( k \)-th coordinate

1: let \( P \) be the \((n \times r)\)-interval matrix s.t. \( P_{i,j} = \text{eval}(t_{i,j}, B_i), i \in \{1, \ldots, n\}, j \in \{1, \ldots, r\} \)
2: \((x^1, x^2) \leftarrow \arg \min \rho, \quad \overline{P} x^1 - \underline{P} x^2 - b \leq 1, \ldots, 1 \end{bmatrix}^T \rho, \quad x^1 \geq 0, x^2 \geq 0 \)
3: \( d \leftarrow \overline{P} x^1 - \underline{P} x^2 - b \) \quad // residual
4: \( i \leftarrow \arg \max_{i \in \{1, \ldots, n\}} \text{residual} \)
5: \( j \leftarrow \arg \max_{j \in \{1, \ldots, r\}} h(P_{i,j}, x^1_j, x^2_j) \) \quad // box \( B_i \) to split
6: \( \text{return } i, \text{splitheur}(t_{i,j}, B_i, \text{sgn}(x^1_j - x^2_j)) \) \quad // coefficient to improve
We will call this version of the algorithm the split-worst version. We will also consider an alternative version that, instead of splitting only the box corresponding to the maximal constraint violation (as computed in Line 4), splits all boxes with positive constraint violation. We will call that version of the algorithm, the split-all version.

As already discussed above, if for the solution \( \rho \) computed in Line 2, \( \rho \leq 0 \), then we know that the interval linear system \( P x \leq q \) from Line 6 of the main algorithm is solvable. Moreover, since \( \rho \leq 0 \) is equivalent to the linear system of Rohn/Kreslová being solvable, this computes the same information as Line 6 of the main algorithm and hence no solving has to be done there. In other words, instead of solving the Rohn/Kreslová linear system of equations, we solve the linear program

\[
\min \rho, \quad P x^0 - b \leq [1, \ldots, 1]^T \rho, x^1 \geq 0, x^2 \geq 0
\]

and use it both for determining the overall solution of the algorithm and heuristics for splitting.

4 Equality Constraints

Now we analyze the extended problem that—in addition to constraints of the form \( \forall y s.t. \phi_i(x_1, \ldots, x_T, y_1, \ldots, y_s) \)—also contain linear equalities over the variables \( x_1, \ldots, x_T \). Viewing each equality as a conjunction of two inequalities one sees that in that case, the two inequalities force the optimum \( \rho^* \) of

\[
\min \rho, \quad P x^1 - b \leq [1, \ldots, 1]^T \rho, x^1 \geq 0, x^2 \geq 0
\]

to be zero. So in this case, the heuristics in the form described above are not useful. In order to handle equalities better, we do not view such equalities as two inequalities, but we handle them directly. That is we solve the linear program

\[
\min \rho, \quad P x^1 - P x^2 - b \leq [1, \ldots, 1]^T \rho, x^1 \geq 0, x^2 \geq 0
\]

where \( Cx = d \) is the linear system of equations containing all the linear equalities.

5 Convergence

Our main algorithm consists of a loop that continues until a solution has been found. In this section we will answer the question: Will the loop terminate for all input constraints? Again we will assume that in Step 6 the algorithm already arrived at an interval linear system of inequalities, since convergence of basic interval branch-and-bound (or branch-and-prune) algorithms is not difficult to show (e.g., it follows as a special case of Theorem 6 in [21]).

Observe that the only place where the algorithm approximates, is the interval evaluation in Step 1 of the main algorithm. In the whole section, for the formal proofs, we assume that the resulting linear programs are solved precisely, using rational number arithmetic. Still, in practice, it suffices to solve them approximately, for example, based on floating-point arithmetic.

In the following we will denote by \( \hat{P} x \leq \hat{q} \) the system of interval linear inequalities that would result from the input constraint if interval evaluation would be non-overapproximating. In a similar way, we will denote by \( \hat{A} z \leq \hat{b} \) the system of linear inequalities corresponding to \( \hat{P} x \leq \hat{q} \).

**Definition 1** We call \( z \) a robust solution of a system of inequalities and equalities \( Az \leq b \land Cz = d \) iff \( Az < b \land Cz = d \). We call a constraint \( \phi \) robust if the corresponding system \( \hat{A} z \leq \hat{b} \land Cz = d \) has a robust solution.

In other words, a robust solution of a system \( Az \leq b \land Cz = d \) is an interior point of \( Az \leq b \) that satisfies the equalities \( Cz = d \).

**Lemma 2** If \( z \) is a robust solution of \( Az \leq b \land Cz = d \), then there is an \( \varepsilon > 0 \) s.t. that for all \( A' \), and \( b' \) differing from \( A \) and \( b \) not more than \( \varepsilon \) for each entry, \( A' z \leq b' \land Cz = d \).
Proof Since $z$ is a robust solution, $Az - b$ is a vector of negative numbers. From this we can compute an upper bound on the allowed changes of $A$ and $b$.

It is not difficult to ensure convergence of the algorithm:

**Lemma 3** Assume that the splitting strategy ensures that the width of every bounding box $B_i$ goes to zero. Then the algorithm will terminate for robust inputs.

Proof Assume an arbitrary iteration of the algorithm. As above, denote by $\hat{P}_x \leq \hat{q}$ the interval system of inequalities that the algorithm would compute if using the precise range instead of over-approximating interval evaluation in Step 1. Denote by $Az \leq b$ the corresponding system of linear inequalities the algorithm computes from $P_x \leq q$ in Step 6. Assuming, in addition, a system of linear equalities $Cz = d$, let $\hat{z}$ be the robust solution of $\hat{A}z \leq \hat{b} \land \hat{C}z = \hat{d}$, so $\hat{A}z < \hat{b}$. Due to the fact, that the algorithm does not compute the precise range in Step 1, but over-approximates it using interval evaluation, the algorithm will compute with an interval matrix $P \supseteq \hat{P}$. The over-approximation error goes to zero due to convergence of interval arithmetic. Hence $A$ will be approximated increasingly well. So, due to Lemma 2, $\hat{A}z \leq \hat{b} \land \hat{C}z = \hat{d}$ will eventually hold and the algorithm terminates.

However, our heuristics do not necessarily ensure that the width of every bounding box goes to zero: Even if it would ensure that every bounding box is split infinitely often, it might still happen that the width of some bounding box does not go to zero, because a certain coordinate of the box is not split infinitely often. This might not even be necessary for termination, because this coordinate might correspond to a variable that does not occur in an coefficient term.

**Theorem 1** Consider the split-all version of the algorithm with heuristics of the form

$$\max_{j \in \{1, \ldots, n\}} h(p_j, x_j^1, x_j^2)$$

where

(a) $\lim_{w(p) \to 0} h(p, x) = 0$
(b) $w(p) > 0$ implies $h(p, x) > 0$.

Then we have: If the input constraint has a robust solution, then the algorithm terminates.

Proof We consider the split-all version of the algorithm and assume that the algorithm does not terminate. Then it creates an infinite sequence of unsolvable interval linear programs and corresponding linear programs (see Line 6 of the main algorithm). In each iteration all $B_i$ with positive constraint violation are split. Hence, all those bounding boxes are split infinitely often. In each iteration, in Line 5 of the algorithm from Sect. 3.5, a coefficient $j$ for improvement is chosen. If all coefficients are chosen infinitely often, then due to convergence of splitheur the width of all coefficients goes to zero, which implies that the constraint will eventually have non-positive constraint violation and the corresponding bounding box would not be chosen for splitting, which is a contradiction.

Hence, non-termination implies that at least one of the coefficients is not chosen infinitely often. Let us analyze the state of the algorithm where all coefficients that are chosen finitely often will not be chosen any more. All other coefficients are chosen infinitely often, which means that due to convergence of spliteur, their interval width goes to zero. Hence, due to Assumption (a), their $h$-value goes to zero. Moreover, due to Assumption (b) the coefficients that are not split any more, have positive $h$-value. This implies that Line 5 eventually chooses one of them, a contradiction. So the algorithm terminates.

Clearly, the heuristics

$$\arg \max_{j \in \{1, \ldots, r\}} [w(p_j)[\max(x_j^1, x_j^2) + \varepsilon]],$$

with $\varepsilon > 0$, as developed in Sect. 3, fulfill the assumptions of the theorem, and hence the algorithm converges.
6 Variable Splitting Heuristics

In this section we discuss, how the function splitheur\(t, B, s\), that we introduced in Sect. 3.4, can be implemented.

A widely used technique (e.g., in global optimization [6]) for this is to use derivatives of the arithmetical expression. An alternative would be Corollary 2.1.2 in Neumaier’s book [17]. However, that would need the computation of interval over-approximation of derivatives, or interval Lipschitz constants, respectively. Moreover, those techniques are only a priori estimates of the decrease that might fail to give exact information. In order to arrive at more precise information, we use the observation that we already have a fixed set of usually small expressions that we want to analyze. Hence, interval evaluation of those expressions will usually take negligible time compared to the rest of the algorithm. Hence we explicitly try all possible splits and compare their effect on the width of the result of interval evaluation [5]:

\[
\text{splitheur}(t, B, s) = \arg \max_{i \in \{1, \ldots, |B|\}} \min_k \{ |b^s(r(B)) - b^s(t(\text{split}^k(B, i)))| \}
\]

where \(b^s\) takes the upper/lower bound respectively of the argument interval according to \(s\), and \(\text{split}^k(B, i)\) denotes the \(k\)-th box resulting from splitting the box \(B\) at variable \(i\) (usually \(k \in \{1, 2\}\)).

However, the method as described up to now does not ensure convergence of the method. The reason is the following: Assume a term \(t\) in \(n\) variables. Assume intervals \(I_1, \ldots, I_n\) on which we evaluate \(t\). Let \(I^n_1\) be the lower and \(I^n_2\) the upper half of \(I_1\). Assume a procedure that replaces that interval \(I_i\) by its lower or upper half, for which this results in the biggest decrease of interval evaluation of \(t\). Repeated application of this procedure does not result in the width of interval evaluation going to zero. For example, for the term \(x^2 + y\) with \(x \in [-1, 1], y \in [-0, 2]\), splitting \([-1, 1]\) does not result in any improvement at all. However, it is necessary for global convergence.

One way of solving this problem is, to take the time since the last split into account. For example, we could use

\[
\text{splitheur}(t, B, s) = \arg \max_{i \in \{1, \ldots, |B|\}} \left[ c(i) + \min_k \{ |b^s(r(B)) - b^s(t(\text{split}^k(B, i)))| \} \right]
\]

where \(c(i)\) is a function that increases with the time of the last split of variable \(i\). If this function goes to infinity, the time with the last split, then every variable will be split eventually, ensuring convergence. The result is some compromise between round-robin-splitting (which ensures convergence) and aggressive local improvement. In order to make this heuristics independent of the size of \(B\) (which decreases during the algorithm) it makes sense to use some scaling with \(w(t(B))\) in the function \(c(t)\).

7 Computational Experiments

We did experiments on examples for computing Lyapunov-like functions [22], with the heuristic function

\[
\arg \max_{j \in \{1, \ldots, r\}} \left[ w(p_j)[\max\{x_1^j, x_2^j\} + \varepsilon] \right]
\]

and \(\varepsilon = 0.001\).

For the resulting examples we have \(\phi_1 = \ldots = \phi_r\) with different bounding boxes for each branch \(i \in \{1, \ldots, r\}\). The bounding boxes and the inequality constraints of the examples are as follows:

Example A:

- \(B_1 = [0.8, 1.2] \times [0.3, 0.49]\)
- \(B_2 = [0.8, 1.2] \times [0.51, 0.7]\)
- \(B_3 = [1.01, 1.2] \times [0.49, 0.51]\)
- \(B_4 = [0.8, 0.99] \times [0.49, 0.51]\)

where \(\phi\) is of the form

\[
\begin{align*}
&x_1(2y_1^3y_2 - 2y_1^2y_2 + y_1) + x_2(y_1^2y_2 - y_1 + 0.5) + x_3(y_1^2y_2 - y_1^3y_2 - y_1y_2 + 0.5y_1 + 0.5y_2) \\
&+ x_4(0.5 - y_1y_2) + x_5((-2)y_1^3y_2^2 + y_2) \leq -0.0001
\end{align*}
\]

Example B:

- \(B_1 = [-0.8, 0.8] \times [-0.8, -0.1]\)
- \(B_2 = [-0.8, 0.8] \times [0.1, 0.8]\)
- \(B_3 = [-0.8, -0.1] \times [-0.1, 0.1]\)
- \(B_4 = [0.1, 0.8], [-0.1, 0.1]\)
Table 1 Results of experiments

<table>
<thead>
<tr>
<th></th>
<th>Round-robin</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Splits</td>
<td>Time</td>
<td>Splits</td>
<td>Time</td>
</tr>
<tr>
<td>A</td>
<td>48</td>
<td>0.260</td>
<td>21</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>B</td>
<td>545</td>
<td>8.777</td>
<td>112</td>
<td>0.648</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
<td>0.128</td>
<td>5</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>D</td>
<td>719</td>
<td>70.176</td>
<td>10</td>
<td>2.472</td>
</tr>
<tr>
<td>A'</td>
<td>0</td>
<td>$\varepsilon$</td>
<td>0</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>B'</td>
<td>719</td>
<td>19.233</td>
<td>82</td>
<td>0.300</td>
</tr>
<tr>
<td>C'</td>
<td>244</td>
<td>6.472</td>
<td>4</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>D'</td>
<td>0</td>
<td>$\varepsilon$</td>
<td>0</td>
<td>$\varepsilon$</td>
</tr>
</tbody>
</table>

where $\phi$ is of the form

$$x_1(-2y_1^2 + 2y_1y_2) + x_2(0.2y_1y_2 - 4y_2^2 - 2y_1^2y_2 - 0.2y_1^2y_2) \leq -0.00001$$

Example C:

$B_1 = [-0.4, 0.4] \times [-0.4, 0.1]$, $B_2 = [-0.4, 0.4] \times [0.1, 0.4]$, $B_3 = [-0.4, 0.1] \times [-0.1, 0.1]$, $B_4 = [0.1, 0.4] \times [-0.1, 0.1]$

where $\phi$ is of the form

$$x_1(-16y_1^6 + 24y_1^5 - 8y_1^4) + x_2(-12y_1^5 + 18y_1^4 - 6y_1^3)$$

$$+ x_3(-8y_1^4 + 12y_1^3 - 4y_1^2) + x_4(-4y_2^2) \leq -0.000001$$

Example D:

$B_1 = [-0.2, 0.2] \times [-0.2, 0.2] \times [-0.2, 0.1]$, $B_2 = [-0.2, 0.2] \times [-0.2, 0.2] \times [0.1, 0.2]$, $B_3 = [-0.2, 0.2] \times [-0.2, 0.1] \times [-0.1, 0.1]$, $B_4 = [-0.2, 0.2] \times [0.1, 0.2] \times [-0.1, 0.1]$, $B_5 = [-0.2, 0.1] \times [-0.2, 0.1] \times [-0.1, 0.1]$, $B_6 = [0.1, 0.2] \times [-0.1, 0.1] \times [-0.1, 0.1]$

where $\phi$ is of the form

$$x_1(-2y_1y_2) + x_2(-2y_2y_3) + x_3(-2y_3^2 - 2y_1y_3 + 2y_1^2y_3) + x_4(-y_1^2 - y_1y_3)$$

$$+ x_5(y_1^2 - 2y_1y_2 - 2y_2y_3 - 2y_3y_1^3 + y_1^4) + x_6(-2y^2 - y_2^2 - y_1y_2 - 2y_2y_3 + y_1^3y_2) \leq -0.0001$$

In all four cases, we normalized the first coefficient $a$ to 1. To create versions with equality constraints we used the pre-processing method described in Section 4.3. of [22]. We will denote the result by $A'$, $B'$, $C'$ and $D'$.

The results of the experiments can be seen in Table 1. Here, round-robin refer to the classical round-robin splitting heuristics where variables are split one after the other, and that we used in earlier work [22]. Empty entries correspond to cases where the algorithm did not terminate within 10 min, and $\varepsilon$ corresponds to cases where the algorithm terminates in less than 0.1 s. The experiments were done based on an implementation in the programming language Objective Caml using the LP solver Glpk, on a Linux operating system and a 64-bit 2.8GHz processor.

8 Conclusion

We have shown how to efficiently solve a class of quantified constraints. Computational experiments show that the corresponding splitting heuristics result in efficiency improvements by orders of magnitude.

Possibilities for further research include:

- The application of the algorithm to areas such as termination analysis [4, 16, 18].
- The extension of the algorithm to a more general class of constraints, for example to ensure applicability in invariant computation [23, 26, 27].
Acknowledgments Stefan Ratschan’s work was supported by the Czech Science Foundation (GA ČR) grant number P202/12/J060 with institutional support RVO:67985807. Milan Hladík’s work was supported by the Czech Science foundation (GA ČR) grant number P402-13-106605.

References


Chapter 12

Satisfiability of Systems of Equations of Real Analytic Functions is Quasi-decidable

This chapter appeared as:


A significantly extended and revised version of the paper is currently under review:

Satisfiability of Systems of Equations of Real Analytic Functions Is Quasi-decidable

Peter Franek¹, Stefan Ratschan¹, and Piotr Zgliczynski²

¹ Institute of Computer Science, Academy of Sciences of the Czech Republic
² Jagellonian University in Krakow

Abstract. In this paper we consider the problem of checking whether a system of equations of real analytic functions is satisfiable, that is, whether it has a solution. We prove that there is an algorithm (possibly non-terminating) for this problem such that (1) whenever it terminates, it computes a correct answer, and (2) it always terminates when the input is robust. A system of equations of robust, if its satisfiability does not change under small perturbations. As a basic tool for our algorithm we use the notion of degree from the field of (differential) topology.

1 Introduction

It is well known that, while the theory of real numbers with addition and multiplication is decidable [23], any periodic function makes the problem undecidable, since it allows encoding of the integers. Recently, several papers [7,17,18,5] have argued, that in continuous domains (where we have notions of neighborhood, perturbation etc.) such decidability results do not always have much practical relevance. The reason is, that real-world manifestations of abstract mathematical objects in such domains will always be exposed to perturbations (imprecision of production, engineering approximations, unpredictable influences of the environment etc.). Engineers take these perturbations into account by coming up with robust designs, that is, designs that do not change essentially under such perturbations. Hence, in this context, it is sufficient to come up with algorithms that are able to decide such robust problem instances. They are allowed to run forever in non-robust cases, but—since robustness may not be checkable—must not return incorrect results, in whatever case. In a recent paper we called problems possessing such an algorithm quasi-decidable [19].

In this paper we show quasi-decidability of a certain fragment of the first-order theory of the reals. We allow n equalities over n variables ranging over closed intervals I₁, . . . , Iₙ, and verify the existence of a solution of the equalities in those intervals. The allowed function symbols include addition, multiplication, exponentiation, and sine. More specifically, they have to be real analytic, and for compact intervals I₁, . . . , Iₙ, we need to be able to compute an interval J ⊇ f(I₁, . . . , Iₙ) such that the over-approximation of J over f(I₁, . . . , Iₙ) can be made arbitrarily small.
Verification of zeros of systems of equations is a major topic in the interval computation community [15,20,11,8]. However, here people are usually not interested in some form of completeness of their methods, but in usability within numerical solvers for systems of equations or global optimization.

The main tool we use is the notion of the degree of a continuous function that comes from differential topology [9,13,16]. For continuous functions \( f : [a, b] \rightarrow \mathbb{R} \), the degree \( \deg(f, [a, b], 0) \) is 0 iff \( f(a) \) and \( f(b) \) have the same sign, otherwise the degree is either \( -1 \) or \( 1 \), depending on whether the sign changes from negative to positive or the other way round. Hence, in this case, the degree gives the information given by the intermediate value theorem plus some directional information. For higher dimensional functions, the degree is an integer whose value may be greater than 1, and that generalizes this information to higher dimensions. However, the degree is defined only when the dimensions of the domain and target space of \( f \) are equal.

If we can over-approximate the function \( f \) arbitrarily precisely on intervals, then the degree is algorithmically computable. Our algorithm for proving satisfiability of a function \( f \) consists of over-approximating the connected components of the zero set of \( f \) by small neighborhoods \( U_i \) and checking, whether \( \deg(f, U_i, 0) \) are zero. If any of them is nonzero, then \( f(x) = 0 \) has a solution. Otherwise we show that there exists an arbitrarily small perturbation \( \tilde{f} \) of \( f \) such that \( \tilde{f}(x) = 0 \) does not have a solution. However, such neighborhoods \( U_i \) may not exist for a general continuous or even differentiable function. Therefore, we restrict ourselves to analytic functions. The zero set of an analytic function consists of a finite number of closed connected components and we may take disjoint small neighborhoods of them.

Collins [4] presents a similar result to ours, formulated in the language of computable analysis [24]. However, the paper unfortunately contains only very rough sketch proofs, for which—from our point of view—it is not at all clear how they can be completed into complete proofs.

Since this work applies results from a quite distant field—topology—to automated reasoning, it is not possible to keep the paper self-contained within a reasonable number of pages. Still, we tried to keep the basic material self-contained and to refer to topological results only later in the paper. The necessary topological pre-requisites can be found in standard textbooks [14, e.g.].

The work of Stefan Ratschan and Peter Franek was supported by MSMT project number OC10048 and by the institutional research plan AV0Z100300504.

2 Main Theorem

In the following, we define a box \( B \) in \( \mathbb{R}^n \) to be the Cartesian product of \( n \) closed intervals of finite positive length (a hyper-rectangle). More general, we define a \( k \)-dimensional box (or \( k \)-box) in \( \mathbb{R}^n \) to be a product of \( k \) closed intervals of positive finite length and \( (n-k) \) constants.

On \( \mathbb{R}^n \), we denote the norm of a vector \( (x_1, \ldots, x_n) \) by \( |x| \) and the norm of a continuous function \( f : \Omega \rightarrow \mathbb{R}^n \) by \( ||f|| := \sup\{|f(x)|; x \in \Omega\} \). Moreover, we will denote the solution set \( \{f(x) = 0 | x \in \Omega\} \) by \( \{f = 0\} \).
For a set $\Omega \subset \mathbb{R}^n$, $\bar{\Omega}$ will be its closure, $\Omega^\circ$ its interior and $\partial \Omega = \bar{\Omega} \setminus \Omega^\circ$ its boundary with respect to the Euclidean topology. If $\Omega$ is a $k$-box in $\mathbb{R}^n$, we will usually denote $\partial \Omega$ the boundary in the topology of $\Omega$ (i.e., union of the $2k$ faces).

**Definition 1.** Let $\Omega \subseteq \mathbb{R}^n$. We call a function $f : \Omega \rightarrow \mathbb{R}$ interval-computable iff there exists an algorithm $I(f)$ that, for every box with rational endpoints such that this box is a subset of $\Omega$, computes a closed interval such that

- for every box $B \subseteq \Omega$, $I(f)(B) \supseteq \{ f(x) \mid x \in B \}$, and
- for every box $S \subseteq \Omega$, for every $\varepsilon > 0$ there is a $\delta > 0$ such that for every box $B$ with $B \subseteq S$, $\text{diam}(B) < \delta$, for all $y \in I(f)(B)$, there is an $x \in B$ such that $|f(x) - y| \leq \varepsilon$.

We call a function $f = (f_1, \ldots, f_n) : \Omega \rightarrow \mathbb{R}^n$ interval-computable iff each $f_i$ is interval-computable. In this case, the algorithm $I(f)$ returns a tuple of intervals, one for each $f_i$.

Usually such functions are written in terms of symbolic expressions containing symbols denoting certain basic functions such as rational constants, addition, multiplication, exponentiation, and sine. In that case, the first property above follows from the fundamental theorem of interval arithmetic, and the second property from Lipschitz continuity of interval arithmetic (e.g., Theorem 2.1.1 in Neumaier’s book [15]). However, in this paper we do not fix a certain notation and will allow an arbitrary language for denoting interval computable functions. We will use interval computable functions and expressions denoting them interchangeably.

Checking satisfiability of a system of equations $f_1 = 0, \ldots, f_n = 0$ in $n$ variables amounts to checking whether the function given by $f_1, \ldots, f_n$ has a zero. Since we aim at showing that this problem is quasi-decidable we have to define some notion of robustness of zeros of such functions.

**Definition 2.** We say that a continuous function $f : \Omega \rightarrow \mathbb{R}^n$ has a robust zero in $\Omega' \subset \Omega$, iff there exists an $\epsilon > 0$ such that for every continuous $g : \Omega' \rightarrow \mathbb{R}^n$, $\|g\| \leq \epsilon$ implies that $(f + g)(x) = 0$ has a solution in $\Omega'$.

We say that a continuous function $f : \Omega \rightarrow \mathbb{R}^n$ is robust iff it either has a robust zero in $\Omega$ or there exists an $\epsilon > 0$ such that for every continuous $g : \Omega \rightarrow \mathbb{R}^n$, $\|g\| \leq \epsilon$, $(f + g)(x) = 0$ does not have a solution in $\Omega$.

The definition of robustness reflects the stability of the existence of a solution with respect to small perturbations of the function. For example, the functions $f(x) = x^2 + 1$ and $g(x) = x^2 - 1$ defined on the interval $[-1, 1]$ are robust, while the function $h(x) = x^2, x \in [-1, 1]$ is not robust.

Using the notion of robustness, we can formulate the main theorem of the paper:

**Theorem 1.** Let $\Omega$ be a (closed) box in $\mathbb{R}^n$ with rational endpoints, $f : \Omega \rightarrow \mathbb{R}^n$ be interval-computable, analytic on $\Omega$, $f \neq 0$ on $\partial \Omega$. Then the existence of a zero of $f$ in $\Omega$ is quasi-decidable. That is, there exists a (possibly non-terminating) algorithm that, for such $f$ and $\Omega$, computes a value $Z$ in \{\text{T}, \text{F}\} such that
– if $Z = T$ then there exists an $x \in \Omega$ s.t. $f(x) = 0$, 
– if $Z = F$ then there does not exist an $x \in \Omega$ s.t. $f(x) = 0$, and 
– the algorithm terminates for all robust inputs.

3 Algorithm

In this section we present the algorithm whose existence implies the main theorem (i.e., quasi-decidability). The algorithm builds on the notion of degree from the field of differential topology [9,13,16]. For a smooth function $f : \Omega \rightarrow \mathbb{R}^n$ and $p \not\in f(\partial\Omega)$, the degree of $f$ with respect to $\Omega$ and a point $p \in \mathbb{R}^n$ is denoted by $\deg(f,\Omega,p)$. For further details on the degree see Section 4 below.

Let $B$ be a box and let $f : B \rightarrow \mathbb{R}^n$ be a function, nowhere zero on $\partial B$. We propose the following algorithm for proving that $f$ has a robust zero in $B$.

$\epsilon \leftarrow 1$

loop

divide $B$ into a grid of boxes of maximal side-length $\epsilon$

for each closed $(n - 1)$-dimensional face $C_i$ of a box $C$ in the grid

if $0 \in I(f)(C_i)$

merge all boxes containing $C_i$

if there exist a grid element $D$ for which $\deg(f,D,0) \neq 0$

return ”robust zero exists”

if for all grid elements $D$, $0 \not\in I(f)(D)$

return ”no zero exists”

$\epsilon \leftarrow \epsilon/2$

Due to the over-approximation property of interval arithmetic (first property of Definition 1), if the algorithm detects non-existence of a zero, this is indeed correct. This proves the second item of Theorem 1. Moreover, due to the fact that a non-zero degree implies the existence of a zero (see next section), if the algorithm detect existence of a zero, this is also correct. This proves the first item of Theorem 1. The main remaining problem is the third item, that is, to show that the algorithm will terminate (and hence detect zero existence/non-existence) for all robust inputs.

4 Degree of a Continuous Function

In this section we describe some basic properties of the degree, and show, how it can be computed. We have already mentioned in the introduction that in the one-dimensional case, the degree captures the information provided by the intermediate value theorem.

Let $\Omega \subset \mathbb{R}^n$ be open and bounded, $f : \Omega \rightarrow \mathbb{R}^n$ continuous and smooth (i.e., infinitely often differentiable) in $\Omega$, $p \not\in f(\partial\Omega)$. For regular values $p \in \mathbb{R}^n$ (i.e., values $p$ such that for all $y$ with $f(y) = p$, $\det f'(y) \neq 0$), a generalization of
the directional information used in the one-dimensional case, is the sign of the
determinant \( \det f'(y) \). Adding up those signs results in the explicit definition [13]
of \( \deg(f, \Omega, p) \) by \( \deg(f, \Omega, p) := \sum_{y \in f^{-1}(p)} \text{sign } \det f'(y) \).

See standard textbooks for a generalization to non-regular values [13]. Here
we give an alternative, axiomatic definition, that can be shown to be unique. In
this approach \( \deg(f, \Omega, p) \) is the unique integer satisfying the following proper-
ties [6,16, e.g.]:

1. For the identity function \( I \), \( \deg(I, \Omega, p) = 1 \) iff \( p \in \Omega \)
2. If \( \deg(f, \Omega, p) \neq 0 \) then \( f(x) = p \) has a solution in \( \Omega \)
3. If there is a continuous function (a “homotopy”) \( h : [0,1] \times \Omega \to \mathbb{R}^n \) such
   that \( h(0) = f, \ h(1) = g \) and \( p \notin h(t, \partial \Omega) \) for all \( t \), then \( \deg(f, \Omega, p) =
   \deg(g, \Omega, p) \)
4. If \( \Omega_1 \cap \Omega_2 = \emptyset \) and \( p \notin f(\partial \Omega_1 \cup \partial \Omega_2) \), then \( \deg(f, \Omega_1 \cup \Omega_2, p) = \deg(f, \Omega_1, p) +
   \deg(f, \Omega_2, p) \)
5. \( \deg(f, \Omega, p) \), as a function of \( p \), is constant on any connected component of
\( \mathbb{R}^n \setminus f(\partial \Omega) \).

This can be extended to the case where \( \Omega \) has dimension \( n \) but is is embedded
into some higher-dimensional space (in topological terms, \( f \) is a differentiable
function between two compact oriented manifolds of the same dimensions). For
example, if \( f \) is a function from a segment \( c \) of a curve (i.e., a set of dimension
1) in \( \mathbb{R}^k \) to another segment of a curve in \( \mathbb{R}^k \), and if \( f \neq 0 \) on the endpoints of
\( c \), then \( \deg(f, c, 0) \) is well defined.

The literature provides several articles [22,2,1, e.g.] that claim to provide an
algorithm that automatically computes the topological degree. However, they
either just contain informal recipes, or require real-number operations for which
it is not clear how to implement them on computers, or whose correctness relies
on unknown Lipschitz constants. In order to clarify the situation, we give an
algorithm here that is based on ideas readily available in the literature, but that
does not have those deficiencies.

The algorithm is based on a theorem that recursively reduces the computation
of the degree wrt. to a \( k \)-dimensional box to the computation of the degree wrt.
to a \( (k-1) \)-dimensional box. The theorem uses the notion of orientation that has
a specific meaning in differential topology [9,13,16]. In order to make the material
digestible to a more general audience, and in order to demonstrate algorithmic
implementability, we describe here an equivalent, but simpler formalization for
the special case of boxes (instead of general manifolds).

We define the orientation of a box in \( \mathbb{R}^n \) to be a sign \( s \in \{1, -1\} \). Let us
consider a \( k \)-dimensional box \( B \) with orientation \( s \). Observe that we can obtain
faces of \( B \) by replacing one of the intervals constituting \( B \) by either its lower or
upper bound (the resulting face is a \( (k-1) \)-dimensional box). Assume that this
interval is the \( r \)-th (non-constant) interval of \( B \) (so \( r \in \{1, \ldots, k\} \)). Then, if the
face results from taking a lower bound, we define the induced orientation of the
face to be \((-1)^rs\), if it results from taking an upper bound, the orientation is
\((-1)^{r+1}s\).
Let \( D \) be a finite union of oriented \( k \)-boxes. The orientation of a union of oriented boxes is, for our purposes, just the information about the orientation of each box in the union. The induced orientation of \( \partial D \) is the set \( \partial D \), consisting of \( k - 1 \)-dimensional boxes with orientations induced from the boxes in \( D \).

**Theorem 2.** Let \( B \) be an oriented finite union of \( n \)-dimensional boxes with connected interior. Let \( f : B \to \mathbb{R}^n \) be continuous such that \( f \neq 0 \) on the boundary of \( B \). Let \( D_1, \ldots, D_k \) be disjoint subsets of the boundary of \( B \) such that each \( D_i \) is a finite union of \((n - 1)\)-dimensional boxes and the interior of \( D_i \) in \( \partial B \) is connected. We denote the boundary of \( D_i \) in the topology of \( \partial B \) by \( \partial D_i \).

The orientation of \( B \) induces an orientation of each \( D_i \), \( i \in \{1, \ldots, k\} \).

For \( r \in \{1, \ldots, k\} \) we denote by \( f_r \) the \( r \)-th component of \( f \) and \( f_{-r} := (f_1, \ldots, f_{r-1}, f_{r+1}, \ldots, f_n) \).

Now let \( r \in \{1, \ldots, n\} \), and \( s \in \{-1, 1\} \) such that

- for all \( i \in \{1, \ldots, k\} \), \( f_r \) has constant sign \( s \) in \( D_i \),
- \( \bigcup_{i \in \{1, \ldots, k\}} D_i \) contains all zeros of \( f_{-r} \) for which \( f_r \) has sign \( s \), and
- for all \( i \in \{1, \ldots, k\} \), \( 0 \notin f_{-r}(\partial D_i) \).

Then

\[
\deg(f, B, 0) = (-1)^{r-1} s \sum_{i \in \{1, \ldots, k\}} \deg(f_{-r}, D_i, 0).
\]

Finally, for a one dimensional closed, connected union of oriented boxes \( D^1 \) of \( B \) with left-most face \( l \) and right-most face \( r \) (according to orientation)

\[
\deg(f, D^1, 0) = \begin{cases} 1 & \text{if } f(l) < 0 < f(r) \\ -1 & \text{if } f(r) < 0 < f(l) \\ 0 & \text{if } f(l)f(r) > 0 \end{cases}
\]

Observe that one-dimensional boxes have two faces (that are points) with opposite induced orientation. We define the **left** face to be the face with the opposite induced orientation as the original box, and the **right** face to be the face with the same induced orientation.

When starting the recursion in the above theorem with an \( n \)-dimensional box in \( \mathbb{R}^n \) of orientation 1, if in the base case \( D^1 \) consists of more than one box, then every left face of a box in \( D^1 \) is either a boundary point, or the right face of another box. This makes the notion of a left-most face \( l \) and right-most face \( r \) of \( D^1 \) well-defined.

The theorem follows directly from Kearfott [10, Theorem 2.2], which again is a direct consequence of some results of Stenger [22].

Now the algorithm just recursively reduces the computation of the topological degree in dimension \( n \) to lower dimension using the above theorem. In each recursive step, for an arbitrary choice for \( r \) and \( s \), it computes sets \( D_1, \ldots, D_k \) fulfilling the necessary conditions. If \( f \) is analytic, \( \{f_r = 0\} \) and \( \{f_{-r} = 0\} \) have a finite number of connected components (see next Section) and the sets \( D_i \) can be found by computing an increasingly fine decomposition of the boundary of \( B \).
into boxes, and checking the necessary conditions using interval arithmetic. Due to the second property in Definition 1 this procedure will eventually approximate \( f \) on the boundary of \( B \) arbitrarily closely, and hence it will eventually find such a decomposition.

5 Zeros of Analytic Functions

A function \( f : \Omega \rightarrow \mathbb{R}^n \) defined on an open set \( \Omega \) is analytic, iff each of its components \( f_1, \ldots, f_n \) is an analytic function, that is, iff for each \( i \in \{1, \ldots, n\} \), for each point \( x_0 \in \Omega \), there exists a neighborhood \( U \) of \( x_0 \) and a sequence of numbers \( \{c_j\}_{j \in \mathbb{N}_0} \) such that \( f_i = \sum_{j \in \mathbb{N}_0} c_j (x - x_0)^j \) on \( U \). The set of analytic functions is closed with respect to addition, multiplication, division by nonzero function, composition and differentiation.

We will need the following statement later:

**Theorem 3.** For an analytic function \( f : \Omega \rightarrow \mathbb{R}^n \), \( 0 \not\in f(\partial \Omega) \), the set \( \{ f = 0 \} \) consists of a finite number of connected components.

**Proof.** It follows from Lojasiewicz’s theorem [12] that the zero set of a real valued analytic functions is locally a union of a finite number of manifolds of various dimensions. So, the zero set of a real valued analytic function defined on a compact set \( \bar{\Omega} \) has a finite number of connected components and the set \( \{ f = 0 \} \) coincides with the zero set of the real valued analytic function \( \sum_i f_i^2 \) on \( \bar{\Omega} \).

An analogous statement for smooth (but not analytic) functions \( f \) does not hold. One can easily construct a smooth function \( f : [0,1] \rightarrow \mathbb{R} \) such that \( \{ f = 0 \} \) is the Cantor set which is totally disconnected.

6 Degree and Robustness

In this section, we will clarify the connection between robust solution of a function \( f \) and the degree of \( f \). First, we prove that if \( \deg (f, \Omega, 0) \neq 0 \), then \( f \) has a robust zero in \( \Omega \). We will use the rest of the section to prove a partial converse of this. We will show that if the degree is zero and the set of solution \( f = 0 \) is connected, then \( f \) does not have a robust zero in \( \Omega \). This will be used as a main ingredient in the proof of the main theorem, given in the next section.

**Theorem 4.** Let \( \Omega \subset \mathbb{R}^n \) be an open, and bounded set. Let \( f : \Omega \rightarrow \mathbb{R}^n \) be continuous and smooth on \( \Omega \), \( 0 \not\in f(\partial \Omega) \) and let \( \deg (f, \Omega, 0) \neq 0 \). Then \( f \) has a robust zero in \( \Omega \).

**Proof.** Let \( \epsilon < \min_{x \in \partial \Omega} |f| \). For any \( g \) such that \( ||g - f|| < \epsilon \), we define a homotopy \( h(t, x) = tf(x) + (1-t)g(x) \) between \( f \) and \( g \). We see that for \( x \in \partial \Omega \) and \( t \in [0,1] \),

\[
|h(t, x)| = |tf(x) + (1-t)g(x)| = |f(x) + (1-t)(g(x) - f(x))| \geq |f(x)| - \epsilon > 0
\]

so that \( h(t, x) \neq 0 \) for \( x \in \partial \Omega \). From Section 4, properties 2 and 3, we see that \( g(x) = 0 \) has a solution. \( \square \)
We will now consider the case when the degree is zero.

**Lemma 1.** Let $B$ be homeomorphic to an n-dimensional ball, $f : B \to \mathbb{R}^n$ be continuous, nowhere zero on $\partial B$ and let $\deg (f, B, 0) = 0$. Then there exists a continuous nowhere zero function $g : B \to \mathbb{R}^n$ such that $g = f$ on $\partial B$ and $\|g\| \leq 2\|f\|$.

**Proof.** We may assume, without loss of generality, that $B = \{x \in \mathbb{R}^n; |x| = 1\}$ and $\partial B = S^{n-1}$ is the $(n-1)$-sphere. Let $R : \mathbb{R}^n \setminus \{0\} \to S^{n-1}$ be defined by $R(x) := x/|x|$. The degree $\deg (f, B, 0) = 0$ is equal to the degree of the function $R \circ f|_{S^{n-1}} : S^{n-1} \to S^{n-1}$ The Hopf theorem ([13, pp. 51]) states that the degree classifies continuous self-functions of a sphere up to homotopy. So, $R \circ f|_{S^{n-1}}$ is homotopy equivalent to a constant map. So, there exists a homotopy $F : [0, 1] \times S^{n-1} \to S^{n-1}$ such that $F(1, x) = (R \circ f)(x)$ and $F(0, x) = c \in S^{n-1}$. Let $r \in [0, 1]$ and $x \in S^{n-1}$. Define the function $g : B \to \mathbb{R}^n$ by

$$g(rx) = F(r, x)(r|f(x)| + (1-r)||f||).$$

This function is continuous, nowhere zero and well defined because $g(0x) = g(0) = c\|f\|$ is independent of $x \in S^{n-1}$. Clearly, $g(x) = f(x)$ on $S^{n-1}$ and $\|g(rx)\| \leq r|f(x)| + (1-r)||f|| \leq 2||f||$. □

Further, we will need the following technical lemma:

**Lemma 2.** Let $\Omega \subset \mathbb{R}^n$ be open and bounded, $f : \bar{\Omega} \to \mathbb{R}^n$ continuous and smooth on $\Omega$, $0 \notin f(\partial \Omega)$. Then there exists a continuous function $\tilde{f} : \bar{\Omega} \to \mathbb{R}^n$, smooth on $\Omega$, with the following properties:

1. $\tilde{f} = f$ on $\partial \Omega$,
2. 0 is a regular value of $\tilde{f}$,
3. $\|\tilde{f}\| \leq 2\|f\|$,
4. $\tilde{f}$ is homotopy equivalent to $f$ due to 0 a homotopy $h(t)$ such that $0 \notin h(t)(\partial \Omega)$.

**Proof.** Let $U$ be an open neighborhood of $\partial \Omega$ in $\bar{\Omega}$ such that $f \neq 0$ on $U$ and $\min\{|f(x)|; x \in U\} = \epsilon > 0$. From Sard’s theorem ([13]), there exists a regular value $x_0$ of $f$ with $|x_0| < \epsilon/2$. It follows that 0 is a regular value of the function $f(x) - x_0$. Consider a covering of $\partial \Omega$ by open sets $\{U_\alpha; \alpha \in \Lambda_1\}$ such that $U_\alpha \subset U$ for all $\alpha \in \Lambda_1$. As $\partial \Omega$ is compact, we may assume that $\Lambda_1$ is finite. Further, define a covering of $\Omega$ by open sets $\{U_\beta; \beta \in \Lambda_2\}$. For $\Lambda = \Lambda_1 \cup \Lambda_2$, we have a covering $\{U_\alpha; \alpha \in \Lambda\}$ of the compact space $\bar{\Omega}$. Let $\{\rho_\alpha; \alpha \in \Lambda\}$ be the subordinated partition of unity consisting of continuous functions smooth in $\bar{\Omega}$ and define $\phi(x) := \sum_{\alpha \in \Lambda, \rho_\alpha \neq 0} \phi(x)$. Then $\phi$ is a smooth function supported in $\bar{U}$ such that $\phi = 1$ on $\partial \Omega$ and $\phi = 0$ on $\bar{\Omega} \setminus U$. Define $\tilde{f}(x) = f(x) - (1 - \phi(x))x_0$. Clearly, $\tilde{f} = f$ on $\partial \Omega$. The function $\tilde{f}$ is nowhere zero on $\bar{U}$, because $|\tilde{f}(x)| \geq |f(x)| - |x_0| \geq \epsilon/2$ on $U$. On $\bar{\Omega} \setminus U$, $\tilde{f}(x) = f(x) - x_0$. In particular, 0 is a regular value of $\tilde{f}$ and $\|\tilde{f}\| \leq \|f\| + |x_0| \leq 2\|f\|$. Finally, a homotopy between $f$ and $\tilde{f}$ may be given by $h(t) = f(x) + (1-t)f(x)$. □
We have seen in Lemma 1 that if the degree of $f$ on a ball is zero, then we may define a nowhere zero function on the ball that coincides with $f$ on the boundary. We will now see that this is true not only for a ball, but for any connected bounded and open set $\Omega \subset \mathbb{R}^n$.

**Lemma 3.** Let $\Omega$ be a connected, open, bounded subset of $\mathbb{R}^n$, $f : \Omega \rightarrow \mathbb{R}^n$ continuous and smooth on $\Omega$, $0 \notin f(\partial \Omega)$ and $\deg(f, \Omega, 0) = 0$. Then there exists a continuous nowhere zero function $g : \Omega \rightarrow \mathbb{R}^n$ such that $g = f$ on $\partial \Omega$ and $||g|| \leq 4||f||$.

**Proof.** If the dimension $n = 1$, the function $f$ is defined on an interval $[a,b]$ and the degree assumption implies that $f(a)$ and $f(b)$ have equal signs. So, we may define $g = f(a) + (x-a)(f(b) - f(a))/(b-a)$ and the lemma is proved. Assume further that $n \geq 2$. From lemma 2, we construct a continuous function $\tilde{f} : \Omega \rightarrow \mathbb{R}^n$ smooth on $\Omega$, $f = \tilde{f}$ on $\partial \Omega$, $||\tilde{f}|| \leq 2||f||$, having 0 as a regular value, homotopy equivalent to $f$. In particular, $\deg(f, \Omega, 0) = 0$.

The compactness of $\Omega$ implies that $\tilde{f}^{-1}(0)$ is finite. Because $\deg(\tilde{f}, \Omega, 0) = 0$, we may enumerate the points in $\tilde{f}^{-1}(0)$ as $\{x_1, \ldots, x_{2m}\}$ so that $\tilde{f}$ is orientation-preserving in the neighborhoods of $x_1, x_2, \ldots, x_m$ and orientation-reversing in the neighborhoods of $x_{m+1}, \ldots x_{2m}$.

Choose $m$ smooth, pairwise disjoint, non-self-intersecting curves $c_i$ in $\Omega$ connecting $x_i$ and $x_{m+i}$. This is possible, because the dimension $n \geq 2$ and the complement of a smooth non-self-intersecting curve in an open connected set $\Omega \subset \mathbb{R}^n$ is still open and connected. For these smooth curves, there exist disjoint neighborhoods homeomorphic to balls $B_1, \ldots, B_m$ (see e.g. [13, Product neighborhood theorem]). Because $\deg(\tilde{f}, B_i, 0) = 1 - 1 = 0$, we may apply lemma 1 to construct nowhere zero functions $g_i : B_i \rightarrow \mathbb{R}^n$ such that $g_i = \tilde{f}$ on $\partial B_i$ and $|g_i(x)| \leq 2||\tilde{f}|| \leq 4||f||$. The resulting function $g(x)$ defined by $g = g_i$ on $B_i$ and $f$ elsewhere is continuous and has the properties required.

We now prove a partial conversion of Theorem 4.

**Theorem 5.** Let $\Omega$ be open, connected, bounded, $f : \bar{\Omega} \rightarrow \mathbb{R}^n$ continuous and smooth on $\Omega$, $0 \notin f(\partial \Omega)$. Let $f$ have a robust zero in $\Omega$ and assume that the set $\{f = 0\} \subset \Omega$ is connected. Then $\deg(f, \Omega, 0) \neq 0$.

**Proof.** Let $\epsilon > 0$. Since $\{f = 0\}$ is connected, is is contained in a single connected component $\Omega'$ of the open set $\{x : |f(x)| < \epsilon\}$. Let $\deg(f, \Omega, 0) = 0$. Applying lemma 2 to the set $\Omega'$, we construct a continuous function $\tilde{f} : \bar{\Omega}' \rightarrow \mathbb{R}^n$ smooth on $\Omega'$, homotopy equivalent to $f : \bar{\Omega}' \rightarrow \mathbb{R}^n$, $\tilde{f} = f$ on $\partial \Omega'$, having 0 as a regular value and $||\tilde{f}|| \leq 2||f||_{\bar{\Omega}'} \leq 2\epsilon$. Because the set $\{f = 0\}$ is connected and contained in $\Omega'$, we obtain that $\deg(f, \Omega, 0) = \deg(f, \Omega', 0) = \deg(\tilde{f}, \Omega', 0) = 0$. Using lemma 3, we obtain a continuous function $g : \Omega' \rightarrow \mathbb{R}^n$ such that $g = \tilde{f}$ on $\partial \Omega'$, $g \neq 0$ on $\Omega'$ and $|g| \leq 2||\tilde{f}|| \leq 4\epsilon$ on $\Omega'$. Extending $g$ to all $\bar{\Omega}$ by $g = f$ on $\bar{\Omega} \setminus \Omega'$, we obtain an everywhere nonzero continuous function $g$ such that $||g - f|| \leq 5\epsilon$. This can be done for any $\epsilon$ and it follows that $f$ has no robust zero in $\Omega$.
7 Proof of the Main Theorem

As we have seen in Section 3, the algorithm presented there—if it terminates—will always correctly detect the existence of a zero. The main remaining problem for proving quasi-decidability, was termination of this algorithm for robust inputs. In the positive case of existence of a zero, we will actually prove that this holds in both directions:

**Theorem 6.** Let \( B \) be a box in \( \mathbb{R}^n \), \( f : B \to \mathbb{R}^n \) be continuous in \( B \) and analytic in the interior of \( B \), \( f \neq 0 \) on \( \partial B \). Assume that \( f \) has a zero in \( B \). Then the algorithm proposed in Section 3 terminates with the output “robust zero exists” if and only if \( f \) has a robust zero in \( B \).

**Proof.** First, if the algorithm terminates, then \( \deg (f,D,0) \neq 0 \), for some union of boxes \( D \subset B \) and it follows from Theorem 4 that \( f \) has a robust zero in \( D \). So, \( f \) has a robust zero in \( B \).

Suppose now that the function \( f \) has a robust zero in \( B \). We know from section 5 that for an analytic function \( f \), there exists a finite number of connected components of \( \{f = 0\} \), so any two of them have positive distance. Let as denote the component by \( Z_1, \ldots, Z_m \). Let \( U_1, \ldots U_m \) be open connected neighborhoods of \( Z_1, \ldots, Z_m \) such that \( \bar{U}_1, \ldots \bar{U}_m \) are disjoint. If \( \deg (f,\bar{U}_i,0) = 0 \), it follows from Theorem 5 that for any \( \epsilon > 0 \), there exists a continuous function \( g_i : \bar{U}_i \to \mathbb{R}^n \) such that \( g_i \) is nowhere zero, \( ||g_i - f|_{\bar{U}_i}|| \leq \epsilon \) and \( g_i = f \) on \( \partial \bar{U}_i \). Replacing \( f \) by \( g_i \) on \( \bar{U}_i \), we would obtain a continuous nowhere zero function \( g : B \to \mathbb{R}^n \), \( ||g - f|| \leq \epsilon \), contradicting the assumption that \( f \) has a robust zero in \( \Omega \). So, at least for one \( i \), \( \deg (f,\bar{U}_i,0) \neq 0 \). Because \( \partial \bar{U}_i \) is compact, it has positive distance \( d > 0 \) from \( Z_i \). Let as assume that \( \epsilon \leq d/(\sqrt{n}) \). If we cover \( B \) with a grid of boxes of side-length smaller then \( \epsilon \), any box that has nonempty intersection with \( Z_i \) is contained in \( U_i \). Possibly merging the boxes whose boundaries intersect \( Z_i \) (if there are any), we obtain a set \( D \subset \cup U_i \) such that the interior of \( D \) is a neighborhood of \( Z_i \). Because the sets \( U_i \) are disjoint, the zero set of \( f \) in \( U_i \) is just \( Z_i \). So, \( \deg (f,D,0) = \deg (f,\bar{U}_i,0) \neq 0 \) and the algorithm terminates.

In the case where the input \( f \) does not have a zero, termination for robust inputs holds due to the second property of \( I(f) \) in Definition 1. The main theorem of this paper follows as a corollary.

8 Possible Generalizations

We needed analyticity of \( f \) only to be sure that \( \{f = 0\} \) decomposes into a finite number of connected components. If \( f \) has this property, (e.g. if \( \{f = 0\} \) is discrete), the algorithm terminates for a robust input.

We will now discuss the generalization of the theorem to the case where the number of variables is not equal to the number of equations. That is, we have the situation where we have an \( n \)-dimensional box \( B \) in \( \mathbb{R}^n \), \( f : B \to \mathbb{R}^m \) analytic. In the case were \( n < m \) (i.e. the number of variables is smaller then the number
of equations) it follows from Sard’s theorem, that there exist regular values $y \in \mathbb{R}^m$ arbitrarily close to 0. So, some neighborhood $U$ of $f^{-1}(y)$ is mapped homeomorphically to $f(U) \subseteq \mathbb{R}^n$. But an $n$-dimensional open set can not be homeomorphic to an open set in $\mathbb{R}^m$ for $m \neq n$, so $f(U)$ does not contain a neighborhood of $y$. So, $0 \in \mathbb{R}^m$ is not in the interior of the image of $f$ and $f$ cannot have a robust zero in $B$.

On the other hand, for $n > m$ (more variables than equations), the situation is much more subtle. In some cases, we could fix some $n - m$ input variables in $f$ to be constants $a \in \mathbb{R}^{n-m}$ and try to solve the equation $f(a, x) = 0$. This is a function from a subset of $\mathbb{R}^m$ to $\mathbb{R}^m$ and if it has a robust zero in $\{x \in \mathbb{R}^m; (a, x) \in B\}$, so, clearly, $f$ has a robust zero in $B$. The converse, however, is not true. If $f(a, x)$ does not have a robust zero for any fixed choice of $a \in \mathbb{R}^{n-m}$ ($a$ ranging from any subset of $m - n$ from the total number of $n$ variables), $f$ still may have a robust zero in $B$. This makes a straightforward generalization of our result difficult.

The ideas presented in Section 6 may be easily generalized to arbitrary dimensions, if the condition of a nonzero degree is replaced by the condition ”the map $f/f$ from $\partial \Omega$ to the sphere $S^{m-1}$ can be extended to a map from $\Omega$ to $S^{m-1}$”. This is the extension problem in computational homotopy theory. If $\Omega$ is the unit ball in $\mathbb{R}^2$, $\partial \Omega = S^1$ and $X$ is an arbitrary space, the question whether or not $f : \partial \Omega \to X$, presented algorithmically, can be extended to $f : \Omega \to X$ is equivalent to the word problem and there is no algorithm to solve it [21]. The question whether or not such an algorithm exists for $X$ being the sphere and $\Omega$ arbitrary, is—up to our knowledge—an open problem.

9 Conclusion

In the paper, we have proved that the problem of checking satisfiability of systems of equations of real analytic functions in a box is quasi-decidable in the sense that there exists an algorithm that successfully can do this check in all robust cases. Hence, problems that correspond to application domains where perturbations in the form of modeling errors, manufacturing imprecision etc. occur, are solvable in practice (provided enough computing power is available).

The generalization to the full first-order case is an open problem.

References