An Efficient Delta-decision Procedure

Soonho Kong

CMU-CS-23-114

May 2023

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Edmund M. Clarke, Co-Chair
Randal E. Bryant, Co-Chair
Jeremy Avigad
Marijn J.H. Heule
Leonardo de Moura (Amazon Web Services)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by the Office of Naval Research under award number N000141010188, the National Science Foundation under award numbers CNS-1035813 and CNS-1330014, the Air Force Research Laboratory under award numbers FA95501210146 and FA955015C0030, and Genera Motors. Soonho Kong was in part supported by Kwanjeong Educational Foundation.
The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Satisfiability Modulo Theory, Ordinary Differential Equations, Decision Procedures, Delta-decidability, Hybrid Systems, Reachability, Numerical Optimization, Bounded Model Checking

Abstract

Delta-complete analysis demonstrates the decidability and complexity of delta-complete decision procedures through appropriate relaxations of exact decision problems. This framework presents a potential for addressing various practical problems in science and engineering involving high-order polynomials, transcendental functions, and ordinary differential equations. However, significant challenges remain in the development of viable and practical delta-decision procedures.

This dissertation aims to address the challenge of designing and implementing a scalable delta-decision procedure that incorporates rich theories and support for quantifiers, as well as a bounded reachability analysis tool that is based on such a procedure.

First, we propose algorithms for solving SMT problems that involve ordinary differential equations (ODEs) by utilizing ODE constraints to design pruning operators within a branch-and-prune framework. Furthermore, we prove the delta-completeness of our algorithms.

Second, we present algorithms for solving SMT problems that involve universal quantification and a broad range of nonlinear functions by integrating interval constraint propagation, counterexample-guided synthesis, and numerical optimization. The proposed algorithms are demonstrated to be effective in handling a wide range of challenging global optimization and control synthesis problems.

Finally, we present dReal and dReach, a delta-SMT solver and a delta-reachability analysis tool respectively, for nonlinear real formulas and hybrid systems. dReal is capable of handling various nonlinear real functions, such as polynomials, trigonometric functions, and exponential functions, and implements the delta-complete decision procedure framework. dReach, on the other hand, encodes reachability problems as first-order real formulas and solves them using dReal. As a result, dReach is equipped to handle a wide range of highly nonlinear hybrid systems, as demonstrated by its scalability on various realistic models from biomedical and robotics applications.

Acknowledgements

I am profoundly grateful to my advisor, Edmund Clarke, for his unwavering support, guidance, and invaluable contributions to my work. His mentorship has instilled in me the importance of perseverance and has played a crucial role in shaping my academic journey. I am forever indebted to him for his guidance.

I also wish to convey my sincere appreciation to Randy Bryant, who took on the role of my advisor following the untimely passing of Edmund Clarke in 2020. His consistent support, encouragement, and guidance have been indispensable in helping me complete this work. This milestone would not have been possible without his invaluable assistance.

My sincere gratitude goes to Jeremy Avigad for his invaluable input in my PhD research. Along with Sicun Gao and Edmund Clarke, he introduced the concept of delta-decidability, which laid the groundwork for my research. I am grateful for his role as a committee member, and his guidance and astute feedback significantly influenced my work.

I am honored to have Marijn Heule as my committee member. His insightful comments on my thesis were invaluable, and I am grateful for the discussions and feedback on my thesis work as well as research projects at Amazon Web Services.

During my PhD program, I was fortunate to have two internships at Microsoft Research, where I had the privilege of working under Leonardo de Moura's mentorship. This opportunity allowed me to contribute to the development of Lean and benefit from his guidance and expertise. I am grateful for his invaluable insights and knowledge, which he generously shared as a committee member.

I am immensely grateful for the opportunity to collaborate with numerous exceptional individuals during my PhD. I extend special thanks to Sicun Gao, who was not only an outstanding collaborator but also a dear friend. The countless discussions we shared in GHC9232 will always remain a fond memory. I am also

grateful to Nikos Arechiga, Kyungmin Bae, Sagar Chaki, Wei Chen, Arie Gurfinkel, James Kapinski, Md. Ariful Islam, Bing Liu, Armando Solar-Lezama, Nima Roohi, Qinsi Wang, Paolo Zuliani, and others for their fruitful discussions and perceptive feedback.

I would like to express my gratitude to my friends at CMU, including Athula Balachandran, Favonia, Dongsu Han, Hanbyul Joo, U Kang, Gunhee Kim, Jinkyu Kim, Will Kileber, Anvesh Komuravelli, Jayyoon Lee, Seunghak Lee, Marius Minea, Wolfgang Richter, and Samir Sapra. Their companionship and support have been invaluable throughout my journey. Additionally, I am grateful to CMU staff members Deborah A. Cavlovich, Catherine Copetas, Jenn Landefeld, Todd Seth, Charlotte Yano, and the rest of the excellent staff for their assistance and support during my academic pursuits. Special thanks go to Martha Clarke, who has consistently shown care and concern for both me and my family.

In 2016, I took a hiatus from my PhD studies at CMU and embarked on a professional journey with Toyota Research Institute and then Amazon Web Services. I extend my gratitude to my colleagues and employers during this time for their support and collaboration. Particularly, I would like to express my appreciation for my leaders at AWS: Byron Cook, Robert Jones, and Leah Daniels. Their encouragement and guidance have been invaluable in helping me balance my professional responsibilities with the completion of my PhD thesis.

Finally, I extend my gratitude to my family members for their unwavering support throughout my journey: Jungkyu Kong, Jungae Yu, Youngeun Kong, Sunil Jung, and Ethan Kong. Your love truly makes this journey worthwhile.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Motivation and Scope

The objective of this dissertation is to demonstrate steps taken to bridge the gap between the theoretical possibility and the practical realization of an efficient delta-decision procedure. In their seminal papers [48, 49], Gao, Avigad, and Clarke introduced the notion of delta-decision problem for bounded first-order sentences over the reals with computable functions. By allowing for an appropriate relaxation of the exact decision problems, they proved the decidability of delta-decision problems and presented their complexity results. However, despite this theoretical breakthrough, the design and implementation of an efficient delta-complete decision procedure remained a challenge to be addressed in future work as noted in [49]:

> "In future work, it would be very interesting to see how this framework can be used in developing efficient SMT/SAT solvers and theorem provers."

The main scope of this dissertation is to address the challenge of designing and implementing a scalable delta-decision procedure that incorporates rich theories and quantifier support.

Solving delta-SMT Problems with ODE Constraints The formal verification of hybrid systems [5] is a challenging task that has been the focus of research in recent years. Hybrid systems [64] consist of both discrete transitions and continuous dynamics and can be found in various domains, such as automotive systems [73, 7], embedded systems [83], and biological systems [86, 87]. One of the key challenges in the verification of these systems is the need to reason about the interactions between the discrete and continuous dynamics. However, traditional SMT solvers are not equipped to handle ODEs, which are a fundamental component of many hybrid systems.

To address this challenge, we propose to investigate a method for solving delta-SMT problems that incorporate ODE constraints, with a specific focus on supporting nonlinear Lipschitz-continuous ODEs. This is motivated by the fact that nonlinear dynamics are commonly found in real-world systems. For example, a non-linear 7 degree of freedom bicycle model [101] is widely used to describe the dynamics of vehicles in the field of autonomous vehicles. In this model, the state vector $\vec{x}$ describing the vehicle consists of 7 components, $(\beta, \psi, \dot{\psi}, v, s_x, s_y, \delta)$ where β is the slip angle at the center of mass, ψ is the heading angle, $\dot{\psi}$ is the yaw rate, v is the velocity, s_x is the vehicle's x position, s_y is the vehicle's y position, and δ is the angle of the front wheel. Its continuous dynamics is governed by the following nonlinear ODEs:

$$
\begin{aligned}
\dot{\beta} &= \left(\frac{C_r l_r - C_f l_f}{m v^2}\right)\dot{\psi} + \left(\frac{C_f}{mv}\right)\delta - \left(\frac{C_f + C_r}{mv}\right)\beta \\
\ddot{\psi} &= \left(\frac{C_r l_r - C_f l_f}{I_z}\right)\beta - \left(\frac{C_f l_f^2 - C_r l_r^2}{I_z}\right)\left(\frac{\dot{\psi}}{v}\right) + \left(\frac{C_f l_f}{I_z}\right)\delta \\
\dot{v} &= a_x \\
\dot{s_x} &= v\cos(\beta + \psi) \\
\dot{s_y} &= v\sin(\beta + \psi) \\
\dot{\delta} &= v_w
\end{aligned}
$$

where C_r, C_f, l_r, l_f, m, I_z, a_x, v_w are constant values for a vehicle. A vehicle planning algorithm may be modeled as a hybrid system by combining a continuous vehicle

dynamics and discrete control decisions. As demonstrated in [97], a lane changing planner can be represented using two discrete states — one for lane following, the other for lane changing. The bounded safety property of the hybrid system model is then established via bounded reachability analysis.

By developing SMT solvers that can handle nonlinear ODEs, we aim to enable the efficient verification of real-world hybrid systems. This is crucial for ensuring the correct operation of these systems in various domains. By supporting nonlinear ODEs, SMT solvers can be used to reason about the interactions between the discrete and continuous dynamics, and check the reachability and safety of the system.

Solving Exists-Forall Formulas in the delta-decision Framework Next, we delve into the challenge of solving exists-forall formulas within the delta-decision framework. This problem is not only important, but also intriguing, as it has the potential to revolutionize the way we tackle general optimization problems. General optimization problems, including non-convex, multi-objective, and disjunctive optimization problems, can all be represented as exists-forall formulas. Thus, by solving exists-forall problems, we can also solve these general optimization problems. While numerical optimization techniques can be effective, they cannot guarantee optimality in all cases. Our approach, on the other hand, offers a guarantee in the form of a delta-close solution to the optimal solution, providing a more reliable solution for these complex problems.

Synthesis problems are another domain where exists-forall encodings can be effectively utilized. A prime example of this is the synthesis of Lyapunov functions for dynamical systems. Lyapunov functions [82] play a critical role in demonstrating the stability of a dynamical system, as they provide a systematic way to measure the amount of energy stored in the system, and thus, its stability. By using exists-forall encodings to represent the synthesis problem, we can leverage the delta-decision framework to systematically search for a Lyapunov function that meets certain criteria and guarantees stability. In this way, our approach has the potential to

greatly simplify the process of verifying the stability of complex dynamical systems.

Design and Implementation of delta-decision Tools The state-of-the-art SMT solvers, such as Z3 [34], CVC [10, 9], and Yices [38, 37], have had a significant impact on the field of automated reasoning and formal verification. They provide an accessible and flexible platform for researchers and practitioners to perform complex logical computations in various applications, such as software verification, hardware verification, and optimization. These solvers are widely adopted due to their ability to handle various theories and logics, including linear arithmetic, arrays, and bit-vectors, among others. Additionally, the open-source nature of these solvers allows for the community to actively contribute to and improve their performance and functionality.

Our goal is to provide an open-source implementation of a delta-SMT solver that is capable of handling a broad spectrum of nonlinear functions and supports quantifier reasoning. This will enable the tool to be utilized by a multitude of applications and tools, further fostering the growth of formal verification techniques in various domains.

Building upon this idea, we present the design and implementation of the delta-decision procedure, dReal, and the delta-reachability analysis tool, dReach, which is built on top of dReal. The efficient implementation of dReal and dReach is crucial because numerous tools, including APEX [97], BioPSy [90], Daisy [33], DiffRNN [91], Drake [111], ETCetera [35], FOSSIL [1], HybridSyncAADL [84], JDart [89], LinSyn [99], Manifold [14], PGCD [8], PolyReach [110], ProbReach [107], STLMC [118], Sally [39], Verisig [69], Viatra [106], SReach [115], and symQV [12] have been developed based on dReal / dReach tools. The performance and functionality of these dependent tools are directly tied to the efficiency of the underlying tools, dReal and dReach.

1.2 Thesis Statement and Contributions

The thesis statement can be summarized as follows:

> An efficient delta-decision procedure can solve real-world problems with precision and computational efficiency, including those involving nonlinear functions, ordinary differential equations, and universal quantification.

We make three major contributions in this dissertation:

1. We formalize the SMT problem over the reals with general Lipschitz-continuous ODEs. To showcase its expressiveness, we illustrate how various standard ODE-related problems can be encoded, including initial and boundary value problems, parameter synthesis problems, differential algebraic equations, and bounded model checking of hybrid systems. Our proposed algorithms utilize ODE constraints to construct pruning operators within a branch-and-prune framework, and we demonstrate their delta-completeness through mathematical proof. Finally, the scalability of these algorithms is demonstrated on practical benchmarks, demonstrating their ability to tackle formal verification problems for a range of nonlinear hybrid systems.

2. We propose delta-complete decision procedures for solving the satisfiability of nonlinear SMT problems over real numbers that contain universal quantification and a wide range of nonlinear functions. Our approach combines interval constraint propagation, counterexample-guided synthesis, and numerical optimization techniques. We demonstrate how to effectively handle the interleaving of numerical and symbolic computation to ensure delta-completeness in quantified reasoning. We demonstrate that the proposed algorithms can handle various challenging global optimization and control synthesis problems that are beyond the reach of existing solvers.

3. We present the design and implementation of the delta-decision procedure, dReal, and the delta-reachability analysis tool, dReach. dReal is capable of handling various nonlinear real functions, such as polynomials, trigonometric functions, and exponential functions, and implements the delta-complete decision procedure framework. dReach, on the other hand, encodes reachability problems as first-order real formulas and solves them using dReal. dReach is equipped to handle a wide range of highly nonlinear hybrid systems, as demonstrated by its scalability on various realistic models from biomedical and robotics applications.

Our work, while building on the foundation of Gao, Avigad, and Clarke's seminal papers [48, 49], sets itself apart in several key aspects.

- In [49], Gao, Avigad, and Clarke introduced the notion of delta-decidability and offered complexity results for general problems, including ODE constraints and bounded quantifiers. Although this work serves as a theoretical basis, it does not delve into the specifics of algorithms. In contrast, this thesis presents not only the algorithms but also the design and implementation of tools based on them.

- In [48], Gao, Avigad, and Clarke established the existence and complexity of delta-complete decision procedures for bounded quantifier-free SMT problems. This paper does not address the quantified problems covered in this thesis. In comparison, this thesis provides an algorithm for exists-forall problems, demonstrates its well-definedness, and implements it in dReal.

- While [48] describes the general DPLL<ICP> framework and suggests the possibility of defining ODE pruning algorithms by providing an example (Proposition 4.3 Simple ODE-pruning), this thesis delves deeper into the various ODE pruning algorithms, proving their well-definedness and providing implementation details.

1.3 Thesis Organization

The remainder of this dissertation is organized as follows.

- In Chapter 2, we present various fundamental definitions and notation used throughout this thesis.
- In Chapter 3, we study SMT problems over the reals containing ordinary differential equations and present delta-complete algorithms for the SMT problems. The work presented in this chapter is a reformulation and extension of a prior publication [54].
- In Chapter 4, we present a delta-complete decision procedure for solving satisfiability of nonlinear SMT problems over real numbers that contain universal quantification and a wide range of nonlinear functions. The work presented in this chapter is a reformulation and extension of a prior publication [80].
- In Chapter 5, we present the design and implementation of dReal. The work presented in this chapter is a reformulation and extension of a prior publication [52].
- In Chapter 6, we present the design and implementation of dReach. The work presented in this chapter is a reformulation and extension of prior publications [78, 86, 87].
- In Chapter 7, we conclude with a summary of contributions and future work.

Chapter 2

Preliminaries

In this chapter, we give various basic definitions and notions in the delta-decision framework [48, 49].

2.1 Computable Real Functions

As studied in Computable Analysis [116, 77], we can encode real numbers as infinite strings, and develop a computability theory of real functions using Turing machines that perform operations using oracles encoding real numbers. We use $||\cdot||$ to denote the max norm $||\cdot||_\infty$ over $\mathbb{R}^n$ for various n. First, a *name* of a real number is a sequence of rational numbers converging to it:

Definition 1 (Names)**.** *A name of $a \in \mathbb{R}$ is any function $\gamma_a : \mathbb{N} \to \mathbb{Q}$ that satisfies: for any $i \in \mathbb{N}$, $|\gamma_a(i) - a| < 2^{-i}$. For $\vec{a} \in \mathbb{R}^n$, $\gamma_{\vec{a}}(i) = \langle \gamma_{a_1}(i), ..., \gamma_{a_n}(i) \rangle$. We write the set of all possible names for $\vec{a}$ as $\Gamma(\vec{a})$.*

Next, a real function f is *computable* if there is a Turing machine that can use any argument x of f as an oracle, and compute the value of $f(x)$ up to an arbitrary precision 2^{-i}, where $i \in \mathbb{N}$. Formally:

Definition 2 (Computable Functions)**.** *We say $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ is computable if there exists an oracle Turing machine $\mathcal{M}_f$ such that for any $\vec{x} \in dom(f)$, any name $\gamma_{\vec{x}}$ of $\vec{x}$, and any $i \in \mathbb{N}$, the machine uses $\gamma_{\vec{x}}$ as an oracle and i as an input to compute a rational number $M_f^{\gamma_{\vec{x}}}(i)$ satisfying $|M_f^{\gamma_{\vec{x}}}(i) - f(\vec{x})| < 2^{-i}$.*

The definition requires that for any $\vec{x} \in \mathrm{dom}(f)$, with access to an arbitrary oracle encoding the name $\gamma_{\vec{x}}$ of $\vec{x}$, M_f outputs a 2^{-i}-approximation of $f(\vec{x})$. In other words, the sequence $M_f^{\gamma_{\vec{x}}}(1), M_f^{\gamma_{\vec{x}}}(2), ...$ is a name of $f(\vec{x})$. Intuitively, f is computable if an arbitrarily good approximation of $f(\vec{x})$ can be obtained using any good enough approximation to any $\vec{x} \in \mathrm{dom}(f)$. Most common continuous real functions are computable [116] including addition, multiplication, absolute value, min, max, exp, and sin. Compositions of computable functions are computable. In particular, solution functions of Lipschitz-continuous ordinary differential equations are computable.

2.2 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ Language

We consider first-order formulas over real numbers that can contain arbitrary nonlinear functions that can be numerically approximated, such as polynomials, exponential, and trigonometric functions. Theoretically, such functions are called *Type-2 computable* functions [116]. We write this language as $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$, formally defined as:

Definition 3 (The $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ Language)**.** *Let $\mathcal{F}$ be the set of Type-2 computable functions. We define $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ to be the following first-order language:*

$$t := x \mid f(\vec{t}), \text{ where } f \in \mathcal{F}, \text{possibly 0-ary (constant)};$$
$$\varphi := t(\vec{x}) > 0 \mid t(\vec{x}) \geq 0 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x_i \varphi \mid \forall x_i \varphi.$$

Remark 1. *Negations are not needed as part of the base syntax, as they can be defined through arithmetic: $\neg(t > 0)$ is simply $-t \geq 0$. Similarly, an equality $t = 0$ is just*

$t \geq 0 \wedge -t \geq 0$. *In this way we can put the formulas in normal forms that are easy to manipulate.*

We focus on formulas whose variables take values from bounded domains, which can be defined using bounded quantifiers:

Definition 4 (Bounded Quantifiers)**.** *The bounded quantifiers* $\exists^{[u,v]}$ *and* $\forall^{[u,v]}$ *are defined as*

$$\exists^{[u,v]} x.\varphi =_{df} \exists x.(u \leq x \wedge x \leq v \wedge \varphi),$$
$$\forall^{[u,v]} x.\varphi =_{df} \forall x.((u \leq x \wedge x \leq v) \rightarrow \varphi),$$

where u *and* v *denote* $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ *terms, whose variables only contain free variables in* φ *excluding* x*. It is easy to check that* $\exists^{[u,v]} x.\varphi \leftrightarrow \neg\forall^{[u,v]} x.\neg\varphi$*.*

2.3 Delta-decidability

The key definition in the framework is δ*-variants* of first-order formulas:

Definition 5 (δ-Variants)**.** *Let* $\delta \in \mathbb{Q}^+ \cup \{0\}$*, and* φ *a bounded* $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$*-sentence of the standard form*

$$\varphi : \; Q_1^{I_1} x_1 \cdots Q_n^{I_n} x_n \; \psi[t_i(\vec{x}) > 0; t_j(\vec{x}) \geq 0],$$

where $i \in \{1, ...k\}$ *and* $j \in \{k+1, ..., m\}$*. Note that negations are represented by sign changes on the terms. The* δ-weakening $\varphi^{-\delta}$ *of* φ *is defined as the result of replacing each atom* $t_i > 0$ *by* $t_i > -\delta$ *and* $t_j \geq 0$ *by* $t_j \geq -\delta$*. That is,*

$$\varphi^{-\delta} : \; Q_1^{I_1} x_1 \cdots Q_n^{I_n} x_n \; \psi[t_i(\vec{x}) > -\delta; t_j(\vec{x}) \geq -\delta].$$

The SMT problem is standardly defined as deciding satisfiability of quantifier-free formulas, which is equivalent to deciding the truth value of fully existentially quantified sentences. We will also consider formulas that are partially universally quantified. Thus, we consider both Σ_1 and Σ_2 formulas here.

Definition 6 (Bounded Σ_1- and Σ_2-SMT Problems)**.** *A Σ_1-SMT problem is a formula of the form*

$$\exists^{I_1} x_1 \cdots \exists^{I_n} x_n . \varphi(\vec{x})$$

and a Σ_2-SMT problem is of the form

$$\exists^{I_1} x_1 \cdots \exists^{I_n} x_n \forall^{I_{n+1}} x_{n+1} \cdots \forall^{I_m} x_m . \varphi(\vec{x}).$$

In both cases $\varphi(\vec{x})$ is a quantifier-free $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula.

Definition 7 (δ-Completeness [48])**.** *Let S be a set of $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ formulas, and $\delta \in \mathbb{Q}^+$. We say a decision procedure A is δ-complete for S, if for any $\varphi \in S$, A correctly returns one of the following answers*

- *φ is false;*
- *$\varphi^{-\delta}$ is true.*

If the two cases overlap, either one is correct.

In [49], it is proved that δ-complete decision procedures exists for arbitrary bounded $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-sentences. In particular, there exists δ-complete decision procedures for the bounded Σ_1 and Σ_2 SMT problems. This serves as the theoretical foundation as well as a correctness requirement for the practical algorithms that we will develop in this thesis.

Theorem 1 (Complexity [49])**.** *Let S be a class of $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-sentences, such that for any φ in S, the terms in φ are in Type 2 complexity class* C*. Then, for any $\delta \in \mathbb{Q}^+$, the δ-decision problem for bounded Σ_n-sentences in S is in $(\Sigma_n^P)^C$.*

The theorem states that, as a general rule, an increase in the number of quantifier alternations will correspond to an increase in the complexity of the problem, unless P = NP (recall that Σ_0^P = P and Σ_1^P = NP). This result can specialized for specific families of functions. For example, with polynomially-computable functions, the δ-decision problem for bounded Σ_n-sentences is (Σ_n^P)-complete. For a comprehensive analysis and related results, the reader is referred to [49].

Chapter 3

Satisfiability Modulo Ordinary Differential Equations

3.1 Introduction

Hybrid systems tightly combine finite automata and continuous dynamics. In most cases, the continuous components are specified by ordinary differential equations (ODEs). Thus, formal verification of general hybrid systems requires reasoning about logic formulas over the reals that contain ODE constraints. This problem is considered very difficult and has not been investigated in the context of decision procedures until recently [40, 41, 68]. It is believed that current techniques are not powerful enough to handle formulas that arise from formal verification of realistic hybrid systems, which typically contain many nonlinear ODEs and other constraints.

Since the first-order theory over the reals with trigonometric functions is already undecidable, solving formulas with general ODEs seems inherently impossible. This theoretical difficulty was resolved by the study of *δ-complete* decision procedures for such formulas [49]. An algorithm is δ-complete for a set of SMT formulas, where δ is an arbitrary positive rational number, if it correctly decides whether a

formula is unsatisfiable or δ-satisfiable. Here, a formula is δ-satisfiable if, under some *δ-perturbations*, a syntactic variant of the original formula is satisfiable [48]. It has shown that δ-complete decision procedures are suitable for various formal verification tasks [48, 49]. It was also proved that δ-complete decision procedures exist for SMT problems over the reals with Lipschitz-continuous ODEs. Such results serve as a theoretical foundation for developing practical decision procedures for the SMT problem.

In this chapter we study practical δ-complete algorithms for SMT formulas over the reals with ODEs. We show that such algorithms can be made powerful enough to scale to realistic benchmark formulas with several hundred nonlinear ODEs.

We develop decision procedures for the problem following a standard DPLL$\langle$ICP$\rangle$ framework, which relies on constraint solving algorithms as studied in Interval Constraint Propagation (ICP) [13]. In this framework, for any ODE system we can consider its solution function $\vec{x}_t = \vec{f}(t, \vec{x}_0)$ as a constraint between the initial variables $\vec{x}_0$, time variable t, and the final state variables $\vec{x}_t$. We define pruning operators that take interval assignments on $\vec{x}_0$, t, and $\vec{x}_t$ as inputs, and output refined interval assignments on these variables. We formally prove that the proposed algorithms are δ-complete. Beyond standard SMT problems where all variables are existentially quantified, we also study $\exists\forall$-formulas under the restriction that the universal quantifications are limited to the time variables (we call them $\exists\forall^t$-formulas). Such formulas have been an obstacle in SMT-based verification of hybrid systems [29, 30].

In brief, we make the following contributions:

- We formalize the SMT problem over the reals with general Lipschitz-continuous ODEs, and illustrate its expressiveness by encoding various standard problems concerning ODEs: initial and boundary value problems, parameter synthesis problems, differential algebraic equations, and bounded model checking of hybrid systems. In some cases, $\exists\forall^t$-formulas are needed.

- We propose algorithms for solving SMT with ODEs, using ODE constraints to

design pruning operators in a branch-and-prune framework. We handle both standard SMT problems with only existentially quantified variables, as well as $\exists\forall^t$-formulas. We prove that the algorithms are δ-complete.

- We demonstrate the scalability of the algorithms, as implemented in our open-source solver dReal [52], on realistic benchmarks encoding formal verification problems for several nonlinear hybrid systems.

This chapter is organized as follows. In Section 3.2, we define the SMT problem with ODEs and show how it can encode various standard problems with ODEs. In Section 3.3, we propose algorithms in the DPLL$\langle$ICP$\rangle$ framework for solving fully existentially quantified formulas as well as $\exists\forall^t$ formulas. In Section 3.4 we show experimental results.

Related Work Solving real constraints with ODEs has a wide range of applications, and much previous work exists for classes with special structures in different paradigms [32, 60, 85]. [59] proposed a more general constraint solving framework, focusing on the formulation of the problem in the standard constraint programming framework. On the SMT solving side, several authors have considered logical combinations of ODE constraints and proposed partial decision procedures [40, 41, 68]. We aim to extend and formalize existing algorithms for a general SMT theory with ODEs, and formally prove that they can be made δ-complete. In terms of practical performance, the proposed algorithms are made scalable to various benchmarks that contain hundreds of nonlinear ODEs and variables.

3.2 SMT over the Reals with ODEs

3.2.1 The ICP framework

The method of Interval Constraint Propagation (ICP) [13] finds solutions of real constraints using a "branch-and-prune" method that performs constraint propagation of interval assignments on real variables. The intervals are represented by floating-point end-points. Only over-approximations of the function values are used, which are defined by interval extensions of real functions.

Definition 8 (Floating-Point Intervals and Hulls)**.** *Let $\mathbb{F}$ denote the finite set of all floating point numbers with symbols $-\infty$ and $+\infty$ under the conventional order $<$. Let*

$$\mathbb{IF} = \{[a,b] \subseteq \mathbb{R} : a,b \in \mathbb{F}, a \leq b\} \text{ and } \mathbb{BF} = \bigcup_{n=1}^{\infty} \mathbb{IF}^n$$

denote the set of closed real intervals with floating-point endpoints, and the set of boxes *with these intervals, respectively. When $S \subseteq \mathbb{R}^n$ is a set of real numbers, the hull of S is:*

$$\text{Hull}(S) = \bigcap \{B \in \mathbb{BF} : S \subseteq B\}.$$

For $I = [a,b] \in \mathbb{IF}$, we write $|I| = |b-a|$ to denote its width. For $B \in \mathbb{IF}^n$, $||B||$ denotes its maximum width.

Definition 9 (Interval Extension [13])**.** *Suppose $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ is a real function. An interval extension operator $\sharp(\cdot)$ maps f to a function $\sharp f :\subseteq \mathbb{BF} \to \mathbb{IF}$, such that*

for any $B \in dom(\sharp f)$, it is always true that $\{f(\vec{x}) : \vec{x} \in B\} \subseteq \sharp f(B)$.

ICP uses interval extensions of functions to "prune" out sets of points that are not in the solution set, and "branch" on intervals when such pruning can not be done, until a small enough box that may contain a solution is found. A high-level description of the decision version of ICP is given in Algorithm 3.1.

In Algorithm 3.1, Branch(B,i) is an operator that returns two smaller boxes $B' = I_1 \times \cdots \times I_i' \times \cdots \times I_n$ and $B'' = I_1 \times \cdots \times I_i'' \times \cdots \times I_n$, where $I_i \subseteq I_i' \cup I_i''$. The key component of the algorithm is the Prune(B,f) operation. Any operation that contracts the intervals on variables can be seen as pruning, but for correctness we

Algorithm 3.1 ICP($f_1, \ldots, f_m, B_0 = I_1^0 \times \cdots \times I_n^0, \delta$)

```
1:  S ← B_0
2:  while S ≠ ∅ do
3:      B ← S.pop()
4:      while ∃1 ≤ i ≤ m, B ≠ Prune(B, f_i) do
5:          B ← Prune(B, f_i)
6:      end while
7:      if B ≠ ∅ then
8:          if ∃1 ≤ i ≤ n, |♯f_i(B)| ≥ δ then
9:              {B_1, B_2} ← Branch(B, i)
10:             S.push({B_1, B_2})
11:         else
12:             return sat
13:         end if
14:     end if
15: end while
16: return unsat
```

need formal requirements on the pruning operator in ICP. Basically, we need to require that the interval extensions of the functions converge to the true values of the functions, and that the pruning operations are well-defined, as specified below.

Definition 10 (δ-Regular Interval Extensions)**.** *We say an interval extension $\sharp f$ of $f : \mathbb{R}^n \to \mathbb{R}$ is δ-regular, if for some constant $c \in \mathbb{R}$, for any $B \in \mathbb{IF}^n$, $|\sharp f(B)| \leq \max(c||B||, \delta)$.*

Definition 11 (Well-defined Pruning Operators [48])**.** *Let $\mathcal{F}$ be a collection of real functions, and $\sharp$ be a δ-regular interval extension operator on $\mathcal{F}$. A* well-defined (equality) pruning operator *with respect to $\sharp$ is a partial function* $\text{Prune}_\sharp :\subseteq \mathbb{BF} \times \mathcal{F} \to \mathbb{BF}$*, such that for any $f \in \mathcal{F}$, $B, B' \in \mathbb{BF}$,*

1. $\text{Prune}_\sharp(B, f) \subseteq B$;
2. *If* $\text{Prune}_\sharp(B, f) \neq \emptyset$, *then* $0 \in \sharp f(\text{Prune}_\sharp(B, f))$;
3. $B \cap \{\vec{a} \in \mathbb{R}^n : f(\vec{a}) = 0\} \subseteq \text{Prune}_\sharp(B, f)$.

When $\sharp$ is clear, we simply write Prune. The rules can be explained as follows. (W1) ensures that the algorithm always makes progress. (W2) ensures that the result of a pruning is always a reasonable box that may contain a zero, and otherwise B is pruned out. (W3) ensures that the real solutions are never discarded. Gao, Avigad, and Clarke proved the following theorem in [48]:

Theorem 2. *Algorithm 3.1 is δ-complete if the pruning operators are well-defined.*

3.2.2 Solution Functions of ODEs

We show that the framework of computable functions allows us to consider solution functions of ODE systems.

Notation 1. *We use $\vec{x} = \vec{y}$ between n-dimensional vectors to denote the system of equations $x_i = y_i$ for $1 \leq i \leq n$.*

Let $D \subseteq \mathbb{R}^n$ be compact and $g_i : D \to \mathbb{R}$ be n Lipschitz-continuous functions, which means that for some constant $c_i \in \mathbb{R}^+$ $(1 \leq i \leq n)$, for all $\vec{x}_1, \vec{x}_2 \in D$,

$$|g_i(\vec{x}_1) - g_i(\vec{x}_2)| \leq c_i ||\vec{x}_1 - \vec{x}_2||.$$

Let t be a variable over $\mathbb{R}$. We consider the first-order autonomous ODE system (3.1)

$$\frac{d\vec{y}}{dt} = \vec{g}(\vec{y}(t, \vec{x}_0)) \tag{3.1a}$$

$$\vec{y}(0, \vec{x}_0) = \vec{x}_0 \tag{3.1b}$$

where $\vec{x}_0 \in D$. Here, each

$$y_i : \mathbb{R} \times D \to \mathbb{R} \tag{3.2}$$

is called the i-th solution function of the ODE system (3.1). A key result in computable analysis is that these solution functions are computable, in the sense of Definition 2:

Proposition 1 ([77]). *The solution functions $\vec{y}$ in the form of* (3.2) *of the ODE system* (3.1) *are computable over $\mathbb{R} \times D$.*

To see why this is true, recall that for any $t \in \mathbb{R}$ and $\vec{x}_0 \in D$, the value of the solution function follows the Picard-Lindelöf form:

$$\vec{y}(t, \vec{x}_0) = \int_0^t \vec{g}(\vec{y}(s, \vec{x}_0))\, ds + \vec{x}_0.$$

Approximations of the right-hand side of the equation can be computed by finite sums, theoretically up to an arbitrary precision.

3.2.3 SMT Encoding of Standard Problems with ODEs

In this section, we list several standard problems related to ODE systems and show that they can be easily encoded and generalized through SMT formulas. They motivate the development of decision procedures for the theory.

Remark 2. *In all the following cases, solutions to the standard problems are obtained from witnesses for the existentially quantified variables in the SMT formulas.*

Remark 3. *In the definitions below, when the solution functions $\vec{y}$ of ODE systems are written as part of a formula, no analytic forms are needed. They are functions included in the signature $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$.*

Generalized Initial Value Problems An initial value problem (IVP) in ordinary differential equations is a problem that involves specifying the values of the variables and their derivatives at a specific point in time (the initial time), and then solving the ODEs to find the values of the variables at all other times. The solution to an IVP is a function that describes how the variables change over time, subject to the constraints imposed by the ODEs and the initial conditions. IVPs are used to model a wide range of physical, biological, and engineering systems. In the form of SMT formulas, we easily allow the initial conditions to be constrained by arbitrary quantifier-free $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas:

Definition 12 (Generalized IVP)**.** *Let $X \subseteq \mathbb{R}^n$ be a compact domain, $T \in \mathbb{R}^+$, and $\vec{y} : [0, T] \times X \to X$ be the computable solution functions of an ODE system. Let $t \in [0, T]$ be an arbitrary constant that represents a time point of interest. The generalized IVP problem is defined by formulas of the form:*

$$\exists^X x_0 \exists^X \vec{x}.\ \varphi(\vec{x}_0) \wedge \vec{x} = \vec{y}(t, \vec{x}_0),$$

where φ is a quantifier-free $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula constraining the initial states $\vec{x}_0$, and $\vec{x}$ is the needed value for time point t.

Generalized Boundary Value Problems A boundary value problem (BVP) in ordinary differential equations is a type of problem where the solution is required to satisfy certain conditions, called boundary conditions, at two or more specified points in the domain of the problem, rather than a single initial point as in an initial value problem. The solution is a function that describes how the variables change over the entire domain of the problem, subject to the constraints imposed by the ODEs and the boundary conditions. A generalized version as encoded by SMT formulas is:

Definition 13 (Generalized BVP)**.** *Let $X \subseteq \mathbb{R}^n$ be a compact domain, $T \in \mathbb{R}^+$, and $\vec{y} : [0, T] \times X \to X$ be the solution functions of an ODE system. Let $t, t' \in [0, T]$ be two time points of interest. The generalized BVP problem is:*

$$\exists^X x_0 \exists^X \vec{x}_t \exists^X \vec{x}.\varphi(\vec{x}_0, \vec{x}_t, t) \wedge \vec{x}_t = \vec{y}(t, \vec{x}_0) \wedge \vec{x} = \vec{y}(t', \vec{x}_0)$$

where φ is a quantifier-free $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula that specifies the boundary conditions. Note that $\vec{x}$ is the value that we are interested in solving in the chosen time point t'.

Data-Fitting and Parameter Synthesis The data-fitting problem, also known as the inverse problem, is the task of finding the unspecified parameters in a system of ordinary differential equations that best fit a given set of data. The goal is to determine the parameters that result in the best match between the predictions of

the ODE model and the observed data. This problem arises in a wide range of applications, including physics, biology, and engineering, where the underlying dynamics of a system are not fully understood or are difficult to measure directly.

We define the data fitting problem as follows. Suppose an ODE system has part of its parameters unspecified. Given a sequence of data $(t_1, \vec{a}_1), \ldots, (t_k, \vec{a}_k)$, we need to find the values of the missing parameters of the original ODE system. More formally:

Definition 14 (Data-Fitting Problems)**.** *Let $X \subseteq \mathbb{R}^n$ and $P \subseteq \mathbb{R}^m$ be compact domains, $T \in \mathbb{R}^+$, and $\vec{y}(\vec{p}) : [0, T] \times X \to X$ be the solution functions of an ODE system, where $\vec{p} \in P$ be a vector of parameters. Let $(t_1, \vec{a}_1), \ldots, (t_k, \vec{a}_k)$ be a sequence of pairs in $[0, T] \times X$. The data-fitting problem is defined by:*

$$\exists^P \vec{p} \exists^X x_0.\ \varphi(\vec{x}_0) \wedge \vec{a}_1 = \vec{y}(\vec{p}, t_1, \vec{x}_0) \wedge \cdots \wedge \vec{a}_k = \vec{y}(\vec{p}, t_k, \vec{x}_0),$$

where a quantifier-free φ constraints the initial states $\vec{x}_0$.

Differential Algebraic Equations Differential algebraic equations (DAEs) are a type of mathematical equation that combines both differential and algebraic equations. They are used to describe a wide range of physical and engineering systems that involve both continuous and discrete variables. Formally, a DAE can be written in the following form:

$$\frac{d\vec{y}}{dt} = \vec{g}(\vec{y}(t, \vec{y}_0), \vec{z}) \tag{3.3}$$

$$0 = \vec{h}(\vec{y}, \vec{z}, t) \tag{3.4}$$

where $\vec{y}, \vec{y}_0 \in \mathbb{R}^n$, $\vec{z} \in \mathbb{R}^m$. To express the problem in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$, we need to use extra universal quantification to ensure that the algebraic relations hold throughout the time duration. Again, we can also generalize Equation (3.4) to an arbitrary quantifier-free $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula. The problem is encoded as:

Definition 15 (DAE Problems)**.** *Let $X \subseteq \mathbb{R}^n$ be a compact domain, $T \in \mathbb{R}^+$, and $\vec{y}_{\vec{z}} : [0, T] \times X \to X$ be the computable solution functions of the ODE system $\frac{d\vec{y}}{dt} = \vec{g}(\vec{y}(t, \vec{y}_0), \vec{z})$*

in (3.3) *parameterized by* $\vec{z}$*. Let* h *be defined by* (3.4)*. Let* $t \in [0, T]$ *be a time point of interest. A DAE problem is defined by the following formula:*

$$\exists^X \vec{x}_0 \exists^X \vec{x} \exists^Z \vec{z} \forall^{[0,t]} t'.\ \varphi(\vec{x}_0) \wedge \vec{x} = \vec{y}_{\vec{z}}(t, \vec{x}_0) \wedge h(\vec{y}_{\vec{z}}(\vec{x}_0, t'), \vec{z}, t') = 0$$

where a quantifier-free φ *specifies the initial conditions for* $\vec{y}$*, and* $\vec{x}$ *is the needed value at time point* t*.*

Bounded Model Checking of Hybrid Systems Bounded model checking problems for hybrid systems can be naturally encoded as SMT formulas with ODEs [40, 41, 68, 29, 30]. We consider a simple hybrid system to show an example. Let H be an n-dimensional 2-mode hybrid system. In mode 1, the flow of the system follows an ODE system whose solution function is $\vec{y}_1(t, \vec{x}_0)$, and in mode 2, it follows another solution function $\vec{y}_2(t, \vec{x}_0)$. The jump condition from mode 1 to mode 2 is specified by $\mathsf{jump}(\vec{x}, \vec{x}')$. The invariants are specified by $\mathsf{inv}_i(\vec{x})$ for mode i. Let $\mathsf{unsafe}(\vec{x})$ denote an unsafe region. Let the continuous variables be bounded in X and time be bounded in $[0, T]$. Now, if H starts from mode 1 with initial states satisfying $\mathsf{init}(\vec{x})$, it can reach the unsafe region after one discrete jump from mode 1 to mode 2, iff the following formula is true:

$$\begin{aligned}\exists^X \vec{x}_1 \exists^X \vec{x}_1^t\ \exists^X \vec{x}_2 \exists^X \vec{x}_2^t\ \exists^{[0,T]} t_1 \exists^{[0,T]} t_2\ \forall^{[0,t_1]} t_1' \forall^{[0,t_2]} t_2'.\ \Big(&\mathsf{init}(\vec{x}_1) \wedge \vec{x}_1^t = \vec{y}_1(t_1, \vec{x}_1) \wedge \mathsf{inv}_1(\vec{y}_1(t_1', \vec{x}_1)) \\ &\wedge \mathsf{jump}(\vec{x}_1^t, \vec{x}_2) \wedge \vec{x}_2^t = \vec{y}_2(t_2, \vec{x}_2) \\ &\wedge \mathsf{inv}_2(\vec{y}_2(t_2', \vec{x}_2)) \wedge \mathsf{unsafe}(\vec{x}_2^t)\Big).\end{aligned}$$

The encoding can be explained as follows. For each mode, we use two variable vectors $\vec{x}_i$ and $\vec{x}_i^t$ to represent the continuous flows. $\vec{x}_i$ denotes the starting values of a flow, and $\vec{x}_i^t$ denotes the final values. In mode 1, the flow starts with some values in the initial states, specified by $\mathsf{init}(\vec{x}_1)$. Then, we follow the continuous dynamics in mode 1, so that $\vec{x}_1^t$ denotes the final value $\vec{x}_1^t = \vec{y}(t_1, \vec{x}_1)$. Then the system follows the jumping condition and resets the variables from $\vec{x}_1^t$ to $\vec{x}_2$ as specified by $\mathsf{jump}(\vec{x}_1^t, \vec{x}_2)$.

After that, the system follows the flow in mode 2. In the end, we check if the final state $\vec{x}_2^t$ in mode 2 satisfies the unsafe predicate, $\mathsf{unsafe}(\vec{x}_2)$.

3.3 Algorithms

3.3.1 ODE Pruning in an ICP Framework

We now study the algorithms for SMT formulas with ODEs. The key is to design the appropriate pruning operators for the solution functions of ODE systems. The pruning operations here strengthen and formalize the ones proposed in [40, 41, 59], such that δ-completeness can be proved.

We recall some notations first. Let $D \subseteq \mathbb{R}^n$ be compact and $g_i : D \to D$ be n Lipschitz-continuous functions. Given the first-order autonomous ODE system

$$\frac{d\vec{y}}{dt} = \vec{g}(\vec{y}(t, \vec{x}_0)) \text{ and } \vec{y}(0, \vec{x}_0) = \vec{x}_0 \tag{3.5}$$

where $\vec{x}_0 \in D$, we write

$$y_i : [0, T] \times D \to D_i$$

to represent the i-th solution function of the ODE system. The δ-regular interval extension of y_i is an interval function

$$\natural y_i : (\mathbb{IF} \cap [0, T]) \times (\mathbb{BF} \cap D) \to \mathbb{IF}$$

such that for a constant $c \in \mathbb{R}$, for any time domain $I_t \subseteq \mathbb{IF} \cap [0, T]$ and any box of initial values $B_{\vec{x}_0} \subseteq \mathbb{BF} \cap D$, we have

$$\{x_t \in \mathbb{R} : x_t = y_i(t, \vec{x}_0), \vec{x}_0 \in B_{\vec{x}_0}, t \in I_t\} \subseteq \natural y_i(I_t, B_{\vec{x}_0})$$

and

$$|\natural y_i(I_t, B_{\vec{x}_0})| \leq \max(c \cdot ||I_t \times B_{\vec{x}_0}||, \delta).$$

We will also need the notion of the *reverse* of the ODE system (3.5), as defined by

$$\frac{d\vec{y}_-}{dt} = \vec{g}_-(\vec{y}_-(t, \vec{x}_t)) \text{ and } \vec{y}(0, \vec{x}_t) = \vec{x}_t. \tag{3.6}$$

Here, $\vec{g}_-$ is defined as $-\vec{g}$, the vector of functions consisting of the negation of each function in $\vec{g}$, which is equivalent to reversing time in the flow defined by the ODE system. That is, for $\vec{x}_0, \vec{x}_t \in D, t \in \mathbb{R}$, we always have

$$\vec{x}_t = \vec{y}(t, \vec{x}_0) \text{ iff } \vec{x}_0 = \vec{y}_-(t, \vec{x}_t). \tag{3.7}$$

Naturally, we write $\sharp(y_-)_i$ to denote the δ-regular interval extension of the i-th component of $\vec{y}_-$.

Algorithm 3.2 ODEPruning$(\sharp\vec{y}, B_{\vec{x}_0}, B_{\vec{x}_t}, I_t)$

1: **repeat**
2: $\quad B'_{\vec{x}_t} \leftarrow \text{Prune}_{\text{fwd}}(\sharp\vec{y}, B_{\vec{x}_0}, B_{\vec{x}_t}, I_t)$
3: $\quad I'_t \leftarrow \text{Prune}_{\text{time}}(\sharp\vec{y}, B_{\vec{x}_0}, B'_{\vec{x}_t}, I_t)$
4: $\quad B'_{\vec{x}_0} \leftarrow \text{Prune}_{\text{bwd}}(\sharp\vec{y}, B_{\vec{x}_0}, B'_{\vec{x}_t}, I'_t)$
5: **until** $B_{\vec{x}_0} = B'_{\vec{x}_0} \wedge B_{\vec{x}_t} = B'_{\vec{x}_t} \wedge I_t = I'_t$
6: **return** $(B'_{\vec{x}_0}, B'_{\vec{x}_t}, I'_t)$

The relation between the initial variables $\vec{x}_0$, the time duration t, and the flow variables $\vec{x}_t$ is specified by the constraint $\vec{x}_t = \vec{y}(t, \vec{x}_0)$. Given the interval assignment on any two of $\vec{x}_0$, $\vec{x}_t$, and t, we can use the constraint to obtain a refined interval assignment to the third variable vector. Thus, we can define three pruning operators as follows.

Remark 4. *The precise definitions of the pruning operators should map the interval assignments on all variables to new assignments on all variables. For notational simplicity, in the pruning operators below we only list the assignments that are actually changed between inputs and outputs. For instance, the forward pruning operator only changes the values on* $B_{\vec{x}_t}$.

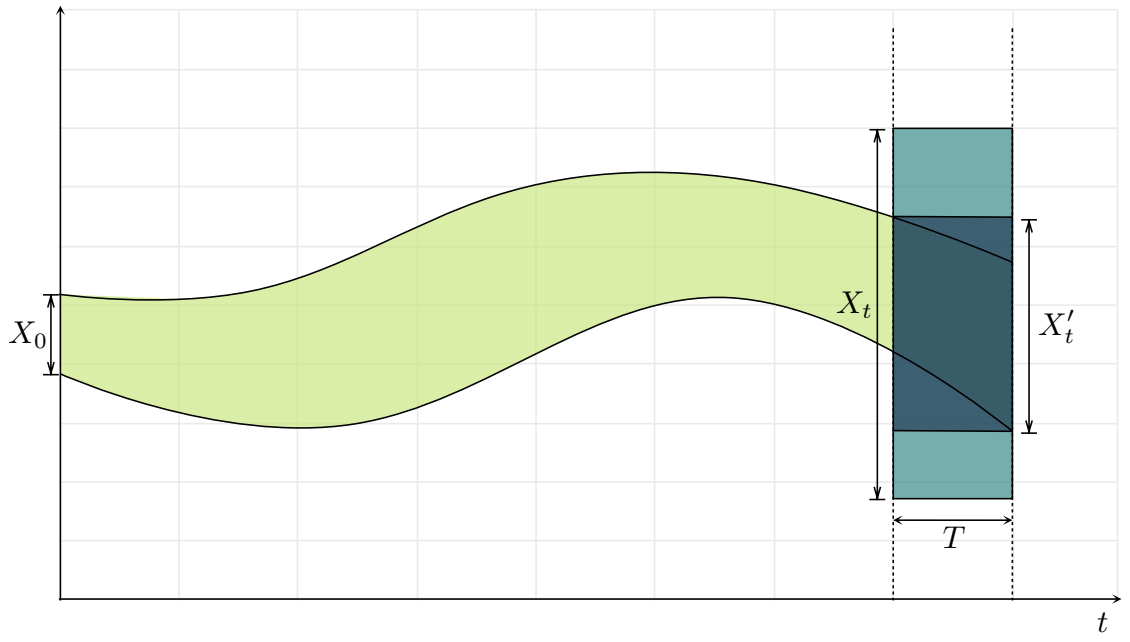


Figure 3.1: Illustration of the forward ODE pruning operator. X_0, X_t, T represent the current interval assignments on $\vec{x}_0$, $\vec{x}_t$, and t. $X'_t \subseteq X_t$ is the refined interval assignment on $\vec{x}_t$ after pruning.

Forward Pruning Given interval assignments on $\vec{x}_0$ and t, we compute a refinement of the interval assignments on $\vec{x}_t$. Figure 3.1 depicts the forward pruning operation. Formally, we define the following operator:

Definition 16 (Forward Pruning)**.** *Let $\vec{y} : [0, T] \times D \to D$ be the solution functions of an ODE system. Let $B_{\vec{x}_0}$, $B_{\vec{x}_t}$, and I_t be interval assignments on the variables $\vec{x}_0$, $\vec{x}_t$, and t. We define the forward-pruning operator as:*

$$\text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y}) = \text{Hull}\left(B_{\vec{x}_t} \cap \sharp\vec{y}(I_t, B_{\vec{x}_0})\right).$$

Backward Pruning Given interval assignments on $\vec{x}_t$ and t, we can compute a refinement of the interval assignments on $\vec{x}_0$ using the reverse of the solution function. Figure 3.2 depicts backward pruning. Formally, we define the following operator:

Algorithm 3.3 $\text{Prune}_{\text{fwd}}(\sharp\vec{y}, B_{\vec{x}_0}, B_{\vec{x}_t}, I_t)$

1: $B'_{\vec{x}_t} \leftarrow \phi$
2: $I_{\Delta t} \leftarrow [I_t^l, I_t^l + \varepsilon]$
3: **while** $I_{\Delta t}^u < I_t^u$ **do**
4: $\quad B'_{\vec{x}_t} \leftarrow \text{Hull}(B'_{\vec{x}_t} \cup \sharp\vec{y}(I_{\Delta t}, B_{\vec{x}_0}))$
5: $\quad I_{\Delta t} \leftarrow I_{\Delta t} + \varepsilon$
6: **end while**
7: **return** $B_{\vec{x}_t} \cap B'_{\vec{x}_t}$

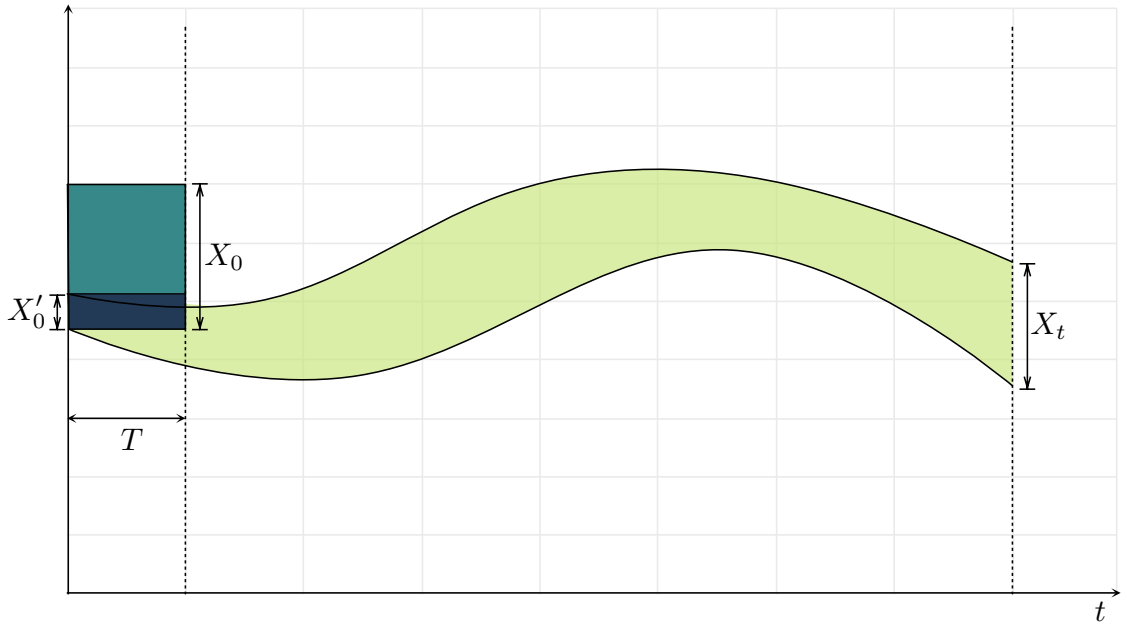


Figure 3.2: Illustration of the backward ODE pruning operator. X_0, X_t, T represents the current interval assignments on $\vec{x}_0$, $\vec{x}_t$, and t. $X'_0 \subseteq X_0$ is the refined interval assignment on $\vec{x}_0$ after pruning.

Definition 17 (Backward Pruning)**.** *Let $\vec{y} : [0, T] \times D \to D$ be the solution functions of an ODE system, and let $\vec{y}_-$ be the reverse of $\vec{y}$. Let $B_{\vec{x}_0}$, $B_{\vec{x}_t}$, and I_t be interval assignments on the variables $\vec{x}_0$, $\vec{x}_t$, and t. We define the backward-pruning operator as:*

$$\text{Prune}_{\text{bwd}}(B_{\vec{x}_0}, \vec{y}) = \text{Hull}\left(B_{\vec{x}_0} \cap \sharp\vec{y}_-(I_t, B_{\vec{x}_t})\right).$$

Algorithm 3.4 $\text{Prune}_{\text{bwd}}(\sharp\vec{y}, B_{\vec{x}_0}, B_{\vec{x}_t}, I_t)$

1: $B'_{\vec{x}_0} \leftarrow \phi$
2: $I_{\Delta t} \leftarrow [I_t^l, I_t^l + \varepsilon]$
3: **while** $I_{\Delta t}^u < I_t^u$ **do**
4: $\quad B'_{\vec{x}_0} \leftarrow \text{Hull}(B'_{\vec{x}_0} \cup \sharp\vec{y}_-(I_{\Delta t}, B_{\vec{x}_t}))$
5: $\quad I_{\Delta t} \leftarrow I_{\Delta t} + \varepsilon$
6: **end while**
7: **return** $B_{\vec{x}_0} \cap B'_{\vec{x}_0}$

Time-Domain Pruning Given interval assignments on $\vec{x}_0$ and $\vec{x}_t$, we can also refine the interval assignment on t by pruning out the time intervals that do not contain any $\vec{x}_t$ that is consistent with the current interval assignments on $\vec{x}_t$. Figure 3.3 depicts time-domain pruning. Formally, we define the following operator:

Definition 18 (Time-Domain Pruning)**.** *Let $\vec{y} : [0, T] \times D \to D$ be the solution functions of an ODE system. Let $B_{\vec{x}_0}$, $B_{\vec{x}_t}$, I_t be interval assignments on the variables $\vec{x}_0$, $\vec{x}_t$, and t. We define the time-domain pruning operator as:*

$$\text{Prune}_{\text{time}}(I_t, \vec{y}) = \text{Hull}\left(I_t \cap \{I : \sharp\vec{y}(I, B_{\vec{x}_0}) \cap B_{\vec{x}_t} \neq \emptyset\}\right).$$

Overall, the pruning algorithm on based on ODE constraints iteratively applies the three pruning operators until a fixed point on the interval assignments is reached.

We show the more detailed steps in the three pruning operations in Algorithms 3.2 to 3.5.

Theorem 3. *The three pruning operators are well-defined.*

Algorithm 3.5 $\text{Prune}_{\text{time}}(\natural\vec{y}, B_{\vec{x}_0}, B_{\vec{x}_t}, I_t)$

1: $I'_t \leftarrow \phi$
2: $I_{\Delta t} \leftarrow [I^l_t, I^l_t + \varepsilon]$
3: **while** $I^u_{\Delta t} < I^u_t$ **do**
4: $\quad B'_{\vec{x}_t} \leftarrow \natural\vec{y}(I_{\Delta t}, B_{\vec{x}_0})$
5: $\quad$ **if** $B'_{\vec{x}_t} \cap B_{\vec{x}_t} \neq \phi$ **then**
6: $\quad\quad I'_t = \text{Hull}(I'_t \cup I_{\Delta t})$
7: $\quad$ **else**
8: $\quad\quad I_{\Delta t} \leftarrow I_{\Delta t} + \varepsilon$
9: $\quad$ **end if**
10: **end while**
11: **return** I'_t

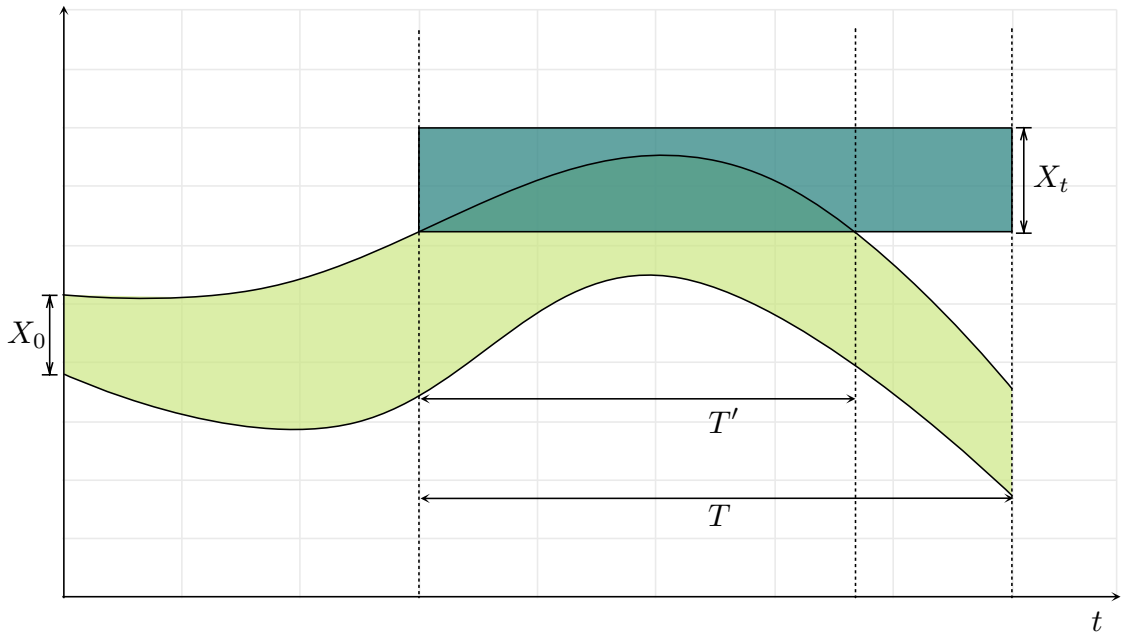


Figure 3.3: Illustration of the time-domain ODE pruning operator. X_0, X_t, T represents the current interval assignments on $\vec{x}_0$, $\vec{x}_t$, and t. $T' \subseteq T$ is the refined interval assignment on t after pruning.

Proof. We prove that the forward pruning operator is well-defined, and the proofs for the other two operators are similar. Note that the definitions of well-defined pruning are formulated for equality constraints compared to 0. Here we use the function $f = \vec{y}(t, \vec{x}_0) - \vec{x}_t$ in the pruning operator. (Strictly speaking f is a function vector that evaluates to $\vec{0}$ on points satisfying the ODE flow. Here for notational simplicity we just write f as a single-valued function and compare with the scalar 0.)

First, (W1) is satisfied because of the simple fact that for any boxes $B_1, B_2 \in \mathbb{BF}$, we have $\text{Hull}(B_1 \cap B_2) \subseteq B_1$.

Next, suppose $0 \notin \sharp f(\text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y}) - B_{\vec{x}_t})$. Then there does not exist any $\vec{a}_t \in \mathbb{R}^n$ that satisfies both $\vec{a}_t \in B_{\vec{x}_t}$ and $\vec{a}_t \in \text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y})$. Since at the same time

$$\text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y}) = \text{Hull}\left(B_{\vec{x}_t} \cap \sharp\vec{y}(I_t, B_{\vec{x}_0})\right) \subseteq B_{\vec{x}_t},$$

this requires that $\text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y}) = \emptyset$. Consequently (W2) is satisfied.

Third, note that $\sharp\vec{y}(I_t, B_{\vec{x}_0})$ is an interval extension of $\vec{y}$. Thus, for any $\vec{a}_t \in \mathbb{R}^n$ such that $\vec{y}(t, \vec{x}_0)$ for some $t \in I_t$ and $\vec{x}_0 \in B_{\vec{x}_0}$, we have $\vec{a}_t \in \sharp\vec{y}(I_t, B_{\vec{x}_0})$. Following the definition of the pruning operator, we have $\vec{a}_t \in \text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y})$. Thus, $B_{\vec{x}_t} \cap Z_f \subseteq \text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, f)$ and (W3) holds. □

3.3.2 $\exists\forall^t$-Formulas and Linear Approximation

For $\exists\forall$-formulas, if the universal quantification is only over the time variables, we can follow the trajectory and prune away the assignments on $\vec{x}_0$, $\vec{x}_t$, and t that violate the constraints on the universally quantified time variable. In fact, although the extra quantification complicates the problem, the universal constraints improve the power of the pruning operations.

Here we focus on problems with one ODE system, which can be easily generalized. Let $\vec{y}$ denote the solution functions of an ODE system, we consider an $\exists\forall^t$-formula of the form

$$\exists^X \vec{x}_0 \exists^X \vec{x}_t \exists^{[0,T]} t \forall^{[0,t]} t'.\ \vec{x}_t = \vec{y}(t, \vec{x}_0) \wedge \varphi(\vec{y}(t', \vec{x}_0)). \tag{3.8}$$

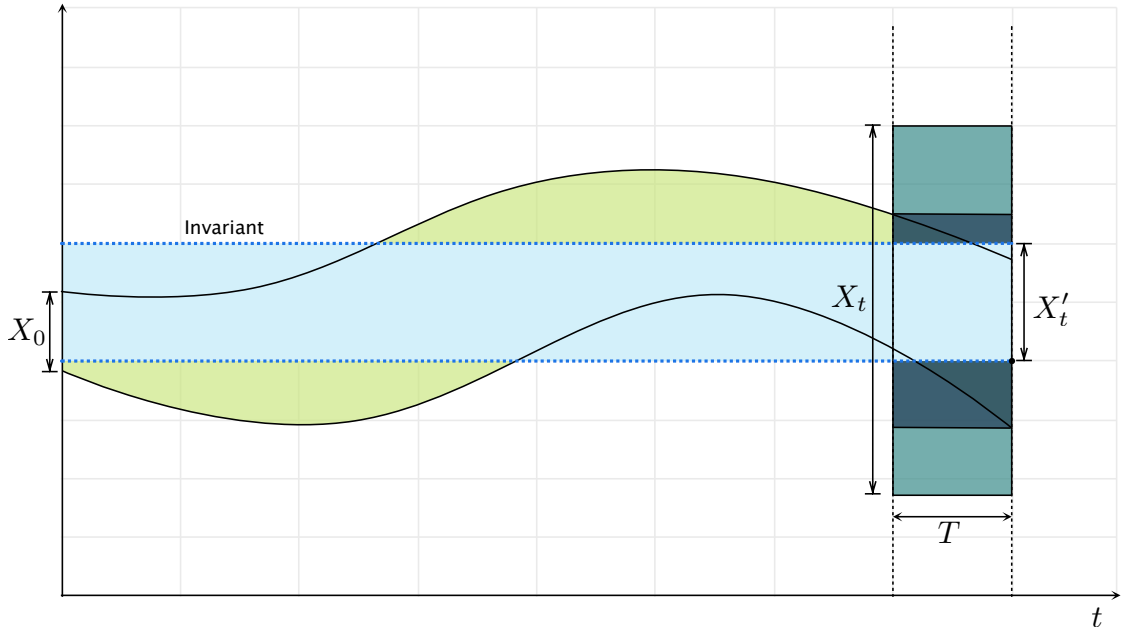


Figure 3.4: Illustration of the ODE pruning operator for $\forall^t$-constraints.

Note that the problems encoded as Σ_2-SMT formulas as listed in Section 3.2.3 are all of this form.

We consider $\varphi(\vec{y}(t', \vec{x}_0))$ as a special constraint on the $\vec{x}_0$ and t variables. Using this constraint, we can further refine the three pruning operators as follows.

Definition 19 (Pruning Refined by $\forall^t$-Constraints)**.** *Let $\vec{y} : [0, T] \times D \to \mathbb{R}^n$ be the solution functions of an ODE system. Let $B_{\vec{x}_0}$, $B_{\vec{x}_t}$, and I_t be interval assignments on the variables $\vec{x}_0$, $\vec{x}_t$, and t. Let $\varphi(\vec{y}(t', \vec{x}_0))$ be a constraint on the universally quantified time variable, as in* (3.8)*. We first define*

$$\sharp\varphi(I_t, B_{\vec{x}_0}) = \text{Hull}(\{\vec{a} \in \mathbb{R}^n : \vec{a} = \vec{y}(t, \vec{x}_0), t \in I_t, \vec{x}_0 \in B_{\vec{x}_0}, \text{ and } \varphi(\vec{a}) \text{ is true.}\})$$

and define $\sharp\varphi_-$ by replacing $\vec{y}$ with $\vec{y}_-$ in the definition above. The forward pruning operator with φ, written as $\text{Prune}^{\varphi}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y})$, is defined as

$$\text{Hull}\left(B_{\vec{x}_t} \cap \sharp\vec{y}(I_t, B_{\vec{x}_0}) \cap \sharp\varphi(I_t, B_{\vec{x}_0})\right).$$

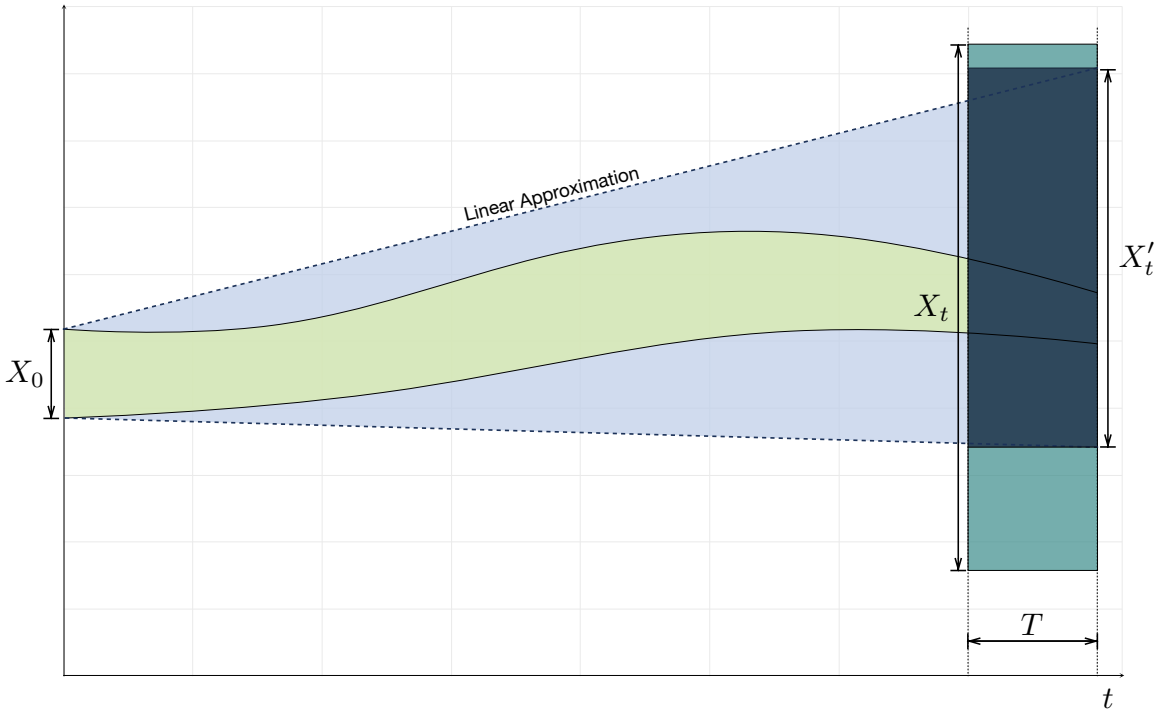


Figure 3.5: Illustration of the ODE pruning operator using linear approximation.

Backward pruning $\text{Prune}^{\varphi}_{\text{bwd}}(B_{\vec{x}_0}, \vec{y})$ *is defined as*

$$\text{Hull}\left(B_{\vec{x}_0} \cap \sharp\vec{y}_-(I_t, B_{\vec{x}_t}) \cap \sharp\varphi_-(I_t, B_{\vec{x}_t})\right).$$

Time-domain pruning $\text{Prune}^{\varphi}_{\text{time}}(I_t, \vec{y})$ *is defined as*

$$\text{Hull}\left(I_t \cap \{I : \sharp\vec{y}(I, B_{\vec{x}_0}) \cap B_{\vec{x}_t} \cap \sharp\varphi(I_t, B_{\vec{x}_0}) \neq \emptyset\}\right).$$

In general, $\sharp\varphi$ can be computed by a recursive call to DPLL⟨ICP⟩, by solving the Σ_1-SMT problem $\varphi(\vec{x})$. In many practical applications, φ is of some simple form such as $\vec{a} \leq \vec{x}_t \leq \vec{b}$, in which case simple pruning is shown in Figure 3.4.

Another useful heuristic in ODE pruning involves utilizing linear approximation to bound the range of the derivatives for a vector space specified by $\vec{g}$. Assuming that at any time $t \in [0, T]$, the derivatives $\vec{g}$ are constrained within $[\vec{l}_g, \vec{u}_g]$. We can then apply the Picard-Lindelöf representation, resulting in the following expression

for $\vec{x}_t$:

$$\vec{x}_t = \int_0^t \vec{g}(\vec{y}(s, \vec{y}_0))\, ds + \vec{y}_0 \in [0, T] \cdot [\vec{l}_g, \vec{u}_g] + B_{\vec{x}_0}..$$

This equation can be employed for preliminary pruning of $`vecx_t$, proving particularly efficient when combined with $\forall^t$-constraints. This pruning method is illustrated in Figure 3.5.

3.4 Experiments

Our tool dReal3 implements the procedures we studied for solving SMT formulas with ODEs. It is built on several existing packages, including opensmt [22] for the general DPLL⟨ICP⟩ framework, IBEX-lib [112] for Interval Constraint Propagation, and CAPD [25] for computing interval-enclosures of ODEs. The tool has been made available for public use through the GNU Public License (GPL) and can be accessed at `https://github.com/dreal/dreal3`. All benchmarks and data shown here are also available on the tool website.

All experiments were conducted on a machine with a 3.4 GHz octa-core Intel Core i7-2600 processor and 16 GB RAM, running 64-bit Ubuntu 18.04 LTS. Table 3.1 is a summary of the running time of the tool on various SMT formulas generated from bounded model checking hybrid systems. The formulas typically contain a large number of variables and nonlinear ODEs.

Atrial Fibrillation Model The AF model as we show in Table 3.1 is obtained from [61]. It is a precise model of atrial fibrillation, a serious cardiac disorder.

Atrial fibrillation is a type of heart rhythm disorder that affects the upper chambers of the heart, known as the atria. In this condition, the atria beat in a rapid and irregular manner, causing the heart to pump blood less efficiently. Atrial fibrillation is a common condition that affects millions of people worldwide and is most commonly seen in people over the age of 60. It is a serious condition that can lead to a range of complications, including stroke, heart failure, and even death.

Problem	#Mode	#Depth	#ODEs	#Vars	δ	Result	Time(s)	Trace
AF	4	3	20	44	0.001	δ-SAT	43.10	90K
AF	8	7	40	88	0.001	δ-SAT	698.86	20M
AF	8	23	120	246	0.001	δ-SAT	4528.13	59M
AF	8	31	160	352	0.001	δ-SAT	8485.99	78M
AF	8	47	240	528	0.001	δ-SAT	15740.41	117M
AF	8	55	280	616	0.001	δ-SAT	19989.59	137M
CT	2	2	15	36	0.005	δ-SAT	345.84	3.1M
CT	2	2	15	36	0.002	δ-SAT	362.84	3.1M
EO	3	2	18	42	0.01	δ-SAT	52.93	998K
EO	3	2	18	42	0.001	δ-SAT	57.67	847K
EO	3	11	72	168	0.01	UNSAT	7.75	–
BB	2	10	22	66	0.01	δ-SAT	0.25	123K
BB	2	20	42	126	0.01	δ-SAT	0.57	171K
BB	2	20	42	126	0.001	δ-SAT	2.21	168K
BB	2	40	82	246	0.01	UNSAT	0.27	—-
BB	2	40	82	246	0.001	UNSAT	0.26	—-
D1	3	2	9	24	0.1	δ-SAT	30.84	72K
DU	3	2	6	16	0.1	UNSAT	0.04	–

Table 3.1: #Mode = Number of modes in the hybrid system, #Depth = Unrolling depth, #ODEs = Number of ODEs in the unrolled formula, #Vars = Number of variables in the unrolled formula, Result = Bounded Model Checking Result (δ-SAT/UNSAT), Time = CPU time (s), Trace = Size of the ODE trajectory, AF = Atrial Filbrillation, CT = Cancer Treatment, EO = Electronic Oscillator, BB = Bouncing Ball with Drag, D1/DU = Decay Models.

This is because the rapid and irregular beating of the atria can cause blood clots to form, which can then travel to the brain and cause a stroke.

The continuous dynamics in the model concerns four state variables (u, s, v, t)

and the ODEs are highly nonlinear, such as:

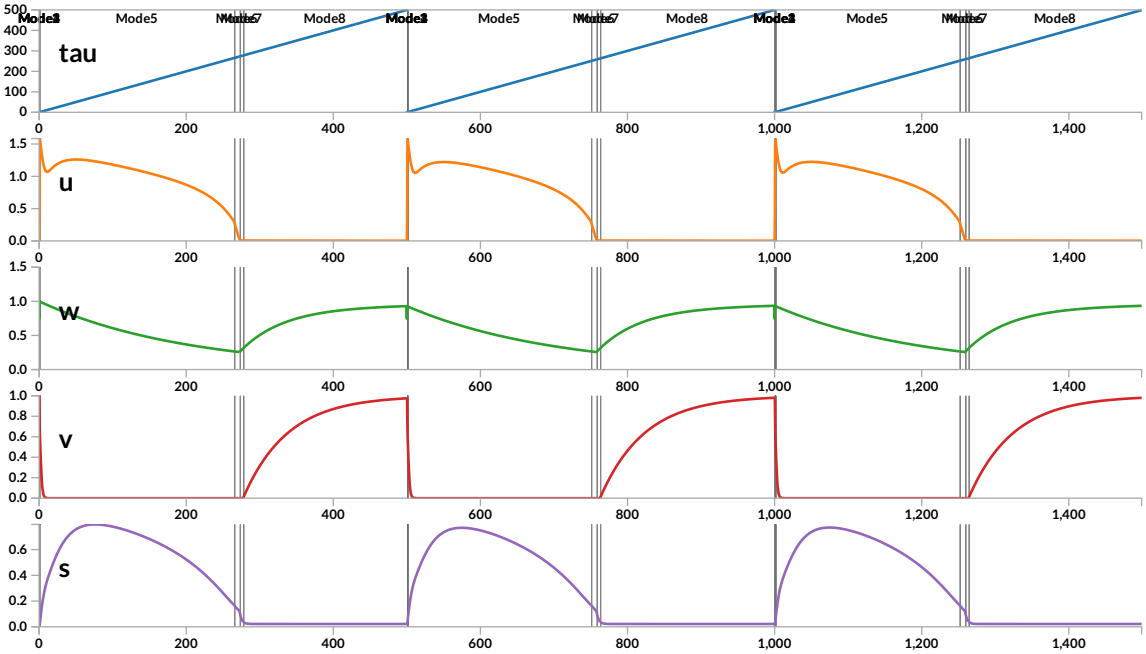
$$\begin{aligned}
\frac{du}{dt} &= e + (u - \theta_v)(u_u - u)vg_{fi} + wsg_{si} - g_{so}(u) \\
\frac{ds}{dt} &= \frac{g_{s2}}{(1 + e^{-2k(u-us)})} - g_{s2}s \\
\frac{dv}{dt} &= -g_v^+ \cdot v \\
\frac{dw}{dt} &= -g_w^+ \cdot w.
\end{aligned}$$

The exponential term on the right-hand side of the ODE is the sigmoid function, which often appears in modeling biological switches. On this model, our tool is able to perform a depth-55 unrolling, and solve the generated logic formula. Such a formula contains 280 nonlinear ODEs of the type shown here, with 616 variables. The computed trace from dReal suggests a witness of the reachability property that can be confirmed by experimental simulation. Figure 3.6 shows the comparison between the trace computed from bounded model checking and the actual experimental simulation trace.

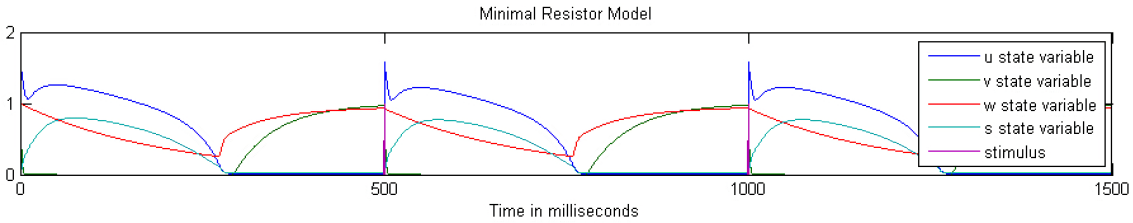
Prostate Cancer Treatment Model The Prostate Cancer Treatment model [87] exhibits more nonlinear ODEs. This model's dynamics is captured by the following ODEs:

$$\begin{aligned}
\frac{dx}{dt} &= (\alpha_x(k_1 + (1 - k_1)\frac{z}{z + k_2} - \beta_x((1 - k_3)\frac{z}{z + k_4} + k_3)) - m_1(1 - \frac{z}{z_0}))x + c_1x \\
\frac{dy}{dt} &= m_1(1 - \frac{z}{z_0})x + (\alpha_y(1 - d\frac{z}{z_0}) - \beta_y)y + c_2y \\
\frac{dz}{dt} &= \frac{-z}{\tau} + c_3z \\
\frac{dv}{dt} &= (\alpha_x(k_1 + (1 - k_1)\frac{z}{z + k_2} - \beta_x(k_3 + (1 - k_3)\frac{z}{z + k_4})) \\
&\quad - m_1(1 - \frac{z}{z_0}))x + c_1x + m_1(1 - \frac{z}{z_0})x + (\alpha_y(1 - d\frac{z}{z_0}) - \beta_y)y + c_2y.
\end{aligned}$$

Figure 3.7 shows the comparison between a trace computed from bounded model checking and the actual experimental simulation trace.

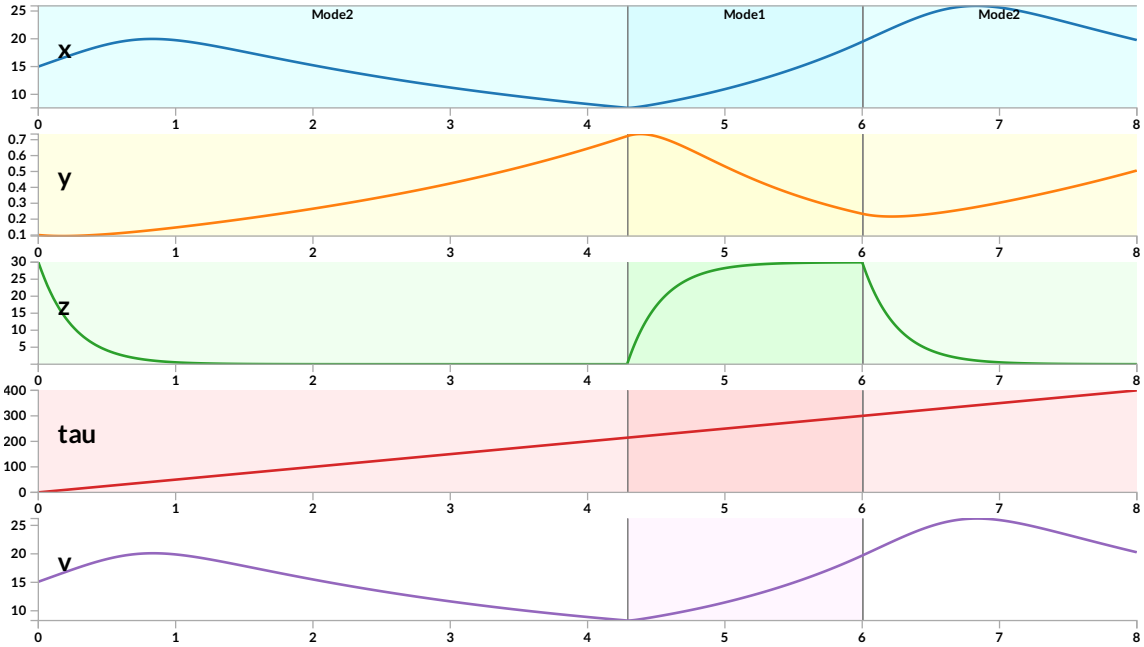


(a) Witness for the Atrial Fibrillation model at depth 23 and 1500 time units.

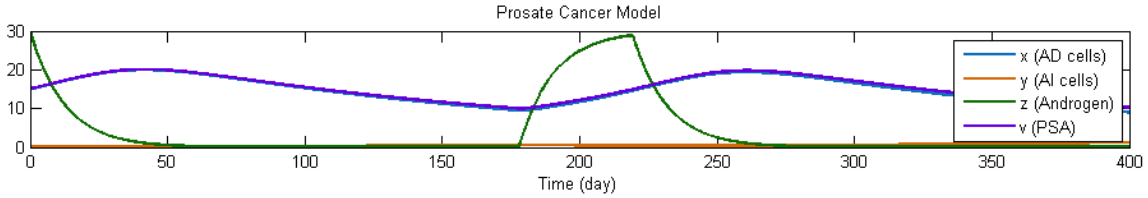


(b) Experimental simulation data. The variables other than tau ared superimposed.

Figure 3.6: Atrial fibrillation model: Comparison between the trace computed from bounded model checking and the actual experimental simulation trace.



(a) Witness computed for the prostate cancer treatment model at depth 2 and 500 time units.



(b) Experimental simulation data.

Figure 3.7: Prostate cancer treatment model: Comparison between the trace computed from bounded model checking and the actual experimental simulation trace.

Electronic Oscillator The EO model represents an electronic oscillator model that contains nonlinear ODEs such as the following:

$$\begin{aligned}
\frac{dx}{dt} &= -ax \cdot \sin(\omega_1 \cdot \tau) \\
\frac{dy}{dt} &= -ay \cdot \sin((\omega_1 + c_1) \cdot \tau) \cdot \sin(\omega_2) \cdot 2 \\
\frac{dz}{dt} &= -az \cdot \sin((\omega_2 + c_2) \cdot \tau) \cdot \cos(\omega_1) \cdot 2 \\
\frac{\omega_1}{dt} &= -c_3 \cdot \omega_1 \\
\frac{\omega_2}{dt} &= -c_4 \cdot \omega_2 \\
\frac{d\tau}{dt} &= 1.
\end{aligned}$$

Quadcopter Control Based on [57], we derived a mathematical model that contains the full dynamics of a quadcopter. We use the model to solve control problems by answering reachability questions. A typical set of the differential equations is the following:

$$
\begin{aligned}
\frac{\mathrm{d}\omega_x}{\mathrm{d}t} &= L \cdot k \cdot (\omega_1^2 - \omega_3^2)(1/I_{xx}) - (I_{yy} - I_{zz})\omega_y\omega_z/I_{xx} \\
\frac{\mathrm{d}\omega_y}{\mathrm{d}t} &= L \cdot k \cdot (\omega_2^2 - \omega_4^2)(1/I_{yy}) - (I_{zz} - I_{xx})\omega_x\omega_z/I_{yy} \\
\frac{\mathrm{d}\omega_z}{\mathrm{d}t} &= b \cdot (\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2)(1/I_{zz}) - (I_{xx} - I_{yy})\omega_x\omega_y/I_{zz} \\
\frac{\mathrm{d}\phi}{\mathrm{d}t} &= \omega_x + \frac{\sin(\phi)\sin(\theta)}{\left(\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)} + \cos(\phi)\cos(\theta)\right)\cos(\phi)}\omega_y + \frac{\sin(\theta)}{\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)} + \cos(\phi)\cos(\theta)}\omega_z \\
\frac{\mathrm{d}\theta}{\mathrm{d}t} &= -\Big(\frac{\sin(\phi)^2\cos(\theta)}{\left(\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)}\omega_y + \cos(\phi)\cos(\theta)\right)\cos(\phi)^2} + \frac{1}{\cos(\phi)}\Big)\omega_y \\
&\quad - \frac{\sin(\phi)\cos(\theta)}{\left(\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)} + \cos(\phi)\cos(\theta)\right)\cos(\phi)}\omega_z \\
\frac{\mathrm{d}\psi}{\mathrm{d}t} &= \frac{\sin(\phi)}{\left(\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)} + \cos(\phi)\cos(\theta)\right)\cos(\phi)}\omega_y + \frac{1}{\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)} + \cos(\phi)\cos(\theta)}\omega_z \\
\frac{\mathrm{d}xp}{\mathrm{d}t} &= (1/m)(\sin(\theta)\sin(\psi)k(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) - k \cdot d \cdot xp) \\
\frac{\mathrm{d}yp}{\mathrm{d}t} &= (1/m)(-\cos(\psi)\sin(\theta)k(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) - k \cdot d \cdot yp) \\
\frac{\mathrm{d}zp}{\mathrm{d}t} &= (1/m)(-g - \cos(\theta)k(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) - k \cdot d \cdot zp) \\
\frac{\mathrm{d}x}{\mathrm{d}t} &= xp \\
\frac{\mathrm{d}y}{\mathrm{d}t} &= yp \\
\frac{\mathrm{d}z}{\mathrm{d}t} &= zp.
\end{aligned}
$$

The other models are standard simple nonlinear models (for instance, bouncing ball with nonlinear friction), on which our tool has no difficulty in solving.

3.5 Conclusion

In this chapter we have studied SMT problems over the real numbers with ODE constraints. We have developed δ-complete algorithms in the DPLL$\langle$ICP$\rangle$ framework, for both the standard SMT formulas that are purely existentially quantified, as well as $\exists\forall$-formulas whose universal quantification is restricted to the time variables. We have demonstrated the scalability of our approach on nonlinear SMT benchmarks. We believe that the proposed decision procedures can scale on nonlinear problems and can serve as the underlying engine for formal verification of realistic hybrid systems and embedded software.

Chapter 4

Delta-Decision Procedures for Exists-Forall Problems over the Reals

4.1 Introduction

Much progress has been made in the framework of delta-decision procedures for solving nonlinear Satisfiability Modulo Theories (SMT) problems over real numbers [49, 48]. Delta-decision procedures allow one-sided bounded numerical errors, which is a practically useful relaxation that significantly reduces the computational complexity of the problems. With such relaxation, SMT problems with hundreds of variables and highly nonlinear constraints (such as differential equations) have been solved in practical applications [79]. Existing work in this direction has focused on satisfiability of quantifier-free SMT problems. Going one level up, SMT problems with both free and universally quantified variables, which correspond to $\exists\forall$-formulas over the reals, are much more expressive. For instance, such formulas can encode the search for robust control laws in highly nonlinear dynamical systems, a central problem in robotics. Non-convex, multi-objective, and disjunctive optimization problems can all be encoded as $\exists\forall$-formulas, through the natural definition of "finding some x such that for all other x', x is better than x'

with respect to certain constraints." Many other examples from various areas are listed in [103].

Counterexample-Guided Inductive Synthesis (CEGIS) [109] is a framework for program synthesis that can be applied to solve generic exists-forall problems. The idea is to break the process of solving $\exists\forall$-formulas into a loop between *synthesis* and *verification*. The synthesis procedure finds solutions to the existentially quantified variables and gives the solutions to the verifier to see if they can be validated, or falsified by *counterexamples*. The counterexamples are then used as learned constraints for the synthesis procedure to find new solutions. This method has been shown effective for many challenging problems, frequently generating more optimized programs than the best manual implementations [109].

A direct application of CEGIS to decision problems over real numbers, however, suffers from several problems. CEGIS is complete in finite domains because it can explicitly enumerate solutions, which can not be done in continuous domains. Also, CEGIS ensures progress by avoiding duplication of solutions, while due to numerical sensitivity, precise control over real numbers is difficult. In this chapter we propose methods that bypass such difficulties.

We propose an integration of the CEGIS method in the branch-and-prune framework as a generic algorithm for solving nonlinear $\exists\forall$-formulas over real numbers and prove that the algorithm is δ-complete. We achieve this goal by using CEGIS-based methods for turning universally-quantified constraints into pruning operators, which is then used in the branch-and-prune framework for the search for solutions on the existentially-quantified variables. In doing so, we take special care to ensure correct handling of numerical errors in the computation, so that δ-completeness can be established for the whole procedure.

This chapter is organized as follows. We first review the background, and then present the details of the main algorithm in Section 4.3. We then give a rigorous proof of the δ-completeness of the procedure in Section 4.4. We demonstrated the effectiveness of the procedures on various global optimization and Lyapunov

function synthesis problems in Section 4.5.

Related Work Quantified formulas in real arithmetic can be solved using symbolic quantifier elimination (using cylindrical decomposition [31]), which is known to have impractically high complexity (double exponential [18]), and can not handle problems with transcendental functions. State-of-the-art SMT solvers such as CVC4 [10] and Z3 [34] provide quantifier support [92, 16, 56, 104] but they are limited to decidable fragments of first-order logic. Optimization Modulo Theories (OMT) is a new field that focuses on solving a restricted form of quantified reasoning [94, 28, 105], focusing on linear formulas. Generic approaches to solving exists-forall problems such as [36] are generally based on CEGIS framework, and not intended to achieve completeness. Solving quantified constraints has been explored in the constraint solving community [95]. In general, existing work has not proposed algorithms that intend to achieve any notion of completeness for quantified problems in nonlinear theories over the reals.

4.2 Preliminaries

4.2.1 Delta-Decisions and CNF$^\forall$-Formulas

We will focus on the $\exists\forall$-formulas in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ (Definition 3) in this chapter. Decision problems for such formulas are equivalent to satisfiability of SMT with universally quantified variables, whose free variables are implicitly existentially quantified.

It is clear that, when the quantifier-free part of an $\exists\forall$ formula is in Conjunctive Normal Form (CNF), we can always push the universal quantifiers inside each conjunct, since universal quantification commute with conjunctions. Thus the decision problem for any $\exists\forall$-formula is equivalent to the satisfiability of formulas in the following normal form:

Definition 20 (CNF$^\forall$ Formulas in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$)**.** *We say an $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula φ is in the CNF$^\forall$, if it is of the form*

$$\varphi(\vec{x}) := \bigwedge_{i=0}^{m} \left(\forall \vec{y}(\bigvee_{j=0}^{k_i} c_{ij}(\vec{x}, \vec{y}))\right) \tag{4.1}$$

where c_{ij} are atomic constraints. Each universally quantified conjunct of the formula, i.e.,

$$\forall \vec{y}(\bigvee_{j=0}^{k_i} c_{ij}(\vec{x}, \vec{y}))$$

is called as a ***$\forall$-clause****. Note that $\forall$-clauses only contain disjunctions and no nested conjunctions. If all the $\forall$-clauses are vacuous, we say $\varphi(\vec{x})$ is a* ground SMT *formula.*

The algorithms described in this chapter will assume that an input formula is in CNF$^\forall$ form. We can now define the *δ-satisfiability* problems for CNF$^\forall$-formulas.

Definition 21 (Delta-Weakening/Strengthening of CNF$^\forall$-formulas)**.** *Let $\delta \in \mathbb{Q}^+$ be arbitrary. Consider an arbitrary CNF$^\forall$-formula of the form*

$$\varphi(\vec{x}) := \bigwedge_{i=0}^{m} \left(\forall \vec{y}(\bigvee_{j=0}^{k_i} f_{ij}(\vec{x}, \vec{y}) \circ 0)\right)$$

where $\circ \in \{>, \geq\}$. We define the δ-weakening of $\varphi(\vec{x})$ to be:

$$\varphi^{-\delta}(\vec{x}) := \bigwedge_{i=0}^{m} \left(\forall \vec{y}(\bigvee_{j=0}^{k_i} f_{ij}(\vec{x}, \vec{y}) \geq -\delta)\right).$$

Namely, we weaken the right-hand sides of all atomic formulas from 0 to $-\delta$. Note how the difference between strict and nonstrict inequality becomes unimportant in the δ-weakening. We also define its dual, the δ-strengthening of $\varphi(\vec{x})$:

$$\varphi^{+\delta}(\vec{x}) := \bigwedge_{i=0}^{m} \left(\forall \vec{y}(\bigvee_{j=0}^{k_i} f_{ij}(\vec{x}, \vec{y}) \geq +\delta)\right).$$

Since the formulas in the normal form no longer contain negations, the relaxation on the atomic formulas is implied by the original formula (and thus weaker), as was easily shown in [49].

Proposition 2. *For any φ and $\delta \in \mathbb{Q}^+$, $\varphi^{-\delta}$ is logically weaker, in the sense that $\varphi \to \varphi^{-\delta}$ is always true, but not vice versa.*

Example 1. *Consider the formula*

$$\forall y \; f(x, y) = 0.$$

It is equivalent to the CNF$^\forall$-formula

$$(\forall y(-f(x, y) \geq 0) \wedge \forall y(f(x, y) \geq 0))$$

whose δ-weakening is of the form

$$(\forall y(-f(x, y) \geq -\delta) \wedge \forall y(f(x, y) \geq -\delta))$$

which is logically equivalent to

$$\forall y(\|f(x, y)\| \leq \delta).$$

We see that the weakening of $f(x, y) = 0$ by $\|f(x, y)\| \leq \delta$ defines a natural relaxation.

4.2.2 The Branch-and-Prune Framework

A practical algorithm that has been shown to be δ-complete for ground SMT formulas is the *branch-and-prune* method developed for interval constraint propagation [13]. A description of the algorithm in the simple case of an equality constraint is in Algorithm 4.1.

The procedure combines *pruning* and *branching* operations. Let $\mathcal{B}$ be the set of all boxes (each variable assigned to an interval), and C a set of constraints in the language. FixedPoint(g, B) is a procedure computing a fixedpoint of a function $g : \mathcal{B} \to \mathcal{B}$ with an initial input B. A pruning operation Prune : $\mathcal{B} \times C \to \mathcal{B}$ takes a box $B \in \mathcal{B}$ and a constraint as input, and returns an ideally smaller box $B' \in \mathcal{B}$ (Line 5) that is guaranteed to still keep all solutions for all constraints if there is any. When such pruning operations do not make progress, the Branch procedure

Algorithm 4.1 Branch-and-Prune

```
1:  function Solve(f(x) = 0, B_x, δ)
2:      S ← {B_x}
3:      while S ≠ ∅ do
4:          B ← S.pop()
5:          B' ← FixedPoint(λB.B ∩ Prune(B, f(x) = 0), B)
6:          if B' ≠ ∅ then
7:              if ‖f(B')‖ > δ then
8:                  {B_1, B_2} ← Branch(B')
9:                  S.push({B_1, B_2})
10:             else
11:                 return δ-sat
12:             end if
13:         end if
14:     end while
15:     return unsat
16: end function
```

picks a variable, divides its interval by halves, and creates two sub-problems B_1 and B_2 (Line 8). The procedure terminates if either all boxes have been pruned to be empty (Line 15), or if a small box whose maximum width is smaller than a given threshold δ has been found (Line 11). In [48], it has been proved that Algorithm 4.1 is δ-complete if and only if the pruning operators satisfy certain conditions for being *well-defined* (Definition 22).

4.3 Algorithms

The core idea of our algorithm for solving CNF$^\forall$-formulas is as follows. We view the universally quantified constraints as a special type of pruning operators, which can be used to reduce possible values for the free variables based on their consistency with the universally-quantified variables. We then use these special $\forall$-pruning operators in an overall branch-and-prune framework to solve the full formula in a way that ensures δ-completeness. A special technical difficulty for

ensuring δ-completeness is to control numerical errors in the recursive search for counterexamples, which we solve using *double-sided error control*. We also improve quality of counterexamples using local-optimization algorithms in the $\forall$-pruning operations, which we call *locally-optimized counterexamples*.

In the following sections we describe these steps in detail. For notational simplicity we will omit vector symbols and assume all variable names can directly refer to vectors of variables.

4.3.1 $\forall$-Clauses as Pruning Operators

Consider an arbitrary CNF$^{\forall}$-formula. Note that without loss of generality we only use non-strict inequality here, since in the context of δ-decisions the distinction between strict and non-strict inequalities as not important, as explained in Definition 5.

$$\varphi(x) := \bigwedge_{i=0}^{m} \Big(\forall y(\bigvee_{j=0}^{k_i} f_{ij}(x,y) \geq 0)\Big).$$

It is a conjunction of $\forall$-clauses as defined in Definition 20. Consequently, we only need to define pruning operators for $\forall$-clauses so that they can be used in a standard branch-and-prune framework. The full algorithm for such pruning operation is described in Algorithm 4.2.

In Algorithm 4.2, the basic idea is to use special y values that witness the *negation* of the original constraint to prune the box assignment on x. The two core steps are as follows.

1. Counterexample generation (Lines 4 – 9). The query for a counterexample ψ is defined as the negation of the quantifier-free part of the constraint (Line 4). The method $\textsf{Solve}(y, \psi, \delta)$ means to obtain a solution for the variables y δ-satisfying the logic formula ψ. When such a solution is found, we have a counterexample that can falsify the $\forall$-clause on some choice of x. Then we use this counterexample to prune on the domain of x, which is currently B_x.

Algorithm 4.2 ∀-Clause Pruning

1: **function** PRUNE(B_x, B_y, $\forall y \bigvee_{i=0}^{k} f_i(x,y) \geq 0$, δ', ε, δ)
2: **repeat**
3: $B_x^{\text{prev}} \leftarrow B_x$
4: $\psi \leftarrow \bigwedge_i f_i(x,y) < 0$
5: $\psi^{+\varepsilon} \leftarrow \mathsf{Strengthen}(\psi, \varepsilon)$
6: $b \leftarrow \mathsf{Solve}(y, \psi^{+\varepsilon}, \delta')$ ▷ $0 < \delta' < \varepsilon < \delta$ should hold.
7: **if** $b = \emptyset$ **then**
8: **return** B_x ▷ No counterexample found, stop pruning.
9: **end if**
10: **for** $i \in \{0, \ldots, k\}$ **do**
11: $B_i \leftarrow B_x \cap \mathsf{Prune}\Big(B_x, f_i(x,b) \geq 0\Big)$
12: **end for**
13: $B_x \leftarrow \bigsqcup_{i=0}^{k} B_i$
14: **until** $B_x \neq B_x^{\text{prev}}$
15: **return** B_x
16: **end function**

The strengthening operation on ψ (Line 5), as well as the choices of ε and δ', will be explained in the next subsection.

2. Pruning on x (Lines 10 – 13). In the counterexample generation step, we have obtained a counterexample b. The pruning operation then uses this value to prune on the current box domain B_x. Here we need to be careful about the logical operations. For each constraint, we need to take the intersection of the pruned results on the counterexample point (Line 11). Then since the original clause contains the disjunction of all constraints, we need to take the box-hull ($\bigsqcup$) of the pruned results (Line 13).

We can now put the pruning operators defined for all ∀-clauses in the overall branch-and-prune framework shown in Algorithm 4.1.

The pruning algorithms are inspired by the CEGIS loop but differ in multiple ways. First, we do not explicitly compute any candidate solutions for the free variables. Instead, we only prune their domain boxes. This ensures that the size of

the domain box decreases, together with branching operations, and the algorithm terminates. Second, we do not explicitly maintain a collection of constraints. Each time the pruning operation works on the previous box, i.e., learning is done on the model level instead of the constraint level. On the other hand, our inability to maintain arbitrary Boolean combinations of constraints requires us to be more sensitive to the type of Boolean operations needed in the pruning results, which differs from the CEGIS approach that treats solvers as black boxes.

4.3.2 Double-Sided Error Control

To ensure the correctness of Algorithm 4.2, it is necessary to avoid spurious counterexamples which do *not* satisfy the negation of the quantified part in a ∀-clause. We illustrate this condition by consider a *wrong* derivation of Algorithm 4.2 where we do not have the strengthening operation on Line 5 and try to find a counterexample by directly executing $b \leftarrow \text{Solve}(y, \psi = \bigwedge_{i=0}^{k} f_i(x, y) < 0, \delta)$. Note that the counterexample query ψ can be highly nonlinear in general and not included in a decidable fragment. As a result, it must employ a delta-decision procedure (i.e., Solve with $\delta' \in \mathbb{Q}^+$) to find a counterexample. A consequence of relying on a delta-decision procedure in the counterexample generation step is that we may obtain a spurious counterexample b such that for some $x = a$:

$$\bigwedge_{i=0}^{k} f_i(a, b) \leq \delta \quad \text{instead of} \quad \bigwedge_{i=0}^{k} f_i(a, b) < 0.$$

Consequently the following pruning operations fail to reduce their input boxes because a spurious counterexample does not witness any inconsistencies between x and y. As a result, the fixedpoint loop in this ∀-Clause pruning algorithm will be terminated immediately after the first iteration. This makes the outer-most branch-and-prune framework (Algorithm 4.1), which employs this pruning algorithm, solely rely on branching operations. It can claim that the problem is δ-satisfiable while providing an arbitrary box B as a model which is small enough ($\|B\| \leq \delta$) but does not include a δ-solution.

To avoid spurious counterexamples, we directly strengthen the counterexample query with $\varepsilon \in \mathbb{Q}^+$ to have

$$\psi^{+\varepsilon} := \bigwedge_{i=0}^{k} f_i(a, b) \leq -\varepsilon.$$

Then we choose a weakening parameter $\delta' \in \mathbb{Q}$ in solving the strengthened formula. By analyzing the two possible outcomes of this counterexample search, we show the constraints on δ', ε, and δ which guarantee the correctness of Algorithm 4.2:

- **δ'-sat case**: We have a and b such that $\bigwedge_{i=0}^{k} f_i(a, b) \leq -\varepsilon + \delta'$. For $y = b$ to be a valid counterexample, we need $-\varepsilon + \delta' < 0$. That is, we have

  $$\delta' < \varepsilon. \tag{4.2}$$

  In other words, the strengthening factor ε should be greater than the weakening parameter δ' in the counterexample search step.

- **unsat case**: By checking the absence of counterexamples, it proved that $\forall y \bigvee_{i=0}^{k} f_i(x, y) \geq -\varepsilon$ for all $x \in B_x$. Recall that we want to show that $\forall y \bigvee_{i=0}^{k} f_i(x, y) \geq -\delta$ holds for some $x = a$ when Algorithm 4.1 uses this pruning algorithm and returns δ-sat. To ensure this property, we need the following constraint on ε and δ:

  $$\varepsilon < \delta. \tag{4.3}$$

4.3.3 Locally-Optimized Counterexamples

The performance of the pruning algorithm for $\text{CNF}^{\forall}$-formulas depends on the quality of the counterexamples found during the search.

Figure 4.1 illustrates this point by visualizing a pruning process for an unconstrained minimization problem, $\exists x \in X_0 \forall y \in X_0 f(x) \leq f(y)$. As it finds a series of counterexamples CE_1, CE_2, CE_3, and CE_4, the pruning algorithm uses those counterexamples to contract the interval assignment on X from X_0 to X_1, X_2, X_3, and

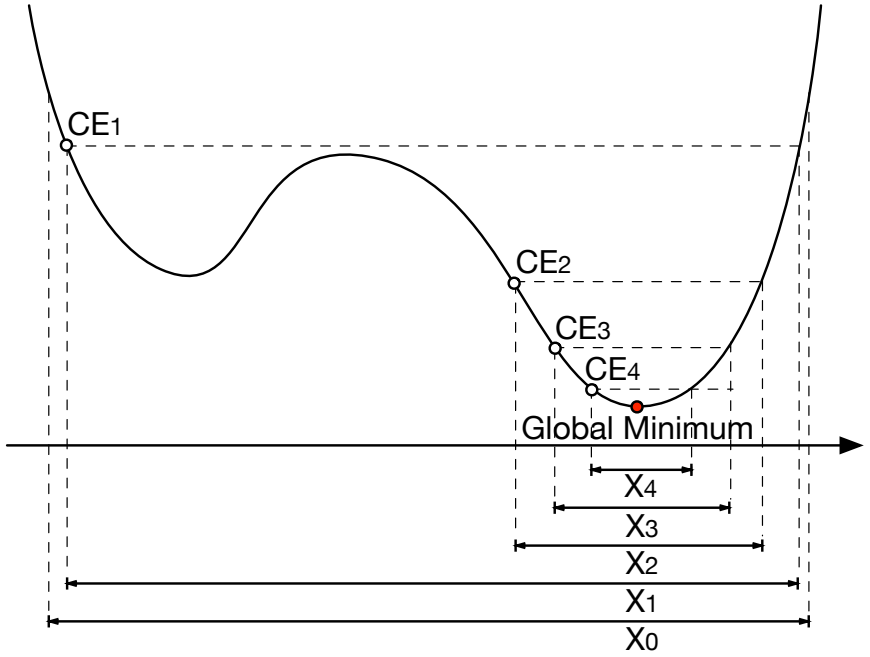


Figure 4.1: Illustration of the pruning algorithm for CNF$^\forall$-formulas without local optimization.

X_4 in sequence. In the search for a counterexample (Line 6 of Algorithm 4.2), it solves the strengthened query, $f(x) > f(y) + \varepsilon$. Note that the query only requires a counterexample $y = b$ to be ε-away from a candidate x while it is clear that the further a counterexample is away from candidates, the more effective the pruning algorithm is.

Based on this observation, we present a way to improve the performance of the pruning algorithm for CNF$^\forall$-formulas. After we obtain a counterexample b, we locally-optimize it with the counterexample query ψ so that it "further violates" the constraints. Figure 4.2 illustrates this idea. The algorithm first finds a counterexample CE_1 then refines it to CE'_1 by using a local-optimization algorithm (similarly, $\mathrm{CE}_2 \rightarrow \mathrm{CE}'_2$). Clearly, this refined counterexample gives a stronger pruning power than the original one. This refinement process can also help the performance of the algorithm by reducing the number of total iterations in the fixedpoint loop.

The suggested method is based on the assumption that local optimization tech-

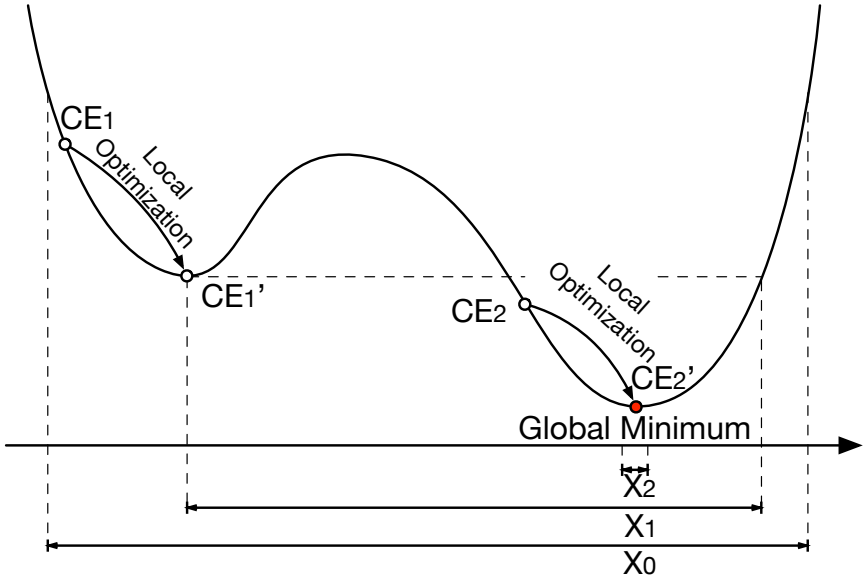


Figure 4.2: Illustration of the pruning algorithm for CNF$^{\forall}$-formulas with local optimization.

niques are cheaper than finding a global counterexample using interval propagation techniques. In our experiments, we observed that this assumption holds true in practice. We provide more details in Section 4.5.

4.4 δ-Completeness

We now prove that the proposed algorithm is δ-complete for arbitrary CNF$^{\forall}$ formulas in $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$. In the work of [48], δ-completeness has been proved for branch-and-prune for ground SMT problems, under the assumption that the pruning operators are *well-defined*. Thus, the key for our proof here is to show that the $\forall$-pruning operators satisfy the conditions of well-definedness.

The notion of a well-defined pruning operator is defined in [48] as follows.

Definition 22. *Let ϕ be a constraint, and $\mathcal{B}$ be the set of all boxes in $\mathbb{R}^n$. A pruning operator is a function* $\mathsf{Prune} : \mathcal{B} \times \mathcal{C} \to \mathcal{B}$. *We say such a pruning operator is well-defined, if for any $B \in \mathcal{B}$, the following conditions are true:*

1. $\mathsf{Prune}(B, \phi) \subseteq B$.
2. $B \cap \{a \in \mathbb{R}^n : \phi(a) \text{ is true.}\} \subseteq \mathsf{Prune}(B, \phi)$.
3. *Write* $\mathsf{Prune}(B, \phi) = B'$. *There exists a constant $c \in \mathbb{Q}^+$, such that, if $B' \neq \emptyset$ and $\|B'\| < \varepsilon$ for some $\varepsilon \in \mathbb{Q}^+$, then for all $a \in B'$, $\phi^{-c\varepsilon}(a)$ is true.*

We will explain the intuition behind these requirements in the next proof, which aims to establish that Algorithm 4.2 defines a well-defined pruning operator.

Lemma 1 (Well-definedness of ∀-Pruning)**.** *Consider an arbitrary ∀-clause in the generic form*

$$c(x) := \forall y \Big(f_1(x, y) \geq 0 \vee \ldots \vee f_k(x, y) \geq 0 \Big).$$

Suppose the pruning operators for $f_1 \geq 0, \ldots, f_k \geq 0$ are well-defined, then the ∀-pruning operation for $c(x)$ as described in Algorithm 4.2 is well-defined.

Proof. We prove that the pruning operator defined by Algorithm 4.2 satisfies the three conditions in Definition 22. Let $B_0, \ldots, B_k$ be a sequence of boxes, where B_0 is the input box B_x and B_k is the returned box B, which is possibly empty.

1. The first condition requires (1) that the pruning operation for $c(x)$ is reductive. That is, we want to show that $B_x \subseteq B_x^{\text{prev}}$ holds in Algorithm 4.2. If it does not find a counterexample (Line 8), we have $B_x = B_x^{\text{prev}}$. So the condition holds trivially. Consider the case where it finds a counterexample b. The pruned box B_x is obtained through box-hull of all the B_i boxes (Line 13), which are results of pruning on B_x^{prev} using ordinary constraints of the form $f_i(x, b) \geq 0$ (Line 11), for a counterexample b. Following the assumption that the pruning operators are well-defined for each ordinary constraint f_i used in the algorithm, we know

that $B_i \subseteq B_x^{\text{prev}}$ holds as a loop invariant for the loop from Line 10 to Line 12. Thus, taking the box-hull of all the B_i, we obtain B_x that is still a subset of B_x^{prev}.

2. The second condition (2) requires that the pruning operation does not eliminate real solutions. Again, by the assumption that the pruning operation on Line 11 does not lose any valid assignment on x that makes the $\forall$-clause true. In fact, since y is universally quantified, any choice of assignment $y = b$ will preserve solution on x as long as the ordinary pruning operator is well-defined. Thus, this condition is easily satisfied.

3. The third condition (3) is the most nontrivial to establish. It ensures that when the pruning operator does not prune a box to the empty set, then the box should not be "way off", and in fact, should contain points that satisfy an appropriate relaxation of the constraint. We can say this is a notion of "faithfulness" of the pruning operator. For constraints defined by simple continuous functions, this can be typically guaranteed by the modulus of continuity of the function (Lipschitz constants as a special case). Now, in the case of $\forall$-clause pruning, we need to prove that the faithfulness of the ordinary pruning operators that are used translates to the faithfulness of the $\forall$-clause pruning results. First of all, this condition would not hold, if we do not have the strengthening operation when searching for counterexamples (Line 5). As is shown in Example 1, because of the weakening that δ-decisions introduce when searching for a counterexample, we may obtain a *spurious counterexample* that does not have pruning power. In other words, if we keep using a wrong counterexample that already satisfies the condition, then we are not able to rule out wrong assignments on x. Now, since we have introduced ε-strengthening at the counterexample search, we know that b obtained on Line 6 is a true counterexample. Thus, for some $x = a$, $f_i(a, b) < 0$ for every i. By assumption, the ordinary pruning operation using b on Line 11 guarantees faithfulness. That is, suppose the pruned result B_i is not empty and $\|B_i\| \leq \varepsilon$,

then there exists constant c_i such that $f_i(x, b) \geq -c_i\varepsilon$ is true. Thus, we can take the $c = \min_i c_i$ as the constant for the pruning operator defined by the full clause, and conclude that the disjunction $\bigvee_{i=0}^{k} f_i(x, y) < -c\varepsilon$ holds for $\|B_x\| \leq \varepsilon$.

□

Using the lemma, we follow the results in [48], and conclude that the branch-and-prune method in Algorithm 4.1 is delta-complete:

Theorem 4 (δ-Completeness)**.** *For any $\delta \in \mathbb{Q}^+$, using the proposed $\forall$-pruning operators defined in Algorithm 4.2 in the branch-and-prune framework described in Algorithm 4.1 is δ-complete for the class of CNF$^\forall$-formulas in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$, assuming that the pruning operators for all the base functions are well-defined.*

Proof. Following Theorem 4.2 (δ-Completeness of ICP$_\varepsilon$) in [48], a branch-and-prune algorithm is δ-complete iff the pruning operators in the algorithm are all well-defined. Following Lemma 1, Algorithm 4.2 always defines well-defined pruning operators, assuming the pruning operators for the base functions are well-defined. Consequently, Algorithm 4.2 and Algorithm 4.1 together define a delta-complete decision procedure for CNF$^\forall$-problems in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$. □

4.5 Experiments

Implementation We implemented the algorithms in dReal [52][1], an open-source δ-SMT solver. We used IBEX-lib [112] for interval constraints pruning and CLP [88] for linear programming. For local optimization, we used NLopt [74]. In particular, we used SLSQP (Sequential Least-Squares Quadratic Programming) local-optimization algorithm [81] for differentiable constraints and COBYLA (Constrained Optimization BY Linear Approximations) local-optimization algorithm [102] for non-differentiable constraints. The prototype solver is able to handle $\exists\forall$-formulas that involve most

[1]The tool is available on `https://github.com/dreal/dreal4` and released under the Apache-2.0 license.

standard elementary functions, including power, exp, log, $\sqrt{\cdot}$, trigonometric functions (sin, cos, tan), inverse trigonometric functions (arcsin, arccos, arctan), hyperbolic functions (sinh, cosh, tanh), etc.

Experiment Environment All experiments were ran on an AWS EC2 instance with AMD EPYC 7R32 and 32 GB RAM running Ubuntu 20.04 LTS. All code and benchmarks are available at `https://github.com/dreal/CAV18`.

Parameters In the experiments, we chose the strengthening parameter $\varepsilon = 0.5\delta$ and the weakening parameter in the counterexample search $\delta' = 0.5\varepsilon$. In each call to NLopt, we used $1e-6$ for both of absolute and relative tolerances on function value, $1e-3$ seconds for a timeout, and 100 for the maximum number of evaluations. These values are used as stopping criteria in NLopt.

4.5.1 Nonlinear Global Optimization

We encoded a range of highly nonlinear ∃∀-problems from constrained and unconstrained optimization literature [72, 117]. Note that the standard optimization problem

$$\min f(x) \text{ s.t. } \varphi(x), \quad x \in \mathbb{R}^n,$$

can be encoded as the logic formula:

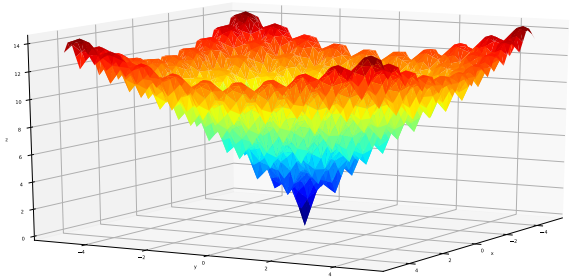
$$\varphi(x) \wedge \forall y \Big(\varphi(y) \rightarrow f(x) \leq f(y) \Big).$$

As plotted in Figure 4.3, these optimization problems are non-trivial: they are highly non-convex problems that are designed to test global optimization or genetic programming algorithms. Many such functions have a large number of local minima. For example, Ripple 1 Function [72]

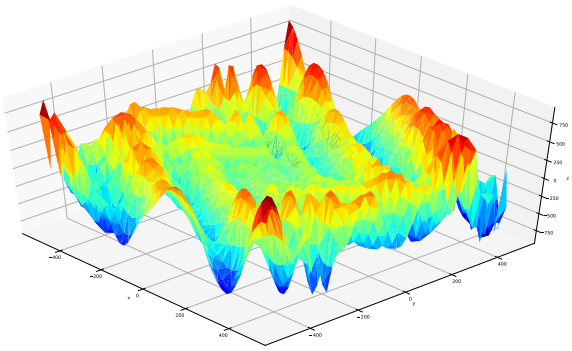
$$f(x_1, x_2) = \sum_{i=1}^{2} -e^{-2(\log 2)\left(\frac{x_1 - 0.1}{0.8}\right)^2} (\sin^6(5\pi x_i) + 0.1\cos^2(500\pi x_i))$$

Name	Solution			Time (sec)		
	Global	No L-Opt.	L-Opt.	No L-Opt.	L-Opt.	Speed Up
Ackley 2D	0.00000	0.00000	0.00000	0.0216	0.0029	7.45
Ackley 4D	0.00000	0.00005	0.00000	1.3958	0.0145	96.26
Aluffi Pentini	-0.35230	-0.35231	-0.35238	0.0149	0.0270	0.55
Beale	0.00000	0.00033	0.00000	0.0147	0.0424	0.35
Bohachevsky1	0.00000	0.00000	0.00000	0.0053	0.0013	4.08
Booth	0.00000	0.00036	0.00000	0.1326	0.0012	110.50
Brent	0.00000	0.00004	0.00000	0.0079	0.0010	7.90
Bukin6	0.00000	0.00015	0.00007	0.0040	0.0027	1.48
Cross in Tray	-2.06261	-2.06254	-2.06253	0.0905	0.0242	3.74
Easom	-1.00000	-1.00000	-1.00000	0.0033	0.0016	2.06
EggHolder	-959.64070	-959.63980	-959.64006	0.0365	0.0123	2.97
Holder Table2	-19.20850	-19.20837	-19.20824	7.0523	4.3070	1.64
Levi N13	0.00000	0.00006	0.00000	0.0517	0.0016	32.31
Ripple 1	-2.20000	-2.20000	-2.20000	0.0056	0.0054	1.04
Schaffer F6	0.00000	0.00038	0.00000	0.0148	0.0040	3.70
Testtube Holder	-10.87230	-10.87230	-10.87230	0.0220	0.0019	11.58
Trefethen	-3.30687	-3.30668	-3.30673	0.7108	0.3554	2.00
W Wavy	0.00000	0.00000	0.00000	0.0515	0.0059	8.73
Zettl	-0.00379	-0.00375	-0.00379	0.0028	0.0022	1.27
Rosenbrock Cubic	0.00000	0.00075	0.00055	0.0034	0.0022	1.55
Rosenbrock Disk	0.00000	0.00001	0.00002	0.0021	0.0014	1.50
Mishra Bird	-106.76454	-106.76393	-106.76445	0.3463	0.0664	5.22
Townsend	-2.02399	-2.02325	-2.02331	0.8872	0.3103	2.86
Simionescu	-0.07262	-0.07194	-0.07198	0.0028	0.0020	1.40

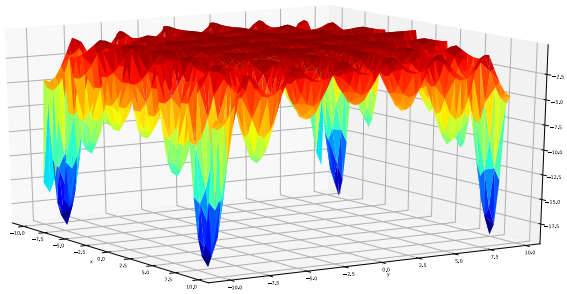
Table 4.1: Experimental results for nonlinear global optimization problems: The first 19 problems (Ackley 2D – Zettl) are unconstrained optimization problems and the last five problems (Rosenbrock Cubic – Simionescu) are constrained optimization problems. We ran our prototype solver over those instances with and without local-optimization option ("L-Opt." and "No L-Opt." columns) and compared the results. We chose $\delta = 0.001, \varepsilon = 0.0005, \delta' = 0.00025$ for all instances.



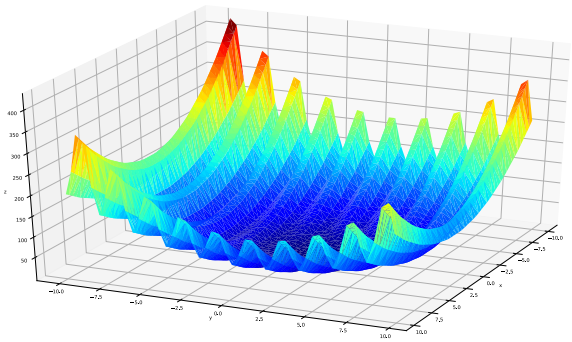
(a) Ackley Function.



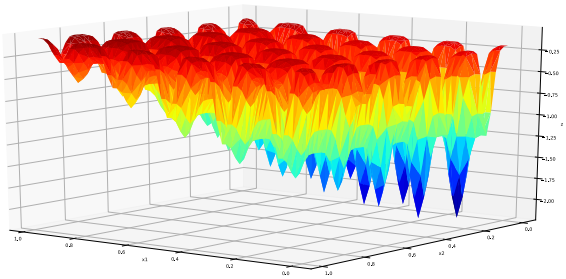
(b) EggHolder Function.



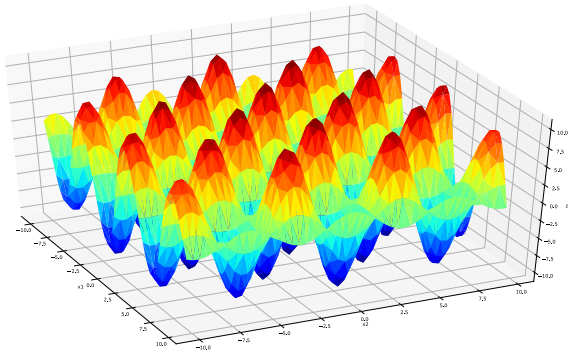
(c) Holder Table2 Function.



(d) Levi N13 Function.



(e) Ripple 1 Function.



(f) Testtube Holder Function.

Figure 4.3: Nonlinear Global Optimization Examples.

defined in $x_i \in [0, 1]$ has 252,004 local minima with the global minima $f(0.1, 0.1) = -2.2$. As a result, local-optimization algorithms such as gradient-descent would not work for these problems for itself. By encoding them as $\exists\forall$-problems, we can perform guaranteed global optimization on these problems.

Table 4.1 provides a summary of the experiment results. First, it shows that we can find minimum values which are close to the known global solutions. Second, it shows that enabling the local-optimization technique speeds up the solving process significantly for 20 instances out of 23 instances.

4.5.2 Synthesizing Lyapunov Function for Dynamical System

We show that the proposed algorithm is able to synthesize Lyapunov functions for nonlinear dynamic systems described by a set of ODEs:

$$\dot{\vec{x}}(t) = f_i(\vec{x}(t)), \quad \forall \vec{x}(t) \in X_i.$$

Lyapunov Function A Lyapunov function is a scalar-valued function that is used to study the stability of a dynamical system. Specifically, it serves as a mathematical tool for proving the stability of a system by demonstrating that the function decreases along the solutions of the system. In particular, a Lyapunov function is a scalar-valued function that is positive definite and decreases along the solutions of the system. A positive definite function is one that is greater than zero everywhere and is equal to zero only at the equilibrium points of the system. By showing that the function decreases along the solutions of the system, it is established that the function has a minimum value at the equilibrium points of the system, thus indicating that the equilibrium points of the system are stable and that solutions of the system converge to these points.

Our Synthesis Approach Our synthesis approach differs from a recent related work [76], in which they used dReal only to verify a candidate function that was found by a simulation-guided algorithm. In contrast, we aim to perform both search

and verification steps by solving a single ∃∀-formula. Note that to verify a Lyapunov candidate function $v : X \to \mathbb{R}^+$, we need to show that the function v satisfies the following conditions:

$$\forall \vec{x} \in X \setminus \vec{0}\ v(\vec{x})(\vec{0}) = 0$$
$$\forall \vec{x} \in X\ \nabla v(\vec{x}(t))^T \cdot f_i(\vec{x}(t)) \leq 0.$$

We assume that a Lyapunov function is a polynomial of some fixed degrees over $\vec{x}$, that is, $v(\vec{x}) = \vec{z}^T \vec{P} \vec{z}$ where $\vec{z}$ is a vector of monomials over $\vec{x}$ and P is a symmetric matrix. Then, we can encode this synthesis problem into the ∃∀-formula:

$$\begin{aligned}\exists \vec{P}\ [&(v(\vec{x}) = (\vec{z}^T \vec{P} \vec{z})) \wedge \\ &(\forall \vec{x} \in X \setminus \vec{0}\ v(\vec{x})(\vec{0}) = 0) \wedge \\ &(\forall \vec{x} \in X\ \nabla v(\vec{x}(t))^T \cdot f_i(\vec{x}(t)) \leq 0)]\end{aligned}$$

In the following sections, we show that we can handle two examples in [76].

Normalized Pendulum

The normalized pendulum is a mathematical model that describes the behavior of a simple pendulum. It is a specific form of the pendulum equation, which is a second-order nonlinear ordinary differential equation that describes the motion of a pendulum.

Given a standard pendulum system with normalized parameters

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\sin(x_1) - x_2 \end{bmatrix}$$

and a quadratic template for a Lyapunov function $v(\vec{x}) = \vec{x}^T \vec{P} \vec{x} = c_1 x_1 x_2 + c_2 x_1^2 + c_3 x_2^2$, we can encode this synthesis problem into the following ∃∀-formula:

$$\begin{aligned}\exists c_1 c_2 c_3\ \forall x_1 x_2\ [&((c_3 x_1 x_2 + x_1^2 c_1 + x_2^2 c_2 > 0) \wedge \\ &(2c_1 x_1 x_2 + x_2 c_3 + (-x_2 - \sin(x_1)(x_1 c_3 + 2x_2 c_2)) < 0)) \vee \\ &\neg((0.01 \leq x_1^2 + x_2^2) \wedge (x_1^2 + x_2^2 \leq 1))].\end{aligned}$$

Our prototype solver takes 44.184 seconds to synthesize the following function as a solution to the problem for the bound $\|\vec{x}\| \in [0.1, 1.0]$ and $c_i \in [0.1, 100]$ using $\delta = 0.05$:

$$v = 40.6843x_1x_2 + 35.6870x_1^2 + 84.3906x_2^2.$$

Damped Mathieu System

The Damped Mathieu system [62] is a type of mathematical model that describes the behavior of a damped oscillator. It is a specific form of the Mathieu equation [114], which is a second-order differential equation that describes the behavior of systems that exhibit periodic motion. It is time-varying and defined by following ODEs:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -x_2 - (2 + \sin(t))x_1 \end{bmatrix}.$$

where $\dot{x}_1$ is the position of the oscillator, $\dot{x}_2$ is the velocity, and t is the time. The solution of this equation describes the behavior of the oscillator over time, including its amplitude, frequency, and phase. The damped Mathieu system can be used to model the behavior of mechanical systems, such as oscillating masses or electrical circuits, and has applications in fields such as control theory, mechanical engineering, and physics.

Using a quadratic template for a Lyapunov function $v(\vec{x}) = \vec{x}^T\vec{P}\vec{x} = c_1x_1x_2 + c_2x_1^2 + c_3x_2^2$, we can encode this synthesis problem into the following $\exists\forall$-formula:

$$\begin{aligned} \exists c_1c_2c_3\ \forall x_1x_2t\ [&(x_1x_2c_2 + x_1^2c_1 + x_2^2c_3 > 0) \wedge \\ &(2c_1x_1x_2 + x_2c_2 + (-x_2 - x_1(2 + \sin(t)))(x_1c_2 + 2x_2c_3) < 0) \\ &\vee \neg((0.01 \leq x_1^2 + x_2^2) \wedge (0.1 \leq t) \wedge (t \leq 1) \wedge (x_1^2 + x_2^2 \leq 1))]. \end{aligned}$$

Our prototype solver takes 26.533 seconds to synthesize the following function as a solution to the problem for the bound $\|\vec{x}\| \in [0.1, 1.0]$, $t \in [0.1, 1.0]$, and $c_i \in [45, 98]$ using $\delta = 0.05$:

$$V = 54.6950x_1x_2 + 90.2849x_1^2 + 50.5376x_2^2.$$

4.6 Conclusion

We have described delta-decision procedures for solving exists-forall formulas in the first-order theory over the reals with computable real functions. These formulas can encode a wide range of hard practical problems such as general constrained optimization and nonlinear control synthesis. We use a branch-and-prune framework, and design special pruning operators for universally-quantified constraints such that the procedures can be proved to be delta-complete, where suitable control of numerical errors is crucial. We demonstrated the effectiveness of the procedures on various global optimization and Lyapunov function synthesis problems.

Chapter 5

dReal: An SMT Solver for Nonlinear Theories over the Reals

5.1 Introduction

SMT formulas over the real numbers can encode a wide range of problems in theorem proving and formal verification. Such formulas are very hard to solve when nonlinear functions are involved. Recent work on δ-complete decision procedures provided a new framework for this problem [48, 49]. We say a decision procedure is *δ-complete* for a set S of SMT formulas, where δ is a positive rational number, if for any φ from S, the procedure returns one of the following:

- unsat: φ is unsatisfiable.
- δ-sat: φ^δ is satisfiable.

Here, φ^δ is a syntactic variant of φ that encodes a notion of numerical perturbation on logic formulas [48]. With such relaxation, δ-complete decision procedures can fully exploit the power of scalable numerical algorithms to solve nonlinear problems, and at the same time provide suitable correctness guarantees for many correctness-critical problems. dReal implements this framework. It solves SMT problems

over the reals with nonlinear functions, such as polynomials, sine, exponentiation, logarithm, etc. The tool is open-source[1], built on PicoSAT [15] to implement the DPLL⟨T⟩, and IBEX-lib [112] for interval pruning algorithms. It returns unsat or δ-sat on input formulas, and the user can obtain certificates (proof of unsatisfiability or solution) for the answers. In this chapter we describe the design, usage and experimental results of the tool.

Related work SMT solving for nonlinear formulas over the reals has gained much attention in recent years and many tools are now available. The symbolic approaches include Cylindrical Decomposition [31], with significant recent improvement [98, 75], and Gröbner bases [100]. A drawback of symbolic algorithms is that it is restricted to arithmetic, namely polynomial constraints, with the exception of [3]. On the other hand, many practical solvers incorporate scalable numerical computations. Examples of numerical algorithms that have been exploited include optimization algorithms [17, 96], interval-based algorithms [43, 40, 50], Bernstein polynomials [93], and linearization algorithms [47]. All solvers show promising results on various nonlinear benchmarks. Our goal is to provide an open-source platform for the rigorous combination of numerical and symbolic algorithms under the framework of δ-complete decision procedures [48].

5.2 Design

5.2.1 The δ-Decision Problem

The standard decision problem is undecidable for SMT formulas over the reals with trigonometric functions. Instead, we proposed to focus on the so-called δ-decision problem, which relaxes the standard decision problem. Let δ be any positive rational number. On a given SMT formula φ, we ask for one of the following answers:

[1] dReal is available at `https://github.com/dreal/dreal4`.

- unsat: φ is unsatisfiable.
- δ-sat: $\varphi^{-\delta}$ is satisfiable.

When the two cases overlap, either answer can be returned. Here, $\varphi^{-\delta}$ is called the δ-perturbation (or δ-weakening) of φ, which is formally defined in Definition 5.

Solving the δ-decision problem is as useful as the standard one for many problems. For instance, suppose we perform bounded model checking on hybrid systems and encode safety properties as an SMT formula φ. Then following standard model checking techniques, if we decide that φ is unsat, then the system is indeed "safe" with in some bounds. If we decide that φ is δ-sat, then the system would become "unsafe" under some δ-perturbation on the system. In this way, when δ is reasonably small, we have essentially taken into account the robustness properties of the system and can justifiably conclude that the system is unsafe in practice.

5.2.2 DPLL⟨ICP⟩

Interval Constraint Propagation (ICP) [13] is a constraint solving algorithm that finds solutions of real constraints using a "branch-and-prune" method, combining interval arithmetic and constraint propagation. The idea is to use interval extensions of functions to "prune" out sets of points that are not in the solution set, and "branch" on intervals when such pruning can not be done, until a small enough box that may contain a solution is found. In a DPLL⟨T⟩ framework, ICP can be used as the theory solver that checks the consistency of a set of theory atoms. In dReal4, we used PicoSAT [15] to implement the DPLL⟨T⟩ framework and integrate IBEX-lib [112] which performs interval pruning operations. We now describe the design of the interface.

Check and Assert For incomplete checks in the assert function, we use the pruning operator provided in ICP to contract the interval assignments on all the variables, by eliminating the boxes in the domain that do not contain any solutions. At complete

checks, we perform both pruning and branching, and look for one interval solution of the system. That is, we prune and branch on the interval assignment of all variables, and stop when either we have obtained an interval vector that is smaller than the preset error bound, or when we have traversed all the possible branching on the interval assignments.

Backtracking and Learning We maintain a stack of assignments on the variables, which are mappings from variables to unions of intervals. When we reach a conflict, we backtrack to the previous environment in the pushed stack. We also collect all the constraints that have appeared in the pruning process leading to the conflict. We then turn this subset of constraints into a learned clause and add it to the original formula.

Witness for δ-satisfiability When the answer is δ-sat on $\varphi(\vec{x})$, we provide a solution $\vec{a} \in \mathbb{R}^n$, such that $\varphi^\delta(\vec{a})$ is a ground formula that can be easily checked to be true. It is important to note that the solution witnesses δ-satisfiability, instead of standard satisfiability of the original formula. While the latter problem is undecidable, *any* point in the interval assignment returned by ICP can witness the satisfiability of φ^δ when the intervals are smaller than an appropriate error bound.

Proofs of Unsatisfiability When the answer is unsat, we produce a proof tree that can be verified to establish the validity of the negation of the formula, i.e., $\forall \vec{x} \neg \varphi(\vec{x})$. We devised a simple first-order natural deduction system, and transform the computation trace of the solving process into a proof tree. We then use interval arithmetic and simple rules to check the correctness of the proof tree. The proof check procedure recursively divide the problem into subproblems with smaller domains. More details can be found in [53].

5.2.3 Parallel ICP

The ICP algorithm serves as the primary computational bottleneck in nonlinear SMT solving within dReal. Given the widespread availability of multi-core processors, exploiting parallelism in ICP offers a promising approach to mitigating this performance limitation. Our goal is to develop a multi-core implementation of ICP in dReal that achieves a linear speed-up proportional to the number of cores employed. By pursuing this parallelization, we aim to enhance the efficiency and scalability of dReal, thereby enabling the solution of larger and more complex nonlinear SMT problems.

Before exploring the design of the parallel ICP, we review the sequential ICP algorithm (Algorithm 4.1). The algorithm maintains a stack of interval boxes representing the search space. It repeatedly pops the top box, applies pruning operations until a fixed-point is obtained. Then it partitions the current box into two sub-boxes, and pushes them back onto the stack. This procedure continues until either a delta-sat solution is found or the stack is emptied.

The parallel ICP architecture employs a global stack to manage the search space, represented by interval boxes. Within this architecture, each thread operates as a worker. The worker retrieves a box from the global stack and successively prunes it until a fixed-point is reached. Three outcomes are possible in this process. First, if a box becomes empty, the worker returns to the global stack to obtain another box. Second, if a box is non-empty and satisfies the delta-sat criteria, the worker signals other workers to terminate and returns the identified delta-sat solution. Lastly, if a box fails to meet the delta-sat criteria, the worker performs a branching operation, producing two new boxes that are subsequently added to the global stack. Through the distribution of workload among multiple workers, the parallel ICP strives to attain a linear speed-up proportional to the number of cores employed.

Figure 5.1 depicts the architecture of the parallel ICP implementation, highlighting the following challenges encountered and corresponding solutions developed to address them.

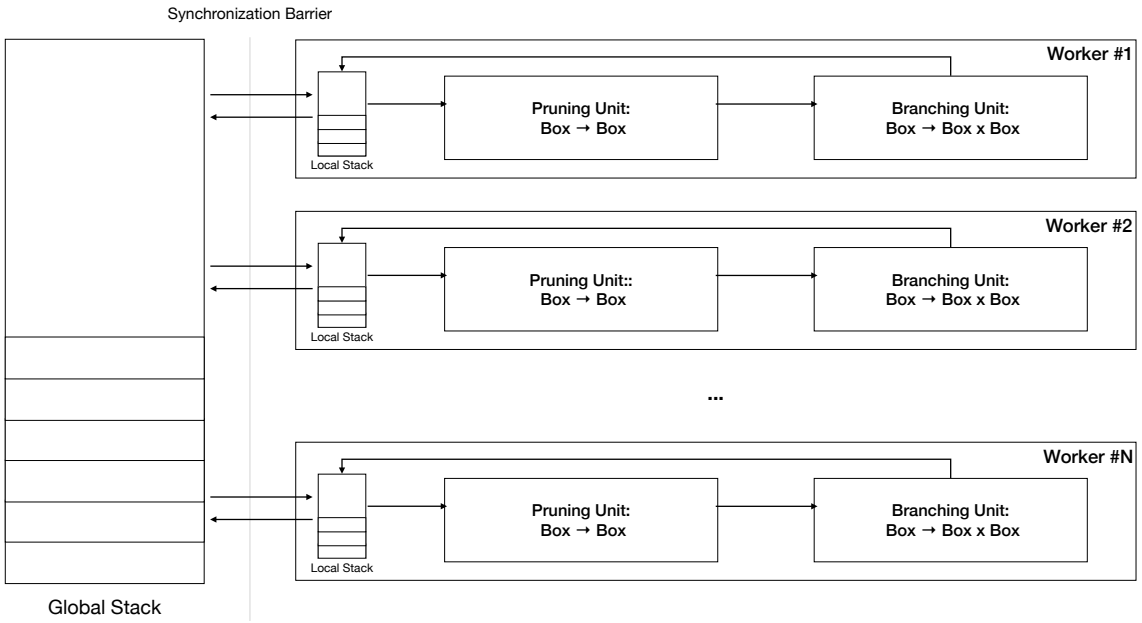


Figure 5.1: The architecture of the parallel ICP implementation: It employs a global stack to manage the search space . Each thread operates as a worker, which retrieves and processes boxes from the global stack. To minimize synchronization overhead, the architecture allocates a local stack to each worker. Additionally, a job spilling mechanism is introduced to enhance load balancing and maintain high utilization across all threads by redistributing boxes from local stacks to the global stack when necessary.

- Synchronization cost: When N threads attempt to access the global stack simultaneously, synchronization overhead arises. To address this issue, we employ two strategies. First, we utilize the Treiber stack [26], a scalable lock-free data structure that exploits fine-grained concurrency primitives to facilitate efficient parallel access. Second, we allocate a local stack to each worker, thereby minimizing communication and further reducing synchronization overhead.

- Low utilization / Load unbalancing: In some cases, certain threads may not have work to perform, resulting in decreased overall utilization and throughput. To counteract this problem, we introduce a job spilling mechanism.

This approach involves transferring boxes from a worker's local stack to the global stack when the local stack contains an excessive number of boxes. By redistributing the workload in this manner, we enhance load balancing and promote higher utilization across all threads.

5.3 Usage

5.3.1 Input Format

We accept formulas in the standard SMT-LIB format [11] with extensions. In addition to nonlinear arithmetic (polynomials), we allow transcendental functions such as sin, tan, arcsin, arctan, exp, log, pow, sinh. More nonlinear functions can be added when needed, by providing the corresponding numerical evaluation algorithms. Floating-point numbers are allowed as constants in the formula.

Bound information on variables can be declared using a list of simple atomic formulas. For instance "`(assert (< 0 x))`", which sets $x \in (0, +\infty)$ at parsing time. Also, the user can set the precision by writing "`(set-info :precision 0.0001)`." The default precision is 10^{-3}, and can be set through command line.

Example 2. *The following is an example input file. It is taken from the Flyspeck project [63]. (Filename* `flyspeck/172.smt2`*. Flyspeck* `ID (6096597438b)`*)*

```
(set-logic QF_NRA)
(set-info :precision 0.001)
(declare-fun x () Real)
(assert (<= 3.0 x))
(assert (<= x 64.0))
(assert
    (not
        (> (- (* 2.0 3.14159265)
              (* 2.0 (* x (arcsin (* (cos 0.797)
                                     (sin (/ 3.14159265 x)))))))
           (+ (- 0.591 (* 0.0331 x))
```

```
12                (+ (* 0.506 (/ (- 1.26 1.0) (- 1.26 1.0))) 1.0)))))
13  (check-sat)
14  (exit)
```

5.3.2 Command-line Options

To use dReal, you need to install dReal on your system. You can download the latest version of dReal from the official website (`https://dreal.github.io`). Follow the installation instructions for your operating system.

dReal provides the following command-line options:

- `-j, --jobs <N>`: Number of threads to use.
- `--precision <N>`: Precision (δ) value (default = 0.001).
- `--local-optimization`: Enable the local-optimization algorithm for exists-forall problems if specified.
- `--model`: Produce models if delta-sat.
- `--random-seed <N>`: Set a seed for the random number generator.
- `--sat-default-phase <N>`: Set the default initial phase for SAT solver: 0 = false, 1 = true, 2 = Jeroslow-Wang (default), 3 = random initial phase.
- `--smtlib2-compliant`: Strictly follow the SMT-LIB2 standard.
- `--verbose <ARG>`: Verbosity level. One of the following: trace, debug, info, warning, error (default), critical, off.

5.4 Experiments

Flyspeck Benchmark We evaluated the performance of dReal on a set of nonlinear SMT benchmarks. Specifically, we extracted 903 SMT2 instances from the Flyspeck

project [63], which aims to formally prove the Kepler's conjecture. The following is a typical formula:

$$\forall \vec{x} \in [2, 2.51]^6. \Big(- \frac{\pi - 4 \arctan \frac{\sqrt{2}}{5}}{12\sqrt{2}} \sqrt{\Delta(\vec{x})} + \frac{2}{3} \sum_{i=0}^{3} \arctan \frac{\sqrt{\Delta(\vec{x})}}{a_i(\vec{x})} \leq -\frac{\pi}{3} + 4 \arctan \frac{\sqrt{2}}{5} \Big)$$

where $a_i(\vec{x})$ are quadratic and $\Delta(\vec{x})$ is the determinant of a nonlinear matrix.

Experimental Environment We conducted our experiments on a system comprising an AMD EPYC 7R32 CPU with 16 cores, running at 2.8 GHz, and 32 GB of RAM. The operating system used was Ubuntu 20.04 LTS, and we set the value of delta to 1e−10.

Experimental Results First, we used a single thread per problem and solved 489 out of the 903 instances within a timeout of 5 minutes. Out of these, dReal returned unsat for 196 instances and delta-sat for 293 instances. Next, we increased the number of threads from 1 to 2, 4, and 8 to evaluate the performance of dReal in a multi-threaded environment. With eight threads per problem, we were able to solve 617 instances within a timeout of 5 minutes. Among them, dReal returned unsat for 263 instances and delta-sat for 354 instances.

In Table 5.1, we present the results of the top 30 instances that we were able to solve using a single thread within a 5 minute timeout. The total time taken to solve these 30 problems using a single thread was 3,175.303 seconds. To evaluate the effectiveness of multi-threading, we ran the same set of problems using eight threads. The total running time was reduced to 505.607 seconds, resulting in a speed-up of 6.28 times compared to the single-threaded approach. These results demonstrate the performance gains that can be achieved with multi-threading in dReal.

Instance	Result	Time (sec)				Speed-up (j1 /j8) (times)
		-j1	-j2	-j4	-j8	
158	unsat	292.048	130.565	77.472	30.864	9.46
417	unsat	284.103	143.560	69.306	36.623	7.76
100	unsat	249.381	121.685	43.164	16.395	15.21
392	unsat	245.812	146.010	92.800	19.329	12.72
140	unsat	244.160	160.260	76.808	49.811	4.90
64	unsat	234.455	146.552	83.528	34.018	6.89
316	unsat	213.503	153.007	84.746	51.373	4.16
352	unsat	181.891	94.480	83.835	60.225	3.02
165	unsat	173.504	117.281	37.299	36.939	4.70
399	unsat	100.879	51.974	24.173	14.092	7.16
147	unsat	99.985	59.440	25.280	13.086	7.64
144	unsat	93.167	49.881	22.594	12.439	7.49
396	unsat	92.095	52.664	22.901	12.031	7.65
403	unsat	89.394	52.652	22.570	15.579	5.74
151	unsat	89.118	43.382	22.588	11.166	7.98
150	unsat	81.057	39.977	19.976	13.557	5.98
402	unsat	78.866	41.272	21.007	10.812	7.29
398	unsat	73.601	37.632	17.345	10.987	6.70
405	unsat	72.486	40.802	18.048	13.870	5.23
146	unsat	72.301	39.227	19.849	10.230	7.07
877	δ-sat	68.098	40.141	22.308	10.518	6.47
894	δ-sat	9.370	6.546	6.679	5.455	1.72
892	δ-sat	9.315	9.389	7.003	6.716	1.39
766	δ-sat	6.780	9.261	3.757	1.490	4.55
767	δ-sat	5.093	3.866	1.063	1.365	3.73
754	δ-sat	3.464	2.355	1.422	1.342	2.58
804	δ-sat	3.267	1.849	1.491	1.310	2.49
794	δ-sat	3.086	2.851	1.327	1.375	2.24
797	δ-sat	2.957	3.701	0.969	1.298	2.28
753	δ-sat	2.067	2.186	1.050	1.312	1.58
Total		3175.303	1804.448	932.358	505.607	6.28

Table 5.1: Experimental results for Flyspeck benchmarks: The top 30 instances with the longest single-thread solving times within a 5-minute timeout are presented. The total time taken for these instances with a single thread was 3,175.303 seconds. To assess multi-threading effectiveness, the same problems were run using two, four, and eight threads. With eight threads, the total running time decreased to 505.607 seconds, achieving a 6.28-fold speed-up.

5.5 Conclusion

We have presented dReal, an open-source tool for solving SMT problems over the reals. dReal is capable of handling various nonlinear real functions, including polynomials, trigonometric functions, and exponential functions. The tool is based on the δ-complete decision procedures framework and returns either unsat or δ-sat on input formulas, where δ is a user-specified numerical error bound.

Our experimental results demonstrate that dReal is effective in solving nonlinear SMT problems, even in multi-threaded environments. By leveraging the power of multi-threading, we were able to achieve significant speed-ups in solving a nonlinear benchmarks.

Chapter 6

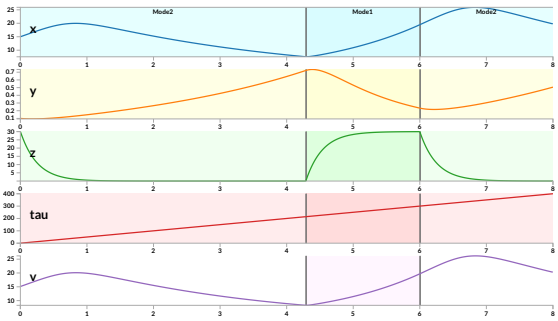
dReach: A Delta-Reachability Analysis Tool for Hybrid Systems

6.1 Introduction

dReach is a bounded reachability analysis tool for hybrid systems. It encodes bounded reachability problems of hybrid systems as first-order formulas over the real numbers, and solves them using δ-decision procedures in the SMT solver

$$\frac{dx}{dt} = \left(\alpha_x\left(k_1 + \frac{(1-k_1)z}{z+k_2}\right) - \beta_x\left(k_3 + \frac{(1-k_3)z}{z+k_4}\right) - m_1\left(1 - \frac{z}{z_0}\right)\right)x$$
$$\frac{dy}{dt} = m_1\left(1 - \frac{z}{z_0}\right)x + \left(\alpha_y\left(1 - d\,\frac{z}{z_0}\right) - \beta_y\right)y$$
$$\frac{dz}{dt} = \frac{z_0 - z}{\tau}$$
$$\frac{dv}{dt} = \left(\alpha_x\left(k_1 + \frac{(1-k_1)z}{z+k_2}\right) - \beta_x\left(k_3 + \frac{(1-k_3)z}{z+k_4}\right) - m_1\left(1 - \frac{z}{z_0}\right)\right)x$$
$$+ m_1\left(1 - \frac{z}{z_0}\right)x + \left(\alpha_y\left(1 - d\,\frac{z}{z_0}\right) - \beta_y\right)y$$

(a) An example of nonlinear hybrid system model: off-treatment mode of the prostate cancer treatment model [87]



(b) Visualization of a generated counterexample. Change in the shade of colors represents discrete mode changes.

Figure 6.1: An example of nonlinear dynamics and counterexample-generation.

dReal [52]. dReach is able to handle a wide range of highly nonlinear hybrid systems [87, 55, 76]. Figure 6.1 highlights some of its features: on the left is an example of some nonlinear dynamics that dReach can handle, and on the right a visualized counterexample generated by dReach on this model.

It is well-known that the standard bounded reachability problems for simple hybrid systems are already highly undecidable [6]. Instead, we work in the framework of δ-reachability of hybrid systems [51]. Here δ is an arbitrary positive rational number, provided by the user to specify the bound on numerical errors that can be tolerated in the analysis. For a hybrid system H and an unsafe region unsafe (both encoded as logic formulas), the δ-reachability problem asks for one of the following answers:

- safe: H cannot reach unsafe.
- δ-unsafe: $H^{-\delta}$ can reach unsafe$^{-\delta}$.

Here, $H^{-\delta}$ and unsafe$^{-\delta}$ encode (δ-bounded) over-approximations of H and unsafe, defined explicitly as their syntactic variants as defined in Definition 5. It is important to note that the definition makes the answers no weaker than standard reachability: When safe is the answer, we know for certain that H does not reach the unsafe region (no δ is involved); when δ-unsafe is the answer, we know that there exists some δ-bounded perturbation of the system that can render it unsafe. Since δ can be chosen to be very small, δ-unsafe answers in fact discover robustness problem in the system, which should be regarded as unsafe indeed. We have proved that bounded δ-reachability is decidable for a wide range of nonlinear hybrid systems, even with reasonable complexity bounds [51]. This framework provides the formal correctness guarantees of dReach.

Apart from solving δ-reachability, the following key features of dReach distinguish it from other existing tools in this domain [44, 46, 4, 45, 65, 27, 30].

1. Expressiveness. dReach allows the user to describe hybrid systems using first-order logic formulas over real numbers with a wide range of nonlinear

functions. This allows the user to specify the continuous flows using highly nonlinear differential equations, and the jump and reset conditions with complex Boolean combinations of nonlinear constraints. dReach also faithfully translates mode invariants into $\exists\forall$ logic formulas, which can be directly solved under certain restrictions on the invariants.

2. Property-guided search. dReach maintains logical encodings (the same approach as [30]), whose size is linear in the size of the inputs, of the reachable states of a hybrid system [51]. The tool searches for concrete counterexamples to falsify the reachability properties, instead of over-approximating the full reachable states. This avoids the usual state explosion problem in reachable set computation, because the full set of states does not need to be explicitly stored. This change is analogous to the difference between SAT-based model checking and BDD-based symbolic model checking.

3. Tight integration of symbolic reasoning and numerical solving. dReach delegates the reasoning on discrete mode changes to SAT solvers, and uses numerical constraint solving to handle nonlinear dynamics. As a result, it can combine the full power of both symbolic reasoning and numerical analysis algorithms. In particular, all existing tools for reachable set computation can be easily plugged-in as engines for solving the continuous part of the dynamics, while logic reasoning tools can overcome the difficulty in handling complex mode transitions.

The chapter is structured as follows. We describe the system architecture in Section 6.3, and give some details about the logical encoding in the tool in Section 6.4. We then explain the input format and usage in Section 6.5. In Section 6.6, we present three case studies to demonstrate the effectiveness of dReach.

6.2 Bounded delta-reachability

Let $H = \langle X, Q, \mathsf{flow}, \mathsf{jump}, \mathsf{inv}, \mathsf{init}\rangle$ be a hybrid system as standardly defined. We use first-order formulae over the real numbers to represent H, by writing

$$H = \langle X, Q, \varphi_{\mathsf{flow}}, \varphi_{\mathsf{jump}}, \varphi_{\mathsf{inv}}, \varphi_{\mathsf{init}}\rangle$$

where $\varphi_{\mathsf{flow}}, \varphi_{\mathsf{jump}}, \varphi_{\mathsf{inv}}$ and φ_{init} are logic formulae that define the corresponding predicates in the standard definition. Now, let $\delta \in \mathbb{Q}^+$ be a chosen error bound, we define the δ-perturbation of H to be

$$H^{-\delta} = \langle X, Q, \varphi^{-\delta}_{\mathsf{flow}}, \varphi^{-\delta}_{\mathsf{jump}}, \varphi^{-\delta}_{\mathsf{inv}}, \varphi^{-\delta}_{\mathsf{init}}\rangle.$$

Here, $\varphi^{-\delta}$ is a syntactic variant of φ which relaxes the numerical terms in φ up to an error bound δ. The notion was formally defined in Definition 5. We now define the bounded delta-reachability problem that dReach solves.

Let $n \in \mathbb{N}$ be a bound and $T \in \mathbb{R}^+$ be an upper bound of time duration. We write unsafe to denote a subset of the state space of H defined by a first-order formula. The bounded delta-reachability problem asks for one of the following answers:

- H cannot reach unsafe in n steps within time T.
- $H^{-\delta}$ can reach unsafe$^{-\delta}$ in n steps within time T.

Note that these answers are not weaker than the precise ones. When safe is the answer, we know for certain that H does not reach the unsafe region; when δ-unsafe is the answer, there exists some δ-bounded perturbation in the system that *would* render it unsafe. Note that the error-bound δ can be chosen to be arbitrarily small, so that the δ-unsafe answer discovers robustness problem in the system, which should be regarded as unsafe indeed.

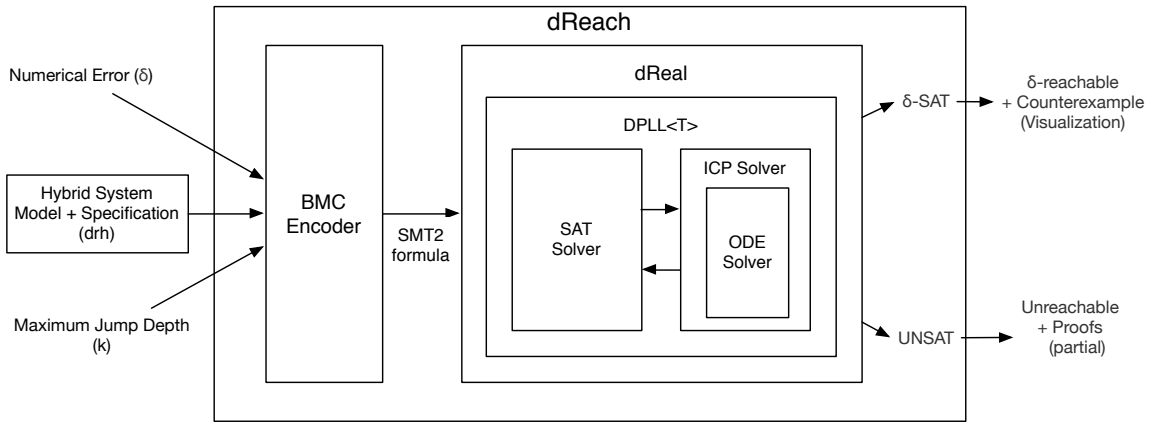


Figure 6.2: Architecture of dReach: It consists of an bounded model-checking module and an SMT solver, dReal. In the first phase, the Encoder module translates an input hybrid system into a logic formula. In the second phase, an SMT solver, dReal, solves the encoded delta-reachability problem using a solving framework that combines DPLL⟨T⟩, Interval Constraint Propagation, and reliable (interval-based) numerical integration.

6.3 System Description

The system architecture of dReach is given in Figure 6.2. We ask the user to provide an input file and two parameter — (i) a bound on the number of mode changes and (ii) a numerical error bound $\delta \in \mathbb{R}^+$.

From these inputs, dReach generates a logical encoding that involves existential quantification and universal quantification on the time variables. The logical encoding is compact, always linear in the size of the inputs. The tool then makes iterative calls to the underlying solver dReal [52] to decide the reachability properties. When the answer is δ-reachable, dReach generates a counterexample and its visualization. When the answer is unreachable, no numerical error is involved and a (partial, for now) logical proof of unsatisfiability can be provided [53].

Input Format (drh format) The drh input format for describing hybrid systems and reachability properties consists of five sections: macro definitions, variable declarations, mode definitions, and initial condition, and goals. Its formal syntax is

defined by the following grammar.

$$
\begin{aligned}
\textit{drh} &:= \textit{macro\_def}^{*}\ \textit{variable\_decl}^{+}\ \textit{mode\_def}^{+}\ \textit{initial\_cond}\ \textit{goal}^{+} \\
\textit{macro\_decl} &:= \texttt{\#define}\ \textit{var}\ (\textit{expr} \mid \textit{formula}) \\
\textit{variable\_decl} &:= \texttt{[}l\texttt{,}\ u\texttt{]}\ \textit{var}\texttt{;} \\
\textit{mode\_def} &:= \texttt{\{mode}\ \textit{id}\texttt{;}\quad \texttt{invt :}\ (\textit{formula}\texttt{;})^{+}\quad \texttt{flow :}\ \textit{ode}^{+}\quad \texttt{jump :}\ \textit{jump}^{+}\texttt{\}} \\
\textit{ode} &:= \texttt{d/dt[}x\texttt{]=}\textit{expr} \\
\textit{jump} &:= \textit{formula}\ \texttt{==>}\ \texttt{@}n\ \textit{formula} \\
\textit{initial\_cond} &:= \texttt{@}\textit{mode\_id}\ \textit{formula}\texttt{;} \\
\textit{goal} &:= \texttt{@}\textit{mode\_id}\ \textit{formula}\texttt{;}
\end{aligned}
$$

Note that we use the standard definitions for *formula* and *expr* here. We focus on intuitive explanations here.

- In macro definitions, we allows users to define macros in C preprocessor style which can be used in the following sections. Macro expansions occur before the other parts are processed.

- A variable declaration specifies a real variable and its domain in a real interval. dReach requires special declaration for *time* variable, to specify the upper-bound of time duration.

- A mode definition consists of mode ID, mode invariant, flow, and jump. *id* is a unique positive integer assigned to a mode. An invariant is a conjunction of logic formulae which must always hold in a mode. A flow describes the continuous dynamics of a mode by providing a system of ODEs. The first formula of *jump* is interpreted as a guard, a logic formula specifying a condition to make a transition. Note that this allows a transition but does not force it. The second argument of *jump*, *n* denotes the target mode-id. The

last one is *reset*, a logic formula connecting the old and new values for the transition.

- *initial-condition* specifies the initial mode of a hybrid system and its initial configuration. *goal* shares the same syntactic structure of *initial-condition*.

6.4 Logical Encoding of Reachability

The details of our encoding scheme is given in [51]. Here we focus on explaining how differential equations and the universal quantifications generated by mode invariants are encoded, as an extension of the SMT-LIB [11] standard. Although such formulas are automatically generated by dReach from the hybrid system description, the explanation below can be helpful for understanding the inner mechanism of our solver.

Encoding ODE Constraints In each mode of a hybrid system, we need to specify continuous flows defined by systems of ordinary differential equations. We extend SMT-LIB with a command `define-ode` to define such systems. For instance, we use `define-ode` as follows to assign a name flow_1 to a group of ODE, $\frac{\mathrm{d}x}{\mathrm{d}t} = v$ and $\frac{\mathrm{d}v}{\mathrm{d}t} = -x^2$.

```
(define-ode flow1 ((= d/dt[x] v) (= d/dt[v] (- 0 (^ x 2)))))
```

We then allow integration terms in the formula. We view the solution of system of differential equations as a constraint between the initial-state variables, time duration, and the end-state variables. We can then write

```
(= [x_t_1 ... x_t_n] (integral 0 t [x_0_1 ... x_0_n] flow_i)),
```

to represent $\vec{x} = \vec{x}_0 + \int_0^t \mathit{flow}_i(\vec{x}(s))\,\mathrm{d}s$. Note that we do not need to explicitly mention $\vec{x}(s)$ as a function in the encoding, which can be inferred by the solver.

Universal Quantification for Mode Invariant Constraints To encode mode invariants in hybrid systems, we need $\exists\forall^t$-formulas defined in Section 3.3.2. This formula is a restricted form of $\exists\forall$ formula where the universal quantifications are limited to the time variables. In SMT2, we introduce a new keyword `forall_t` to encode $\exists\forall^t$ formulas. Given a time bound $[0, time_i]$, mode invariant f at mode n is encoded into `(forall_t n [0 time_i] f)`.

Note that dReal only allows invariants to be expressed as a conjunction of comparisons between a variable and a constant ($f = \bigwedge_i v_i \sim c_i$).

6.5 Using dReach

6.5.1 How to Install

dReach is an open source project under GPL-3 license. We bundled dReal and dReach together and host them at `http://dreal.github.io`. The BMC encoder module is written in OCaml and uses Oasis and OCaml Batteries library. At the release page[1], we host pre-compiled static-binaries for Linux and OS X, which do not require any compilation to use dReach in those platforms.

6.5.2 Command-line Options

dReach follows the standard Unix command-line usage:

```
dReach <options> <drh file>
```

It has the following options:

- If `-k <N>` is used, set the unrolling bound k as N (Default: 3). It also provides `-u <N>` and `-l <N>` options to specify upper- and lower-bounds of unrolling bound.

[1] `http://dreal.github.io/download`

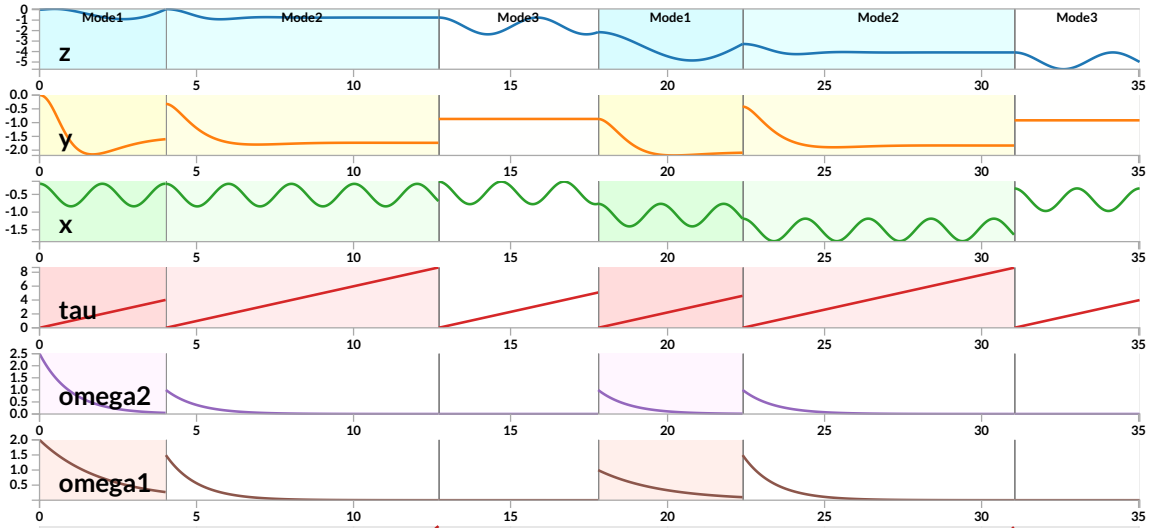


Figure 6.3: Visualization of δ-reachable trajectory for a 3-mode ocillator model.

- If `-precision` <p> is used, use precision p (Default: 0.001).
- If `-visualize` is set, dReach generates extra visualization data.

We have a web-based visualization toolkit[2] which processes the generated visualization data and shows the counterexample trajectory. It provides a way to navigate and zoom-in/out trajectories which helps understand and debug the target hybrid system better. Figure 6.3 is a screenshot of the visualization for a cardiac-cell model.

6.6 Case Studies

In this section, we aim to demonstrate the effectiveness of the dReach tool through three distinct case studies.

[2]The detailed instructions are available at `https://github.com/dreal/dreal2/blob/master/doc/ode-visualization.md`.

1. The first study (Section 6.6.1) presents a simple model of a bouncing ball with air resistance, which serves as a fundamental demonstration of the utilization of dReach.

2. The second case study (Section 6.6.2) showcases a model of atrial fibrillation. We formulate a hybrid system model of cardiac cells and translate the question of "Can we find a set of initial values/parameters that result in a loss of excitability in a cardiac cell?" into a reachability problem. The dReach tool is then applied to answer this query.

3. In the third case study (Section 6.6.3), we examine a personalized prostate cancer treatment model using a parameterized hybrid system. Our goal is to determine the parameters for a personalized treatment schedule that will prevent cancer recurrence within a specified number of days. We encode this query as a bounded model checking problem and demonstrate the use of dReach to solve it.

6.6.1 Inelastic Bouncing Ball with Air Resistance

Consider the following standard bouncing-ball example with the following assumptions:

1. Air Friction acts on the ball, which is proportional to $D \cdot v^2$.

2. The ball is inelastic. That is, whenever it hits the wall, it loses some of its kinetic energy ($v' = K \cdot v$).

We formally define this as a hybrid system as follows:

- $X = \mathbb{R}^2$ and $Q = \{q_d, q_u\}$. We use q_d the falling mode and q_u to represent bounce-back mode.

- $\mathsf{flow} = \{\mathsf{flow}_{q_d}(x_0, v_0, x_t, v_t, t), \mathsf{flow}_{q_u}(x_0, v_0, x_t, v_t, t)\}$. We use x to denote the height of the ball and v its velocity. Instead of using time derivatives, we can directly write the flows as integrals over time, using $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas:
  - $\mathsf{flow}_{q_d}(x_0, v_0, x_t, v_t, t)$ defines the dynamics in the falling phase:
$$\left(x_t = x_0 + \int_0^t v(s)\, ds\right) \wedge \left(v_t = v_0 + \int_0^t -g - D \cdot v(s)^2\, ds\right).$$
  - $\mathsf{flow}_{q_u}(x_0, v_0, x_t, v_t, t)$ defines the dynamics in the bounce-back phase:
$$\left(x_t = x_0 + \int_0^t v(s)\, ds\right) \wedge \left(v_t = v_0 + \int_0^t -g + D \cdot v(s)^2\, ds\right).$$

  where D is a constant. Again, note that the integration terms define Type 2 computable functions.

- $\mathsf{jump} = \{\mathsf{jump}_{q_u \to q_d}(x, v, x', v'), \mathsf{jump}_{q_d \to q_u}(x, v, x', v')\}$ where
  - $\mathsf{jump}_{q_u \to q_d}(x, v, x', v')$ is $(v = 0 \wedge x' = x \wedge v' = v)$.
  - $\mathsf{jump}_{q_d \to q_u}(x, v, x', v')$ is $(x = 0 \wedge v' = -k \cdot v \wedge x' = x)$, for some constant $0 < k < 1$.

- $\mathsf{init}_{q_d} : (x \geq 5 \wedge v = 0)$ and $\mathsf{init}_{q_u} : \bot$.

- $\mathsf{inv}_{q_d} : (x \geq 0 \wedge v \leq 0)$ and $\mathsf{inv}_{q_u} : (x \geq 0 \wedge v \geq 0)$.

In mode q_d, the ball is subject to gravity and air resistance, which cause the ball to fall towards the ground. In mode q_u, the ball bounces back up, transitioning from a falling to a rising state. This transition is a discrete event, triggered when the ball reaches the ground. The behavior of the hybrid system is determined by the interaction between the continuous and discrete dynamics. For example, air resistance affects the velocity of the ball as it falls and affects the height of the bounce. The inelasticity of the ball, in turn, affects the energy dissipation during the bounce, leading to a reduction in the height of subsequent bounces.

```
1  #define D 0.1
2  #define K 0.9
3  #define g 9.8
4
5  [0, 50] x;
6  [-50, 50] v;
7  [0, 30] time;
8
9  { mode 1;
10
11   invt:
12         (v <= 0);
13         (x >= 0);
14   flow:
15         d/dt[x] = v;
16         d/dt[v] = -g + (D * v ^ 2);
17   jump:
18         (x = 0) ==> @2 (and (x' = x) (v' = - K * v));
19 }
20 {
21   mode 2;
22   invt:
23         (v >= 0);
24         (x >= 0);
25   flow:
26         d/dt[x] = v;
27         d/dt[v] = -g + (- D * v ^ 2);
28   jump:
29         (v = 0) ==> @1 (and (x' = x) (v' = v));
30 }
31 init:
32 @1         (and (x >= 5) (v = 0));
33
34 goal:
35 @1         (and (x >= 0.45));
36
```

Figure 6.4: An example of drh format: Inelastic bouncing ball with air resistance. Lines 1 – 3 define a drag coefficient $D = 0.1$, an elastic coefficient $K = 0.9$, and the gravity constant $g = 9.8$. Lines 5 – 7 declares variables x, v, $time$ with their domains ($0 \leq x \leq 50$, $-50 \leq v \leq 50$, $0 \leq time \leq 30$). At lines 9 – 19 and 20 – 30, we define two modes — the falling and the rising modes respectively. At line 32, we specify that the hybrid system starts at mode 1 (@1) with initial condition satisfying $x \geq 5 \wedge v = 0$. At line 35, we ask whether there exists a trajectory ending at mode 1 (@1) while the height of the ball is higher than 0.45.

Input Format (drh) Figure 6.4 shows how to describe this hybrid system in drh format. In the first part, we declare variables (x, v) and constants ($g = 9.8$, $drag = 0.1$, $elasticity = 0.9$), and special variable $time$ which is a bounded variable representing time in each mode. Then, we describe each mode of this hybrid system. Mode 1 represents the status where a ball is falling down toward a floor, while mode 2 describes a ball bouncing back from a floor. `invt` describes the invariant of each mode. In this example, we have simple invariants — velocity of the ball in mode 1 should be negative since it's falling and the velocity of a ball should be positive in mode 2 because it's bouncing back from the floor. Dynamics of variables in each mode is described by `flow`, which is a set of differential equations. Here, we have simple dynamics: $\dot{x} = v$ and $\dot{v} = -g \pm (D^2 \cdot v)$. `jump` describes the conditions to switch mode, the destination mode, and changes of values. Note that when it jumps from mode 1 to mode 2, the velocity of the ball reduces due to the inelasticity, $v' = K \cdot v$. In the end, we describe the initial condition, and the goal which we want to check its satisfiability. In this example, we start with $x \geq 5$ and $v = 0$ and check whether it is possible to reach the mode 1 while $x \geq 0.45$.

Running dReach dReach takes in a hybrid system description (.drh) and unrolling bound k, and performs bounded model-checking.

```
$ dReach -k 10 -l 10 bouncing_ball.drh --visualize --precision=0.1
```

The command-line argument `-k 10` specifies the upper bound on the unrolling depth of bounded model checking, and the optional `-l 10` specifies the lower bound. The options `-visualize` and `-precision=0.001` will be passed to dReal. The first option `-visualize` enables dReal to store additional information to visualize the witness of δ-sat result. The second option `-precision` specifies the value of numerical perturbation δ we allow. Running the above command, it first generates an SMT2 file. Figure 6.5 shows the SMT2 encoding of the bounded reachability problem of a bouncing ball example (when $k = 10$).

```
(set-logic QF_NRA_ODE)
(declare-fun x () Real)
(declare-fun v () Real)
(declare-fun x_0_0 () Real)
(declare-fun x_0_t () Real)
...
(declare-fun x_10_0 () Real)
(declare-fun x_10_t () Real)
(declare-fun v_0_0 () Real)
(declare-fun v_0_t () Real)
...
(declare-fun v_10_0 () Real)
(declare-fun v_10_t () Real)
(declare-fun time_0 () Real)
...
(declare-fun time_10 () Real)
(declare-fun mode_0 () Real)
...
(declare-fun mode_10 () Real)
(define-ode flow_1 ((= d/dt[x] v)
                    (= d/dt[v] (+ (- 0.000000 9.800000) (* -0.450000 (^ v 1.000000))))))
(define-ode flow_2 ((= d/dt[x] v) (= d/dt[v]
                    (+ (- 0.000000 9.800000) (* -0.450000 (^ v 1.000000))))))
(assert (<= 0.000000 x_0_0))
(assert (<= x_0_0 15.000000))
...
(assert (<= -18.000000 v_10_t))
(assert (<= v_10_t 18.000000))
(assert (<= 0.000000 time_0))
(assert (<= time_0 3.000000))
...
(assert (<= 0.000000 time_10))
(assert (<= time_10 3.000000))
...

(assert (and (= v_0_0 0.000000) (>= x_0_0 5.000000)
             (= mode_0 1.000000)
             (= [x_0_t v_0_t] (integral 0. time_0
             [x_0_0 v_0_0] flow_1))
             (= mode_0 1.000000)
             (forall_t 1.000000 [0.000000 time_0] (<= v_0_t 0.000000))
             (<= v_0_t 0.000000)
             ...
             (forall_t 1.000000 [0.000000 time_10] (>= x_10_t 0.000000))
             (>= x_10_t 0.000000)
             (>= x_10_0 0.000000)
             (= mode_10 1.000000)
             (>= x_10_t 0.450000)))
(check-sat)
(exit)
```

Figure 6.5: SMT2 encoding of the problem of an inelastic bouncing ball with air resistance.

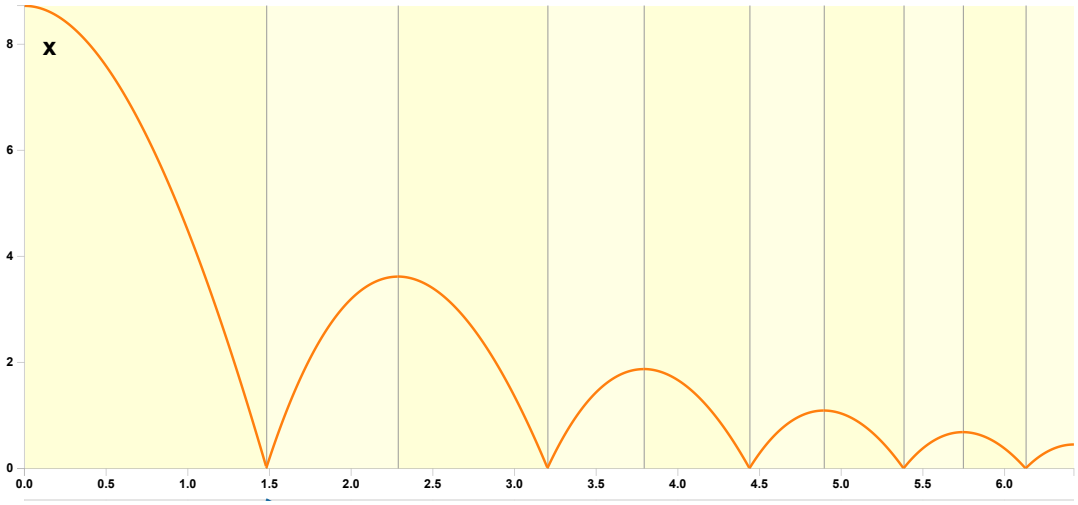


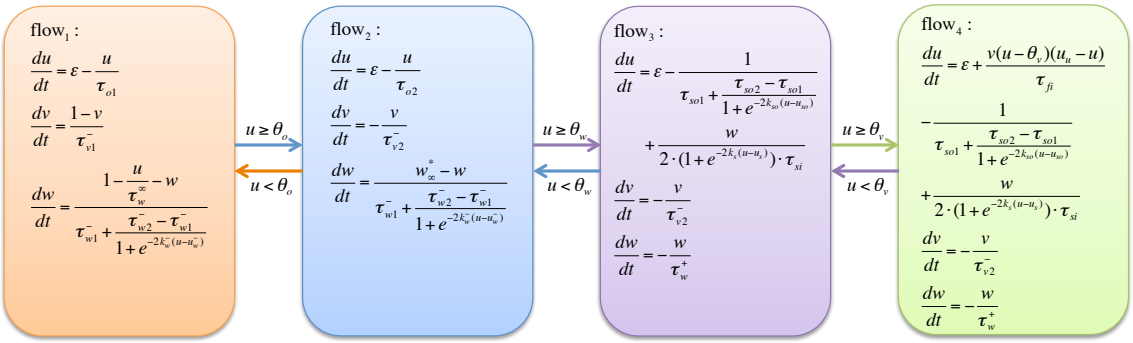
Figure 6.6: Visualization of δ-reachable trajectory for a bouncing-ball model.

dReach returns δ-sat and generates the visualization of its reachable trace. Figure 6.6 shows that if the ball starts at a height between $[8.720703125, 8.73046875]$, it can reach the height 0.45 after bouncing five times. Note that we only specify the initial condition $x \geq 5$ and dReal finds out the satisfying solution $[8.720703125, 8.73046875]$ by solving the encoded constraints.

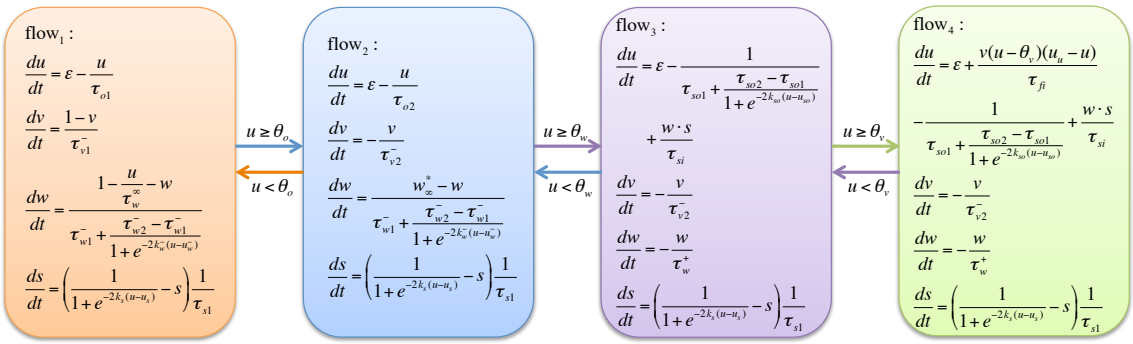
6.6.2 delta-Reachability Analysis of Cardiac Cell Models

Cardiac Cell Hybrid Models The proper functioning of the heart's rhythm relies on the coordinated electrical activity of the cardiac muscle cells in both the atria and ventricles. This electrical behavior is controlled by the opening and closing of ion channel gates on the cell membrane. Disruptions in the ionic channel functions can lead to a loss of cell excitability, which can further lead to cardiac abnormalities, such as ventricular tachycardia or fibrillation. To comprehend the underlying mechanisms of cardiac disorders, recent development of hybrid automata models such as the Fenton-Karma (FK) model [42] and the Bueno-Cherry-Fenton (BCF) model [24] are being utilized. Figure 6.7 illustrates the two models.

- FK Model: The Fenton-Karma model [42], introduced by Flavio Fenton and



(a) Fenton-Karma (FK) model [42].



(b) Bueno-Cherry-Fenton (BCF) model [24].

Figure 6.7: Hybrid models of cardiac cells.

Alain Karma in 1998, is a simplified model of cardiac cell electrophysiology. It is designed to represent the electrical activity within cardiac cells by using three variables: membrane potential (v), gating variable (w), and recovery variable (u). These variables capture the essential features of action potential generation and propagation in cardiac tissue, including the fast inward sodium current, the slow inward calcium current, and the outward potassium current. The Fenton-Karma model is based on a set of nonlinear ordinary differential equations (ODEs) that describe the time evolution of the three variables. This model is computationally efficient and can be used to simulate electrical activity in large cardiac tissue networks without requiring excessive computational resources.

Run	Model	Initial State	Result	Time (sec)
1	BCF	$(u = 0, v = 1, w = 1, s = 0, \epsilon \in [0.9, 1.1])$	δ-sat	13.357
2	FK	$(u = 0, v = 1, w = 1, \epsilon \in [0.9, 1.1])$	δ-sat	9.380
3	BCF	$(u = 0, v = 1, w = 1, s = 0, \epsilon \in [0.0, 0.25])$	unsat	0.060
4	FK	$(u = 0, v = 1, w = 1, \epsilon \in [0.0, 0.25])$	unsat	0.909

Table 6.1: Experimental results for cardiac-cell models: Runs 1 and 2 investigate whether a significant stimulation can induce excitability in the cardiac cell models BCF and FK. Runs 3 and 4 aim to determine whether these models can disregard minor stimulation in each cardiac cell.

- BCF Model: The Bueno-Cherry-Fenton model [24] is an extension of the Fenton-Karma model. It builds upon the original model by incorporating additional ionic currents and a more detailed description of the intracellular calcium dynamics. This increased complexity allows the model to better represent the electrophysiological properties of cardiac cells, specifically ventricular cells.

Checking Excitability via Reachability Analysis To determine if a successful action potential (AP) initiation occurs, each hybrid system must transition from mode 1 to mode 4, where the peak of the variable u corresponds to cardiac muscle contraction. The question of identifying initial values or parameters leading to a loss of excitability in a cardiac cell is translated into reachability problems.

To maintain proper cardiac cell function in noisy environments, the system should filter out insignificant stimulation (parameter ϵ in the models). We expect action potentials (APs) to be initiated for large ϵ ($[0.9, 1.1]$) but not for small ϵ ($[0.0, 0.25]$).

We formulated four reachability problems using dReach for two different models (FK, BCF) and two different ϵ ranges (small, large). Then, we ran dReach to assess mode 4's reachability from mode 1 using the parameter values from [24]. Our experiment was conducted in an environment consisting of an AMD EPYC 7R32 CPU running at 2.8 GHz with 16 cores, and 32 GB of RAM. The operating system used was Ubuntu 20.04 LTS. We used $\delta = 1e - 3$ in dReach.

The experimental results are presented in Table 6.1. For the initial state ($u = 0, v = 1, w = 1, s = 0, \epsilon \in [0.9, 1.1]$) in Mode 1, we found that Mode 4 is reachable (δ-sat) for both models (Run#1 and Run#2 in Table 6.1). However, starting from the state ($u = 0, v = 1, w = 1, s = 0, \epsilon \in [0.0, 0.25]$) in Mode 1, we found that Mode 4 is not reachable (unsat) for both the BCF and FK models (Run#3 and Run#4 in Table 6.1), demonstrating the models' robustness to stimulation amplitude.

6.6.3 Personalized Prostate Cancer Therapy

Hormone Therapy for Prostate Cancer Prostate cancer is a significant threat to the health of men in the United States, ranking as the second leading cause of cancer-related deaths [108]. Hormone therapy, in the form of androgen deprivation, has been used to manage advanced prostate cancer for several decades, although its optimal application remains controversial [23]. However, this treatment method, known as Continuous Androgen Suppression (CAS), is associated with various side effects [2], including anemia, osteoporosis, and impotence. Furthermore, the median duration of CAS treatment before a relapse occurs is 18-24 months, due to the proliferation of Castration Resistant Cancer Cells (CRCs).

A method for limiting the negative effects of Castration-resistant cancer (CRC) and delaying relapse is Intermittent Androgen Suppression (IAS) [58]. IAS restricts the duration of low-androgen conditions to prevent the emergence of CRCs, as reported in [19]. The therapy alternates between treatment and non-treatment phases by monitoring the Prostate-Specific Antigen (PSA) levels in the patient's serum:

1. When the PSA level decreases and reaches a lower threshold value r_0, androgen suppression is suspended.

2. When the PSA level increases and reaches an upper threshold value r_1, androgen suppression is resumed by the administration of medical agents.

Recently clinical phase II and III trials have established that IAS provides improvements in quality of life and cost [20, 21]. However, the superiority of IAS over CAS in terms of time to relapse and cancer-specific survival is subject to patient-specific factors and the on- and off-treatment scheme, according to these trials [20, 21]. Hence, a critical challenge remains in determining a personalized treatment plan that optimizes therapeutic outcomes for each individual patient.

Nonlinear Hybrid System Model In [87], we proposed a nonlinear hybrid-system model (Figure 6.8) to describe the prostate cancer progression dynamics under IAS thereapy. Our model extends the models previously proposed in [71, 70, 67]. This model has the following four state variables:

- $x(t)$: The population of HSCs (Hormone Sensitive Cells).
- $y(t)$: The population of CRCs (Castration Resistant Cells).
- $z(t)$: Serum androgen concentration level.
- $v(t)$: Serum PSA (Prostate-Specific Antigen) level.

The model has two modes: *on-treatment* mode and *off-treatment* mode. Following [67], in the off-treatment mode (Mode 2), the androgen concentration is maintained at the normal level z_0 by homeostasis. In the on-treatment (Mode 1), the androgen is cleared at a rate $\frac{1}{\tau}$.

Personalized Therapy Design We utilize delta-reachability analysis to create treatment schemes tailored to individual patients. Specifically, the treatment scheme is customized to the patient by determining appropriate parameter values through solving the parameter identification problem. This problem involves (i) setting the ranges of scheduling parameters, r_0 and r_1, and (ii) checking if the "no cancer relapse" invariants ($x \leq 35$ and $y \leq 1$) are not violated within a year, while also ensuring that the goal state is reachable.

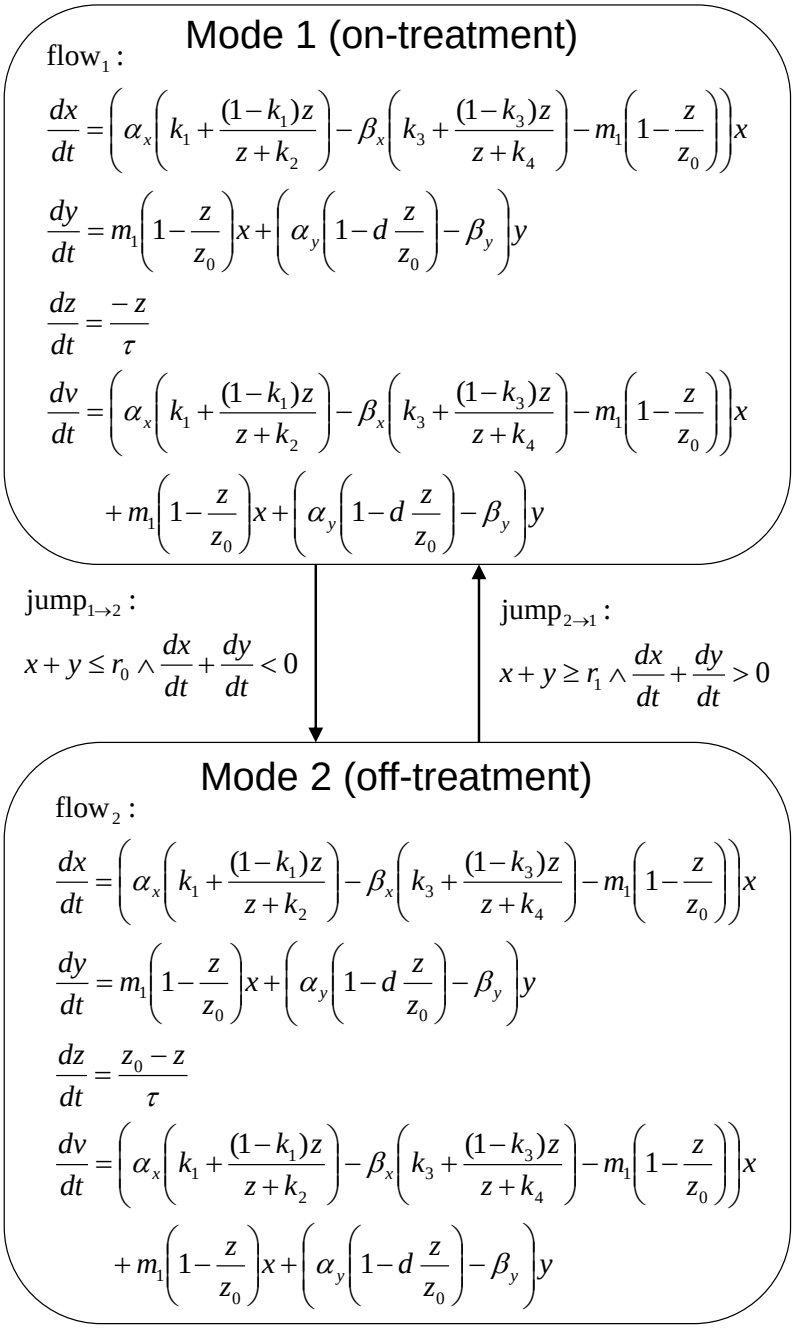


Figure 6.8: A hybrid automaton model for prostate cancer hormone therapy.

Patient#	Initial State	Result	Time (sec)
1	$r_0 \in [0.0, 7.99]$, $r_1 \in [8.0, 15.0]$	δ-sat (k=1)	236.70
15	$r_0 \in [0.0, 7.99]$, $r_1 \in [8.0, 15.0]$	δ-sat (k=1)	7764.23
26	$r_0 \in [0.0, 7.99]$, $r_1 \in [8.0, 15.0]$	δ-sat (k=3)	2420.55
11	$r_0 \in [0.0, 7.99]$, $r_1 \in [8.0, 15.0]$	unsat ($k = [1..8]$)	3723.21

Table 6.2: Experimental results for personalized therapy design: It shows that an IAS treatment schedule was successfully designed for patients 1, 15, and 26. However, no feasible treatment schemes could be identified for Patient 11.

1. If the result of this analysis is *unsat*, it indicates that androgen suppression therapy is not the optimal treatment for the patient and other therapeutic interventions should be considered.

2. Conversely, if the analysis returns a *delta-sat* answer, a treatment scheme containing feasible values of r_0 and r_1 is provided, which can help in preventing or delaying the relapse within a bounded time. Notably, a result of $r_0 = 0$ suggests that a CAS scheme may be a more appropriate treatment option for the patient instead of an IAS scheme.

Evaluation Results Our method was evaluated using patient data collected by [21][3]. Our experiment was conducted in an environment consisting of an AMD EPYC 7R32 CPU running at 2.8 GHz with 16 cores, and 32 GB of RAM. The operating system used was Ubuntu 20.04 LTS. Our study's outcomes are presented in Table 6.2. Notably, we were able to design an IAS treatment schedule for patients 1, 15, and 26. In contrast, our parameter identification approach returned unsat for Patient 11, which indicates that no feasible treatment schemes could be identified for this individual. We have generated a personalized treatment schedule for Patient #26 using the IAS scheme with $r_0 = 5.984$ and $r_1 = 8.314$, and visualized the results in Figure 6.9. The visualization demonstrates that the proposed treatment scheme

[3]The patient data can be accessed through the website, `http://nicholasbruchovsky.com/clinicalResearch.html`

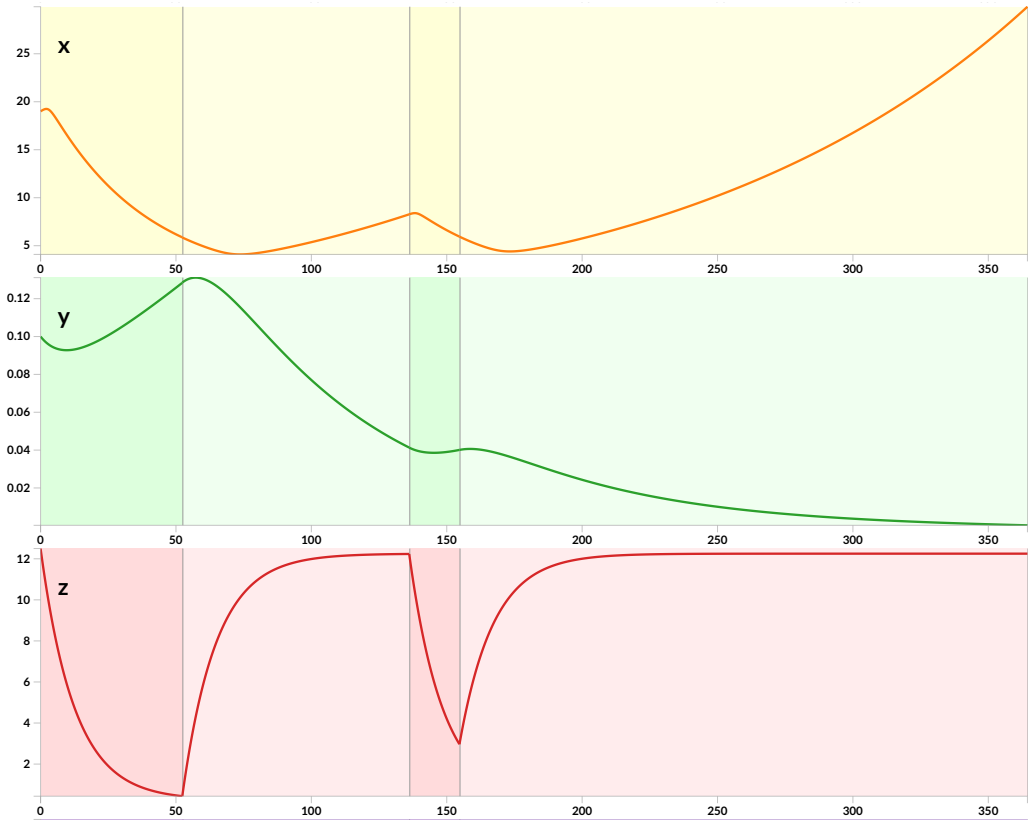


Figure 6.9: Visualization of a personalized treatement schedule for Patient #26 ($r_0 = 5.984$ and $r_1 = 8.314$). The population of Hormone Sensitive Cells (HSCs) is denoted by $x(t)$, while the population of Castration Resistant Cells (CRCs) is represented by $y(t)$. The serum androgen concentration level is denoted by $z(t)$. The visualization demonstrates that the proposed treatment scheme is effective in maintaining the HSCs and CRCs level below the corresponding thresholds ($x \leq 35$ and $y \leq 1$).

is effective in maintaining the HSCs and CRCs level below the corresponding thresholds ($x \leq 35$ and $y \leq 1$).

6.7 Conclusion

We have presented dReach, a bounded reachability analysis tool for hybrid systems. dReach encodes bounded reachability problems of hybrid systems as first-order

formulas over the real numbers and solves them using δ-decision procedures in the SMT solver dReal. Our tool is capable of handling a wide range of highly nonlinear hybrid systems. Our experimental results demonstrate the effectiveness of dReach on several case studies, including a model of atrial fibrillation and a personalized prostate cancer treatment model. These case studies demonstrate the wide range of applications for which dReach can be used.

Chapter 7

Conclusion

7.1 Review of Thesis Contributions

To summarize, this dissertation introduces an efficient delta-decision procedure capable of precisely and efficiently addressing real-world problems, including those that involve nonlinear functions, ordinary differential equations, and universal quantifications.

Delta-decision Algorithms This dissertation presents two algorithms within the delta-decision framework. The first algorithm addresses the SMT problem for real numbers with general Lipschitz-continuous ODEs. The second algorithm proposes delta-complete decision procedures for nonlinear SMT problems over real numbers that incorporate universal quantification and a range of nonlinear functions.

Delta-decision Tools This dissertation also presents the design and implementation of the delta-decision procedure, dReal, and the delta-reachability analysis tool, dReach. These algorithms have been shown to be effective and scalable through practical benchmarks and realistic models from various fields, including global nonlinear optimization, computational biology, theorem proving, and control synthesis, demonstrating their ability to solve real-world problems with precision and compu-

tational efficiency. Our tools have been adopted in academia and industry, serving as the foundation for numerous tools such as APEX [97], BioPSy [90], Daisy [33], DiffRNN [91], Drake [111], ETCetera [35], FOSSIL [1], HybridSyncAADL [84], JDart [89], LinSyn [99], Manifold [14], PGCD [8], PolyReach [110], ProbReach [107], STLMC [118], Sally [39], Verisig [69], Viatra [106], SReach [115], and symQV [12].

7.2 Future Directions

As Alan Turing [113] stated, "We can only see a short distance ahead, but we can see plenty there that needs to be done." In this section, we present the following research directions that future research can expand upon the work presented in this thesis.

Incorporate Under-approximation-based Methods Our current approach of Interval Constraint Propagation (ICP) maintains over-approximated sets of solution spaces, but we aim to enhance it by incorporating under-approximation-based techniques. To that end, we plan to investigate the dual of ICP and integrate it with our current method. We will start by leveraging sampling-based techniques. Each sample point will undergo a satisfiability check, and we will consider incorporating local-optimization techniques to enhance the quality of the samples and find better points. The combination of these approaches should lead to a more efficient theory solver. Multi-threading can be utilized to achieve this integration, with ICP running in one thread and under-approximation-based methods in another.

Learning Solver Heuristics from Data We aim to create solver heuristics that are tailored to specific problem groups through data-driven methods. In industrial settings, it is common to face similar problems repeatedly, and we aim to make the most of this information to improve solver performance. One example of this is the branching heuristics in ICP. As we have demonstrated in [66], different branching heuristics can have a significant impact on overall solver performance.

To achieve this, we plan to employ reinforcement learning techniques to learn branching heuristics from previous runs, and then apply them to solve similar problem instances in the future. This will allow us to effectively leverage historical data and enhance the performance of the solver on similar problem groups.

Bibliography

[1] Alessandro Abate, Daniele Ahmed, Alec Edwards, Mirco Giacobbe, and Andrea Peruffo. "FOSSIL: A Software Tool for the Formal Synthesis of Lyapunov Functions and Barrier Certificates using Neural Networks". In: *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*. HSCC '21. Association for Computing Machinery, 2021, p. 11. ISBN: 978-1-4503-8339-4/21/05. DOI: 10.1145/3447928.3456646. URL: https://doi.org/10.1145/3447928.3456646.

[2] Hamed Ahmadi and Siamak Daneshmand. "Androgen deprivation therapy for prostate cancer: long-term safety and patient outcomes". In: *Patient related outcome measures* (2014), pp. 63–70.

[3] Behzad Akbarpour and Lawrence C. Paulson. "MetiTarski: An Automatic Theorem Prover for Real-Valued Special Functions". In: *J. Autom. Reasoning* 44.3 (2010), pp. 175–205.

[4] Matthias Althoff and Bruce H. Krogh. "Reachability Analysis of Nonlinear Differential-Algebraic Systems". In: *IEEE Trans. Automat. Contr.* 59.2 (2014), pp. 371–383. DOI: 10.1109/TAC.2013.2285751. URL: http://dx.doi.org/10.1109/TAC.2013.2285751.

[5] Rajeev Alur. "Formal verification of hybrid systems". In: *Proceedings of the ninth ACM international conference on Embedded software*. 2011, pp. 273–278.

[6] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei Hsin Ho. "Hybrid automata: An algorithmic approach to the specification and

verification of hybrid systems". In: *Hybrid Systems*. Ed. by Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 209–229. ISBN: 978-3-540-48060-0.

[7] Andrea Balluchi, Luca Benvenuti, Maria Domenica Di Benedetto, Claudio Pinello, and Alberto L Sangiovanni-Vincentelli. "Automotive engine control and hybrid systems: Challenges and opportunities". In: *Proceedings of the IEEE* 88.7 (2000), pp. 888–912.

[8] Gregor B Banusić, Rupak Majumdar, Marcus Pirron, Anne-Kathrin Schmuck, and Damien Zufferey. "PGCD: robot programming and verification with geometry, concurrency, and dynamics". In: *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*. 2019, pp. 57–66.

[9] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, et al. "cvc5: A versatile and industrial-strength SMT solver". In: *Tools and Algorithms for the Construction and Analysis of Systems: 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I*. Springer. 2022, pp. 415–442.

[10] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. "CVC4". In: *Proceedings of the 23rd International Conference on Computer Aided Verification*. CAV'11. Snowbird, UT: Springer-Verlag, 2011, pp. 171–177. ISBN: 978-3-642-22109-5. URL: http://dl.acm.org/citation.cfm?id=2032305.2032319.

[11] Clark Barrett, Aaron Stump, and Cesare Tinelli. "The SMT-LIB Standard: Version 2.0". In: *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*. Ed. by A. Gupta and D. Kroening. 2010.

[12] Fabian Bauer-Marquart, Stefan Leue, and Christian Schilling. "symQV: Automated Symbolic Verification of Quantum Programs". In: (Dec. 2022). arXiv: 2212.02267 [quant-ph].

[13] Frederic Benhamou and Laurent Granvilliers. "Continuous and Interval Constraints". In: *Handbook of Constraint Programming*. Ed. by F. Rossi, P. van Beek, and T. Walsh. Elsevier, 2006. Chap. 16.

[14] Murphy Berzish, Asif Khan, Atulan Zaman, Vijay Ganesh, and Derek Rayside. "Manifold: An SMT-Based Declarative Language for Electronic and Microfluidic Design Synthesis". In: *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*. CASCON '16. Toronto, Ontario, Canada: IBM Corp., 2016, 188–193.

[15] Armin Biere. "PicoSAT Essentials". In: *JSAT* 4.2-4 (2008), pp. 75–97.

[16] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. "*v*Z - An Optimizing SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Christel Baier and Cesare Tinelli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 194–199. ISBN: 978-3-662-46681-0.

[17] Cristina Borralleras, Salvador Lucas, Rafael Navarro-Marset, Enric Rodríguez-Carbonell, and Albert Rubio. "Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic". In: *CADE*. 2009, pp. 294–305.

[18] Christopher W. Brown and James H. Davenport. "The Complexity of Quantifier Elimination and Cylindrical Algebraic Decomposition". In: *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*. ISSAC '07. Waterloo, Ontario, Canada: ACM, 2007, pp. 54–60. ISBN: 978-1-59593-743-8. DOI: 10.1145/1277548.1277557. URL: http://doi.acm.org/10.1145/1277548.1277557.

[19] N. Bruchovsky, S. L. Goldenberg, P. S. Rennie, and Gleave M. E. "Theoretical considerations and initial clinical results of intermittent hormone treatment

of patients with advanced prostatic carcinoma". In: *Urologe A* 34 (1995), pp. 389–392.

[20] N. Bruchovsky, L. Klotz, J. Crook, S. Malone, C. Ludgte, W. Morris, M. E. Gleave, S. L. Goldenberg, and P. S. Rennie. "Final Results of the Canadian Prospective Phase II Trial of Intermittent Androgen Suppression for Men in Biochemical Recurrence after Radiotherapy for Locally Advanced Prostate Cancer". In: *Cancer* 107 (2006), pp. 389–395.

[21] N. Bruchovsky, L. Klotz, J. Crook, S. Malone, C. Ludgte, W. Morris, M. E. Gleave, S. L. Goldenberg, and P. S. Rennie. "Locally Advanced Prostate Cancer—Biochemical Results From a Prospective Phase II Study of Intermittent Androgen Suppression for Men With Evidence of Prostate-Specific Antigen Recurrence After Radiotherapy". In: *Cancer* 109 (2007), pp. 858–867.

[22] Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. "The OpenSMT Solver". In: *TACAS*. 2010, pp. 150–153.

[23] Nicholas C. Buchan and S. Larry Goldenberg. "Intermittent androgen suppression for prostate cancer". In: *Nat. Rev. Urol.* 7 (2010), pp. 552–560.

[24] Alfonso Bueno-Orovio, Elizabeth M Cherry, and Flavio H Fenton. "Minimal model for human ventricular action potentials in tissue". In: *Journal of theoretical biology* 253.3 (2008), pp. 544–560.

[25] Maciej Capinski, Jacek Cyranka, Zbigiew Galias, Tomasz Kapela, Marian Mrozek, Paweł Pilarczyk, Daniel Wilczak, and Piotr Zgliczynski. *CAPD4*. 2000-2018. URL: `http://capd.ii.uj.edu.pl`.

[26] Thomas J. Watson IBM Research Center and R.K. Treiber. *Systems Programming: Coping with Parallelism*. Research Report RJ. International Business Machines Incorporated, Thomas J. Watson Research Center, 1986. URL: `https://books.google.com/books?id=YQg3HAAACAAJ`.

[27] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. "Taylor Model Flowpipe Construction for Non-linear Hybrid Systems". In: *RTSS*. 2012, pp. 183–192.

[28] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Roberto Sebastiani, and Cristian Stenico. "Satisfiability Modulo the Theory of Costs: Foundations and Applications". In: *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS'10. Paphos, Cyprus: Springer-Verlag, 2010, pp. 99–113. ISBN: 3-642-12001-6, 978-3-642-12001-5. DOI: 10.1007/978-3-642-12002-2_8. URL: http://dx.doi.org/10.1007/978-3-642-12002-2_8.

[29] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. "A quantifier-free SMT encoding of non-linear hybrid automata". In: *FMCAD*. 2012, pp. 187–195.

[30] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. "SMT-Based Verification of Hybrid Systems". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. 2012.

[31] George E. Collins. "Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition". In: *Automata Theory and Formal Languages*. 1975, pp. 134–183.

[32] Jorge Cruz and Pedro Barahona. "Constraint Satisfaction Differential Problems". In: *CP*. 2003, pp. 259–273.

[33] Eva Darulova, Anastasiia Izycheva, Fariha Nasir, Fabian Ritter, Heiko Becker, and Robert Bastian. "Daisy - Framework for Analysis and Optimization of Numerical Programs (Tool Paper)". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Dirk Beyer and Marieke Huisman. Cham: Springer International Publishing, 2018, pp. 270–287. ISBN: 978-3-319-89960-2.

[34] Leonardo De Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS'08/ETAPS'08. Budapest, Hungary: Springer-Verlag, 2008, pp. 337–340. ISBN: 3-540-78799-2, 978-3-540-78799-0. URL: http://dl.acm.org/citation.cfm?id=1792734.1792766.

[35] Giannis Delimpaltadakis, Gabriel de Albuquerque Gleizer, Ivo van Straalen, and Manuel Mazo Jr. "ETCetera: Beyond Event-Triggered Control". In: *25th ACM International Conference on Hybrid Systems: Computation and Control*. HSCC '22. Milan, Italy: Association for Computing Machinery, 2022. ISBN: 9781450391962. DOI: 10.1145/3501710.3519523. URL: https://doi.org/10.1145/3501710.3519523.

[36] Bruno Dutertre. "Solving Exists/Forall Problems With Yices". In: *Workshop on Satisfiability Modulo Theories*. 2015.

[37] Bruno Dutertre. "Yices 2.2". In: *Computer Aided Verification: 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings 26*. Springer. 2014, pp. 737–744.

[38] Bruno Dutertre and Leonardo De Moura. "The Yices SMT solver". In: *Tool paper at http://yices. csl. sri. com/tool-paper. pdf* 2.2 (2006), pp. 1–2.

[39] Bruno Dutertre, Dejan Jonanovic, and Jorge Navas. *Advanced symbolic analysis tools for fault-tolerant integrated distributed systems*. Tech. rep. 2018.

[40] Andreas Eggers, Martin Fränzle, and Christian Herde. "SAT Modulo ODE: A Direct SAT Approach to Hybrid Systems". In: *ATVA*. 2008, pp. 171–185.

[41] Andreas Eggers, Nacim Ramdani, Nedialko Nedialkov, and Martin Fränzle. "Improving SAT Modulo ODE for Hybrid Systems Analysis by Combining Different Enclosure Methods". In: *SEFM*. 2011, pp. 172–187.

[42] Flavio Fenton and Alain Karma. "Vortex dynamics in three-dimensional continuous myocardium with fiber rotation: Filament instability and fibrillation". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 8.1 (1998), pp. 20–47.

[43] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. "Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure". In: *JSAT* 1.3-4 (2007), pp. 209–236.

[44] Martin Fränzle, Tino Teige, and Andreas Eggers. "Engineering constraint solvers for automatic analysis of probabilistic hybrid automata". In: *J. Log. Algebr. Program.* 79.7 (2010), pp. 436–466.

[45] Goran Frehse. "PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech". In: *HSCC*. 2005, pp. 258–273.

[46] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. "SpaceEx: Scalable Verification of Hybrid Systems". In: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. 2011, pp. 379–395. DOI: 10.1007/978-3-642-22110-1_30. URL: http://dx.doi.org/10.1007/978-3-642-22110-1_30.

[47] Malay K. Ganai and Franjo Ivančić. "Efficient Decision Procedure for Non-linear Arithmetic Constraints using CORDIC". In: *Formal Methods in Computer Aided Design (FMCAD)*. 2009.

[48] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. "Delta-Complete Decision Procedures for Satisfiability over the Reals". In: *IJCAR*. Ed. by Bernhard Gramlich, Dale Miller, and Uli Sattler. Springer, 2012, pp. 286–300.

[49] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. "Delta-Decidability over the Reals". In: *LICS*. IEEE Computer Society, 2012.

[50] Sicun Gao, Malay Ganai, Franjo Ivancic, Aarti Gupta, Sriram Sankaranarayanan, and Edmund Clarke. "Integrating ICP and LRA Solvers for Deciding Nonlinear Real Arithmetic". In: *FMCAD*. 2010.

[51] Sicun Gao, Soonho Kong, Wei Chen, and Edmund M. Clarke. "Delta-Complete Analysis for Bounded Reachability of Hybrid Systems". In: (2014). CMU SCS Technical Report CMU-CS-14-111.

[52] Sicun Gao, Soonho Kong, and Edmund M. Clarke. "dReal: An SMT Solver for Nonlinear Theories over the Reals". In: *CADE*. Ed. by Maria Paola Bonacina. 2013, pp. 208–214.

[53] Sicun Gao, Soonho Kong, and Edmund M. Clarke. "Proof Generation from Delta-Decisions". In: *Symbolic and Numeric Algorithms for Scientific Computing*. SYNASC'14. Sept. 2014, pp. 156–163. DOI: 10.1109/SYNASC.2014.29.

[54] Sicun Gao, Soonho Kong, and Edmund M. Clarke. "Satisfiability modulo ODEs". In: *Formal Methods in Computer-Aided Design*. FMCAD'13. 2013, pp. 105–112. DOI: 10.1109/FMCAD.2013.6679398.

[55] Sicun Gao, Soonho Kong, and Edmund M. Clarke. "Satisfiability Modulo ODEs". In: *FMCAD*. 2013, pp. 105–112.

[56] Yeting Ge, Clark Barrett, and Cesare Tinelli. "Solving Quantified Verification Conditions using Satisfiability Modulo Theories". In: *Proceedings of the 21st International Conference on Automated Deduction (CADE-21), Bremen, Germany*. Ed. by F. Pfenning. Vol. 4603. Lecture Notes in Computer Science. Springer, 2007, pp. 167–182. URL: ftp://ftp.cs.uiowa.edu/pub/tinelli/papers/GeBT-CADE-07.pdf.

[57] Andrew Gibiansky. "Quadcopter dynamics, simulation, and control". In: *Andrew.gibiansky.com* (2012).

[58] Martin Gleave, S Larry Goldenberg, N Bruchovsky, and P Rennie. "Intermittent androgen suppression for prostate cancer: rationale and clinical experience". In: *Prostate cancer and prostatic diseases* 1.6 (1998), pp. 289–296.

[59] Alexandre Goldsztejn, Olivier Mullier, Damien Eveillard, and Hiroshi Hosobe. "Including Ordinary Differential Equations Based Constraints in the Standard CP Framework". In: *CP*. 2010, pp. 221–235.

[60] Laurent Granvilliers. "Parameter Estimation Using Interval Computations". In: *SIAM J. Sci. Comput.* 26.2 (Feb. 2005), pp. 591–612.

[61] Radu Grosu, Grégory Batt, Flavio H. Fenton, James Glimm, Colas Le Guernic, Scott A. Smolka, and Ezio Bartocci. "From Cardiac Cells to Genetic Regulatory Networks". In: *CAV*. 2011, pp. 396–411.

[62] Wassim M. Haddad and VijaySekhar Chellaboina. *Nonlinear Dynamical Systems and Control: A Lyapunov-Based Approach*. Princeton University Press, 2008. ISBN: 9780691133294. URL: `http://www.jstor.org/stable/j.ctvcm4hws` (visited on 01/29/2023).

[63] Thomas C. Hales. "Introduction to the Flyspeck Project". In: *Mathematics, Algorithms, Proofs*. 2005.

[64] Thomas A. Henzinger. "The Theory of Hybrid Automata". In: *LICS*. 1996, pp. 278–292.

[65] Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige. "Analysis of Hybrid Systems Using HySAT". In: *ICONS*. 2008, pp. 196–201.

[66] Calvin Huang, Soonho Kong, Sicun Gao, and Damien Zufferey. "Evaluating Branching Heuristics in Interval Constraint Propagation for Satisfiability". In: *Numerical Software Verification*. Ed. by Majid Zamani and Damien Zufferey. Cham: Springer International Publishing, 2019, pp. 85–100. ISBN: 978-3-030-28423-7.

[67] Aiko Miyamura Ideta, Gouhei Tanaka, Takumi Takeuchi, and Kazuyuki Aihara. "A mathematical model of intermittent androgen suppression for prostate cancer". In: *J. Nonlinear Sci.* 18 (2008), pp. 593–614.

[68] Daisuke Ishii, Kazunori Ueda, and Hiroshi Hosobe. "An interval-based SAT modulo ODE solver for model checking nonlinear hybrid systems". In: *STTT* 13.5 (2011), pp. 449–461.

[69] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. "Verisig: Verifying Safety Properties of Hybrid Systems with Neural Network Controllers". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. HSCC '19. Montreal, Quebec, Canada: Association for Computing Machinery, 2019, 169–178. ISBN: 9781450362825. DOI: 10.1145/3302504.3311806. URL: https://doi.org/10.1145/3302504.3311806.

[70] T. L. Jackson. "A mathematical investigation of the multiple pathways to recurrent prostate cancer: comparison with experimental data". In: *Neoplasia* 6 (2004), pp. 697–704.

[71] T. L. Jackson. "A mathematical model of prostate tumor growth and androgen-independent replace". In: *Discrete Cont. Dyn. Syst. Ser. B* 4 (2004), pp. 187–201.

[72] Momin Jamil and Xin-She Yang. "A literature survey of benchmark functions for global optimisation problems". In: *International Journal of Mathematical Modelling and Numerical Optimisation* 4.2 (2013), pp. 150–194.

[73] Rolf Johansson and Anders Rantzer. *Nonlinear and hybrid systems in automotive control*. Vol. 146. Springer, 2003.

[74] Steven G. Johnson. *The NLopt nonlinear-optimization package*. 2011. URL: http://ab-initio.mit.edu/nlopt.

[75] Dejan Jovanovic and Leonardo Mendonça de Moura. "Solving Non-linear Arithmetic". In: *IJCAR*. 2012, pp. 339–354.

[76] James Kapinski, Jyotirmoy V. Deshmukh, Sriram Sankaranarayanan, and Nikos Arechiga. "Simulation-guided Lyapunov Analysis for Hybrid Dynamical Systems". In: *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*. HSCC '14. Berlin, Germany: ACM, 2014, pp. 133–142. ISBN: 978-1-4503-2732-9. DOI: 10.1145/2562059.2562139. URL: http://doi.acm.org/10.1145/2562059.2562139.

[77] Ker-I. Ko. *Complexity Theory of Real Functions*. BirkHauser, 1991.

[78] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. "dReach: Delta-Reachability Analysis for Hybrid Systems". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 2015.

[79] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. "dReach: δ-reachability analysis for hybrid systems". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2015, pp. 200–205.

[80] Soonho Kong, Armando Solar-Lezama, and Sicun Gao. "Delta-Decision Procedures for Exists-Forall Problems over the Reals". In: *Computer Aided Verification*. Ed. by Hana Chockler and Georg Weissenbacher. CAV'18. Cham: Springer International Publishing, 2018, pp. 219–235. ISBN: 978-3-319-96142-2. DOI: 10.1007/978-3-319-96142-2_15. URL: https://link.springer.com/content/pdf/10.1007/978-3-319-96142-2_15.pdf.

[81] Dieter Kraft. "Algorithm 733: TOMP–Fortran Modules for Optimal Control Calculations". In: *ACM Trans. Math. Softw.* 20.3 (Sept. 1994), pp. 262–281. ISSN: 0098-3500. DOI: 10.1145/192115.192124. URL: http://doi.acm.org/10.1145/192115.192124.

[82] Joseph La Salle and Solomon Lefschetz. *Stability by Liapunov's direct method*. Academic Press, 1961.

[83] Edward A Lee, Sanjit A Seshia, et al. "Introduction to embedded systems". In: *A cyber-physical systems approach* 499 (2011).

[84] Jaehun Lee, Sharon Kim, Kyungmin Bae, and Peter Csaba Ölveczky. "Hybrid SynchAADL: Modeling and Formal Analysis of Virtually Synchronous CPSs in AADL". In: *Computer Aided Verification*. Ed. by Alexandra Silva and K. Rustan M. Leino. Cham: Springer International Publishing, 2021, pp. 491–504. ISBN: 978-3-030-81685-8.

[85] Youdong Lin and Mark A. Stadtherr. "Guaranteed state and parameter estimation for nonlinear continuous-time systems with bounded-error measurements". In: *Industrial and Engineering Chemistry Research* (2007), pp. 7198–7207.

[86] Bing Liu, Soonho Kong, Sicun Gao, Paolo Zuliani, and Edmund M. Clarke. "Parameter Synthesis for Cardiac Cell Hybrid Models Using δ-Decisions". In: *Computational Methods in Systems Biology - 12th International Conference, CMSB 2014, Manchester, UK, November 17-19, 2014, Proceedings*. Ed. by Pedro Mendes, Joseph O. Dada, and Kieran Smallbone. Vol. 8859. Lecture Notes in Computer Science. Springer, 2014, pp. 99–113. ISBN: 978-3-319-12981-5. DOI: 10.1007/978-3-319-12982-2_8. URL: http://dx.doi.org/10.1007/978-3-319-12982-2_8.

[87] Bing Liu, Soonho Kong, Sicun Gao, Paolo Zuliani, and Edmund M. Clarke. "Towards Personalized Prostate Cancer Therapy Using Delta-reachability Analysis". In: *Hybrid Systems: Computation and Control*. HSCC'15. Seattle, Washington: ACM, 2015, pp. 227–232. ISBN: 978-1-4503-3433-4. DOI: 10.1145/2728606.2728634. URL: http://doi.acm.org/10.1145/2728606.2728634.

[88] R. Lougee-Heimer. "The Common Optimization INterface for Operations Research: Promoting Open-source Software in the Operations Research Community". In: *IBM J. Res. Dev.* 47.1 (Jan. 2003), pp. 57–66. ISSN: 0018-8646. DOI: 10.1147/rd.471.0057. URL: http://dx.doi.org/10.1147/rd.471.0057.

[89] Kasper Luckow, Marko Dimjašević, Dimitra Giannakopoulou, Falk Howar, Malte Isberner, Temesghen Kahsai, Zvonimir Rakamarić, and Vishwanath Raman. "JDart: A Dynamic Symbolic Analysis Framework". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Marsha Chechik and Jean-François Raskin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 442–459. ISBN: 978-3-662-49674-9.

[90] Curtis Madsen, Fedor Shmarov, and Paolo Zuliani. "BioPSy: An SMT-based Tool for Guaranteed Parameter Set Synthesis of Biological Models". In: *Computational Methods in Systems Biology*. Ed. by Olivier Roux and Jérémie Bourdon. Cham: Springer International Publishing, 2015, pp. 182–194. ISBN: 978-3-319-23401-4.

[91] Sara Mohammadinejad, Brandon Paulsen, Jyotirmoy V. Deshmukh, and Chao Wang. "DiffRNN: Differential Verification of Recurrent Neural Networks". In: *Formal Modeling and Analysis of Timed Systems*. Ed. by Catalin Dima and Mahsa Shirmohammadi. Cham: Springer International Publishing, 2021, pp. 117–134. ISBN: 978-3-030-85037-1.

[92] Leonardo Moura and Nikolaj Bjørner. "Efficient E-Matching for SMT Solvers". In: *Proceedings of the 21st International Conference on Automated Deduction: Automated Deduction*. CADE-21. Bremen, Germany: Springer-Verlag, 2007, pp. 183–198. ISBN: 978-3-540-73594-6. DOI: 10.1007/978-3-540-73595-3_13. URL: http://dx.doi.org/10.1007/978-3-540-73595-3_13.

[93] César Muñoz and Anthony Narkawicz. "Formalization of Bernstein Polynomials and Applications to Global Optimization". In: *Journal of Automated Reasoning* 51.2 (2013), pp. 151–196. ISSN: 1573-0670. DOI: 10.1007/s10817-012-9256-3. URL: https://doi.org/10.1007/s10817-012-9256-3.

[94] Robert Nieuwenhuis and Albert Oliveras. "On SAT Modulo Theories and Optimization Problems". In: *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing*. SAT'06. Seattle, WA: Springer-

Verlag, 2006, pp. 156–169. ISBN: 3-540-37206-7, 978-3-540-37206-6. DOI: 10.1007/11814948_18. URL: http://dx.doi.org/10.1007/11814948_18.

[95] Peter Nightingale. "Consistency for Quantified Constraint Satisfaction Problems". In: *Principles and Practice of Constraint Programming - CP 2005*. Ed. by Peter van Beek. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 792–796. ISBN: 978-3-540-32050-0.

[96] Pierluigi Nuzzo, Alberto Puggelli, Sanjit A. Seshia, and Alberto L. Sangiovanni-Vincentelli. "CalCS: SMT solving for non-linear convex constraints". In: *FMCAD*. 2010, pp. 71–79.

[97] Matthew O'Kelly, Houssam Abbas, Sicun Gao, Shin'ichi Shiraishi, Shinpei Kato, and Rahul Mangharam. "APEX: Autonomous Vehicle Plan Verification and Execution". In: *SAE World Congress* 1 (2016).

[98] Grant Olney Passmore and Paul B. Jackson. "Combined Decision Techniques for the Existential Theory of the Reals". In: *Calculemus/MKM*. 2009, pp. 122–137.

[99] Brandon Paulsen and Chao Wang. "LinSyn: Synthesizing Tight Linear Bounds for Arbitrary Neural Network Activation Functions". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Dana Fisman and Grigore Rosu. Cham: Springer International Publishing, 2022, pp. 357–376. ISBN: 978-3-030-99524-9.

[100] André Platzer, Jan-David Quesel, and Philipp Rümmer. "Real World Verification". In: *CADE*. 2009, pp. 485–501.

[101] Philip Polack, Florent Altché, Brigitte d'Andréa Novel, and Arnaud de La Fortelle. "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" In: *2017 IEEE intelligent vehicles symposium (IV)*. IEEE. 2017, pp. 812–818.

[102] MJD Powell. "Direct search algorithms for optimization calculations". In: *Acta numerica* 7 (1998), pp. 287–336.

[103] Stefan Ratschan. "Applications of Quantified Constraint Solving over the Reals - Bibliography". In: *CoRR* abs/1205.5571 (2012). arXiv: 1205.5571. URL: http://arxiv.org/abs/1205.5571.

[104] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett. "Counterexample-Guided Quantifier Instantiation for Synthesis in SMT". In: *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*. 2015, pp. 198–216. DOI: 10.1007/978-3-319-21668-3_12. URL: https://doi.org/10.1007/978-3-319-21668-3_12.

[105] Roberto Sebastiani and Silvia Tomasi. "Optimization in SMT with LA(Q) Cost Functions". In: *Proceedings of the 6th International Joint Conference on Automated Reasoning*. IJCAR'12. Manchester, UK: Springer-Verlag, 2012, pp. 484–498.

[106] Oszkár Semeráth, András Szabolcs Nagy, and Dániel Varró. "A Graph Solver for the Automated Generation of Consistent Domain-Specific Models". In: *40th International Conference on Software Engineering (ICSE 2018)*. ACM, May 2018, pp. 969–980.

[107] Fedor Shmarov and Paolo Zuliani. "ProbReach: verified probabilistic delta-reachability for stochastic hybrid systems." In: *HSCC* (2015), pp. 134–139.

[108] Rebecca Siegel, Deepa Naishadham, and Ahmedin Jemal. "Cancer Statistics, 2013". In: *CA. Cancer J. Clin.* 63 (2013), pp. 11–30.

[109] Armando Solar-Lezama. *Program synthesis by sketching*. University of California, Berkeley, 2008.

[110] Tim Sweering. *Applying Koopman methods for nonlinear reachability analysis*. Tech. rep. 2021.

[111] Russ Tedrake and the Drake Development Team. *Drake: Model-based design and verification for robotics*. 2019. URL: https://drake.mit.edu.

[112] Gilles Trombettoni, Ignacio Araya, Bertrand Neveu, and Gilles Chabert. "Inner Regions and Interval Linearizations for Global Optimization". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. AAAI'11. San Francisco, California: AAAI Press, 2011, pp. 99–104. URL: http://dl.acm.org/citation.cfm?id=2900423.2900439.

[113] Alan M. Turing. "Computing Machinery and Intelligence". In: *Mind* 59.October (1950), pp. 433–60. DOI: 10.1093/mind/lix.236.433.

[114] Lawrence Turyn. "The Damped Mathieu Equation". In: *Quarterly of Applied Mathematics* 51.2 (1993), pp. 389–398. ISSN: 0033569X, 15524485. URL: http://www.jstor.org/stable/43637931 (visited on 01/29/2023).

[115] Qinsi Wang, Paolo Zuliani, Soonho Kong, Sicun Gao, and Edmund M. Clarke. "SReach: A Probabilistic Bounded Delta-Reachability Analyzer for Stochastic Hybrid Systems". In: *Computational Methods in Systems Biology - 13th International Conference, CMSB 2015, Nantes, France, September 16-18, 2015, Proceedings*. Ed. by Olivier F. Roux and Jérémie Bourdon. Vol. 9308. Lecture Notes in Computer Science. Springer, 2015, pp. 15–27. ISBN: 978-3-319-23400-7. DOI: 10.1007/978-3-319-23401-4_3. URL: http://dx.doi.org/10.1007/978-3-319-23401-4_3.

[116] Klaus Weihrauch. *Computable Analysis: An Introduction*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2000. ISBN: 3-540-66817-9.

[117] Wikipedia contributors. *Test functions for optimization — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Test_functions_for_optimization&oldid=816115123.

[118] Geunyeol Yu, Jia Lee, and Kyungmin Bae. "STLmc: Robust STL Model Checking of Hybrid Systems Using SMT". In: *Computer Aided Verification*. Ed. by Sharon Shoham and Yakir Vizel. Cham: Springer International Publishing, 2022, pp. 524–537. ISBN: 978-3-031-13185-1.