Technical Report

UCAM-CL-TR-910
ISSN 1476-2986

Number 910

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Automated verification of continuous and hybrid dynamical systems

William Denman

July 2017

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted September 2014 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Clare College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Abstract

The standard method used for verifying the behaviour of a dynamical system is simulation. But simulation can check only a finite number of operating conditions and system parameters, leading to a potentially incomplete verification result. This dissertation presents several automated theorem proving based methods that can, in contrast to simulation, completely guarantee the safety of a dynamical system model.

To completely verify a purely continuous dynamical system requires proving a universally quantified first order conjecture, which represents all possible trajectories of the system. Such a closed form solution may contain transcendental functions, rendering the problem undecidable in the general case. The automated theorem prover MetiTarski can be used to solve such a problem by reducing it to one over the real closed fields. The main issue is the doubly exponential complexity of the back-end decision procedures that it depends on. This dissertation proposes several techniques that make the required conjectures easier for MetiTarski to prove. The techniques are shown to be effective at reducing the amount of time required to prove high dimensional problems and are further demonstrated on a flight collision avoidance case study.

For hybrid systems, which contain both continuous and discrete behaviours, a different approach is proposed. In this case, qualitative reasoning is used to abstract the continuous state space into a set of discrete cells. Then, standard discrete state formal verification tools are used to verify properties over the abstraction. MetiTarski is employed to determine the feasibility of the cells and the transitions between them. As these checks are reduced to proving inequalities over the theory of the reals, it facilitates the analysis of nonpolynomial hybrid systems that contain transcendental functions in their vector fields, invariants and guards. This qualitative abstraction framework has been implemented in the QUANTUM tool and is demonstrated on several hybrid system benchmark problems.

Publications

Parts of this dissertation have been published in the following refereed papers. All reported experimental results have been obtained solely by the author William Denman.

[72] William Denman, Mohamed H. Zaki, Sofiène Tahar, and Luis Rodrigues. Towards flight control verification using automated theorem proving. In *NASA Formal Methods*, LNCS 6617, pages 89–100. Springer, April 2011

[66] William Denman. Verification of nonpolynomial systems using MetiTarski. In *Proceedings of the 19th Automated Reasoning Workshop*, pages 31–32. School of Computer Science, The University of Manchester, April 2012

[67] William Denman. QUANTUM: Qualitative abstractions of non-polynomial models. In *Proceedings of the 27th International Workshop on Qualitative Reasoning*, pages 9–15, August 2013

[68] William Denman. Verifying nonpolynomial hybrid systems by qualitative abstraction and automated theorem proving. In *NASA Formal Methods*, LNCS 8430, pages 203–208. Springer, April 2014

[69] William Denman and César Muñoz. Automated real proving in PVS via MetiTarski. In *FM 2014: Formal Methods*, LNCS 8442, pages 194–199. Springer, May 2014

Acknowledgements

There are many people who have helped me to get here. I owe any and all success to them and thank them all equally by including their names here.

My parents for giving me a strong work ethic. Dad for teaching me to think like an engineer from an early age and letting me play with the 'puter. Mom for teaching me to *play the game*.

My wife Teresa for keeping me alive during my bachelor, master and doctoral degrees. Giving me tough love and being incredibly supportive when I was down in the PhD dumps. She encouraged me to head off to Cambridge even if that meant the potential end to our relationship. As it turned out we did a long distance relationship for 2 years, which we managed to get through! She gets much deserved credit here, having read over this entire dissertation twice!

To my friends in Canada who kept in contact during the last four years and came to visit me in Cambridge. Special thanks goes to Max and Danny. They might not remember, but the only reason I applied to Concordia for graduate school, after getting rejected from McGill, was based on their suggestions while we were playing Frisbee on Mount Royal. I remember that day very well indeed! They also kept in touch and provided much moral support even from across the pond.

To McGill University (and the corresponding Professors in the Electrical Engineering Department), how grateful I am for them not accepting me into their graduate program. For without their rejection, I would have never made it to Concordia. From there, I would never have left Montreal for Cambridge and consequently not travelled around the world from Hong Kong, to Singapore, to Taiwan, to Pasadena, to Austin, to Houston, to Palo Alto, to Trento, to Marktoberdorf during my PhD studies. I certainly would never made it into my current position in the financial industry. As the saying goes: "As one door closes, another one opens".

Dr. Sofiène Tahar for taking me on as a masters student at Concordia University and showing confidence in me. Introducing me to the world of research and the domain of formal methods.

Mohammed Zaki for taking me under his wing during the late stages of his PhD degree and teaching me some of the finer aspects of being a good researcher. How he made time for me, during the crunch of his own thesis writing, I still don't know how! I only realised this fact during the writing of this dissertation. Other good friends from Concordia who were quite supportive include Rajeev and Sanaz.

Behzad Akbarpour for connecting me with MetiTarski and Cambridge. I will never be able to express more gratitude for the collaboration with him which allowed me to come to the Computer Laboratory.

Various people in the Computer Laboratory were of much technical and moral support over the last few years. Grant Passmore helped me very early on and then too many times to count in between to get me through the most difficult parts of the PhD degree. James Bridge was a pleasure to have as an office mate, always there to make Monty Python Jokes and puns. He read over much of this dissertation several times. Nic Sultana also read over the entire thesis, for which I am grateful for.

During my time spent at SRI International I was helped immensely by N Shankar, Sam Owre and Ashish Tiwari. I shared an office with Martin Schäf who always kept me grounded when the PhD load was starting to get overwhelming. The work I conducted at SRI led to a collaboration with César Muñoz from NASA LaRC, who had a tremendous impact on the direction and quality of my research.

Significant research support was provided by Dr Paul Jackson and his student Andrew Sogonkon from the University of Edinburgh. Paul read over several iterations of this dissertation.

Thank you Dr. Eva Navarro Lopez and Dr. Mateja Jamnik for being wonderful examiners. Being tough, yet fair, their comments and critique definitely brought this dissertation to another level.

Last but not least, I have to give my greatest thanks to my supervisor Larry. He gave me support at every stage of the degree, provided pertinent feedback and was always available when I needed help. I am extremely grateful to have had him as a supervisor.

Contents

1 Introduction **13**
- 1.1 Motivation . . . 13
- 1.2 Formal Verification . . . 14
- 1.3 Models of Autonomous Dynamical Systems . . . 15
- 1.4 Choice of Automated Theorem Prover: MetiTarski . . . 17
- 1.5 Research Methodology . . . 17
  - 1.5.1 Preliminary Experiments . . . 17
  - 1.5.2 Hybrid System Verification . . . 18
- 1.6 Contributions . . . 19
- 1.7 Dissertation Outline . . . 20

2 Background **21**
- 2.1 Dynamical Systems . . . 21
- 2.2 Continuous Systems . . . 23
  - 2.2.1 Mathematical Model . . . 23
  - 2.2.2 Linear Dynamical Systems . . . 25
  - 2.2.3 Nonlinear Dynamical Systems . . . 27
  - 2.2.4 Lyapunov Functions . . . 29
  - 2.2.5 Barrier Certificates . . . 30
- 2.3 Hybrid Systems . . . 31
  - 2.3.1 Mathematical Model . . . 32
  - 2.3.2 Solutions of Hybrid Systems . . . 33
- 2.4 MetiTarski : An Automated Theorem Prover . . . 35
  - 2.4.1 High Dimensional Problems . . . 37
  - 2.4.2 Experimental Results . . . 38

3 Linear Continuous System Verification **39**
- 3.1 Preliminaries . . . 40
  - 3.1.1 Simulation Methods . . . 40
  - 3.1.2 Validated Solutions to Dynamical Systems . . . 41
  - 3.1.3 Algorithmic Reachability . . . 42

3.1.4 Theorem Proving and Deductive Reasoning . . . 43
3.1.5 MetiTarski Syntax . . . 46
3.1.5.1 Axioms . . . 46
3.2 Experimental Methodology . . . 47
3.3 Example: Linear System Reachability . . . 48
3.3.1 System Definition . . . 48
3.3.2 Analytical Solution . . . 49
3.3.3 MetiTarski Input . . . 50
3.3.4 Limiting the RCF time . . . 50
3.3.5 Recasting Technique . . . 53
3.3.6 Verification of Timed Reachability Properties . . . 54
3.3.7 Beyond Simple Regions . . . 55
3.3.8 Discussion . . . 56
3.4 Case Study: Flight Collision Avoidance . . . 57
3.4.1 Flight Dynamics . . . 58
3.4.2 Differential Axiomatisation . . . 58
3.4.3 Analytical Solution . . . 59
3.4.4 Safety Property Definition . . . 59
3.4.5 Experimental Setup . . . 60
3.4.5.1 Range Splitting . . . 61
3.4.6 Experimental Results . . . 61
3.5 Chapter Summary . . . 63

4 PVS/MetiTarski Integration **65**
4.1 Preliminaries . . . 66
4.1.1 Prototype Verification System . . . 66
4.1.2 Back-ends for Interactive Theorem Provers . . . 67
4.2 The PVS Strategy metit . . . 68
4.2.1 Strategy Application Example . . . 72
4.3 Experimental Results . . . 73
4.4 Chapter Summary . . . 74

5 Hybrid System Verification **75**
5.1 Background . . . 77
5.1.1 Verification by Abstraction . . . 77
5.1.2 Qualitative Reasoning . . . 79
5.1.2.1 Qualitative Simulation with QSIM . . . 82
5.1.2.2 QSIM Extensions . . . 85
5.1.3 Types of Abstraction . . . 87
5.1.4 Verification of Discrete State Abstractions . . . 88

5.1.4.1 Model Checking . . . 89
5.1.4.2 Simulation Relation . . . 91
5.1.5 Choice of Landmarks . . . 92
5.1.5.1 Generating Nonpolynomial Lyapunov Functions . . . 94
5.1.5.2 Generating Nonpolynomial Barrier Certificates . . . 97
5.2 Qualitative Abstraction of Hybrid Systems . . . 99
5.2.1 Abstracting the Continuous State Space . . . 100
5.2.2 Abstracting the Discrete State Space . . . 103
5.2.3 Analysis of the HybridSAL algorithm . . . 104
5.3 Initial Evaluation . . . 108
5.3.1 Picking the Functions in F . . . 109
5.3.2 Experimental Results . . . 110
5.4 Lazy Improvements to QUANTUM . . . 114
5.4.1 CEGAR Loop Implementation . . . 116
5.4.2 Multiprocessing . . . 117
5.4.3 Evaluation . . . 117
5.5 Chapter Summary . . . 117

6 Case Studies **121**
6.1 Bouncing Ball on a Sine Curve . . . 122
6.1.1 Model Description . . . 122
6.1.2 Experimental Results . . . 123
6.2 Two Tank System . . . 126
6.2.1 Model Description . . . 126
6.2.2 Experimental Results . . . 127
6.3 Self-Driving Car . . . 129
6.3.1 Model Description . . . 129
6.3.2 Experimental Results . . . 131
6.4 Modified Self-Driving Car . . . 133
6.4.1 Model Description . . . 133
6.4.2 Experimental Results . . . 134
6.5 Chapter Summary . . . 136

7 Related Work **139**
7.1 Modelling and Simulation of Hybrid Systems . . . 139
7.2 Formal Verification of Hybrid Systems . . . 142
7.2.1 Linear Dynamics . . . 143
7.2.2 Nonlinear Dynamics . . . 145
7.2.2.1 Discrete Abstraction Based Methods . . . 146
7.2.2.2 Interval Based Methods . . . 147

7.2.2.3 Non-Reachability Based Methods 149

8 Conclusion **151**
8.1 Contribution Summary 151
8.2 Future Work 153
8.3 Final Comments 153

A QUANTUM - Qualitative Abstractions of Nonpolynomial Models **155**
A.1 Model Input 155
A.2 Using QUANTUM 157

Bibliography **161**

CHAPTER 1

INTRODUCTION

1.1 Motivation

Safety is an engineer's most important concern. An aeroplane should never crash, a bridge must never fall down and a train must not derail. These dynamical systems are considered safe if there is a guarantee they will never exhibit unwanted behaviour. To obtain this guarantee, the physical dynamics are first translated into a mathematical model. This model is then numerically simulated to predict the behaviour of the original system. Even under the assumption that the model is a correct representation of the real system, safety guarantees obtained via simulation are inherently incomplete. This is because dynamical systems, defined by variables over the field $\mathbb{R}$, generally have an uncountably infinite number of input and output states. In many cases it is impossible to simulate every potential outcome.[1] This dissertation is concerned with using automated formal methods to construct a mathematical proof that, unlike simulation, guarantees that some types of systems are safe under all possible operating conditions.

A pertinent question is whether striving for a safety verification proof of a dynamical system is a realistic and useful goal. For instance, aeroplanes are able to fly without a formal safety analysis of their dynamics. This is acceptable because pilots are always in a supervisory role over the flight controls. In the case of an unexpected failure, the pilot can make the correct decisions to rectify the problem manually. On the other hand, consider the class of autonomous systems where the "human-in-the-loop" assumption is not valid. A computer is expected to make all decisions, adding a layer of complexity to the system. Consequently, simulation and rigorous testing can potentially fall short of providing the desired confidence in the design [195].[2] This is the key motivation behind researching a

[1] Dynamical system theory and control theory have difficulty with nonlinear and discontinuous systems.

[2] There is an entire field devoted to the design of controllers (control engineering) to ensure a dynamical system is safe. This dissertation takes an alternative viewpoint that relies on a machine constructed mathematical proof to provide a safety guarantee.

formal solution to the safety verification problem. Automated proof methods have the potential to enable engineers to meet the safety critical constraints of next generation autonomous systems [81].

1.2 Formal Verification

To address the incomplete verification of designs provided by simulation, where confidence in the result is based primarily on intuition and experience, formal methods have been developed as a rigorous solution. They are quite different from simulation based methods and include techniques rooted in logic, mathematics and computer science. In general, a specification is constructed and formal reasoning is used to show that a model under analysis follows from or is equivalent to the specification. There are two broad classes of methods to formally specify and verify dynamical systems, one based on transition systems and the other based on logical proof.

State space exploration methods are based on representing behaviour as a transition system and then verifying temporal properties over the resulting state machine. One popular example is model checking [53] where an exhaustive search of the state space is undertaken. For large designs containing many variables, most model checking techniques fail to produce an answer. This problem is called the state space explosion [49] and occurs when the amount of computer memory required to hold the state information is insufficient. Although memory constraints pose a significant problem for model checking, it is a preferred method because it is fully automatic.

Another set of methods model the behaviour of the systems in some type of logic. Deductive reasoning is then used to construct a proof. For example, in interactive theorem proving [141] a complete proof is constructed by hand (potentially with help from a proof assistant) using a base set of axioms and rules. Theorem proving environments include Isabelle [165], HOL [90], PVS [160], Coq [25] and several others. These tools are interactive and generally require an extremely high level of skill to construct the proofs.

There is another set of deductive techniques that are purely automatic but in general might not terminate. This class of automated theorem provers use automatic rules to infer that one logical formula follows from another, mechanically constructing the proof. The main difference between interactive and automated theorem provers is that automated theorem provers are constrained to a much smaller fragment of logical theories. This directly affects the types of conjectures that can be analysed. For instance, resolution theorem provers can handle sentences of First-Order Logic (FOL). Satisfiability Modulo Theories (SMT) solvers can deal with combinations of theories including arrays, linear and nonlinear arithmetic. Examples of automated theorem provers include Metis [114], Vampire [180], Z3 [65] and Yices [77]. The trade-off for automation is in the expressiveness of systems that can be modelled.

With the clear advantage of being able to consider all outcomes, formal verification seems like the ideal method to solve the safety verification problem. However, it is only the correctness of a design in relation to its formal specification that can be assured. The constructed system can still fail because there is no guarantee that the formal specification is correct. Additionally, fabrication and human errors can also potentially cause safety failures. Note, that neither of these problems would be picked up by simulation.

To date, there have been some industrial successes with applying formal methods to the verification of software and digital circuits [103]. However, the problem of applying formal verification to dynamical systems is still far from being solved [138]. A goal of this dissertation is to extend the success of discrete state methods to the safety verification of continuous[3] and hybrid dynamical systems.

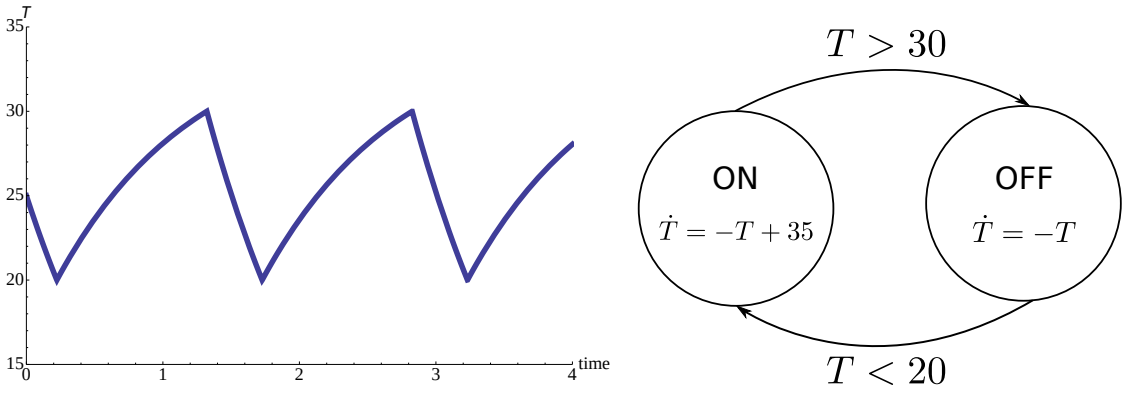
1.3 Models of Autonomous Dynamical Systems

Aeroplanes, trains and cars are all representative dynamical systems that play an important role in our lives. In each of these examples, autonomous control has been introduced to increase their operational safety. Autopilots in commercial aircraft are ubiquitous. Driverless trains (also known as automated people movers) are common in airports, rapid transit systems and freight railways [193]. Recent developments in self-driving car technology aim to drastically reduce the number of road fatalities [134]. Automation, however, comes at a cost. It introduces an extra layer of complexity, which can make it difficult to verify the safety of a system under all operating conditions [100]. To verify that an autonomous dynamical system is safe, we must choose an appropriate model for its behaviour.

A simple way to view an autonomous dynamical system is as a digital computer embedded within a continuously varying environment than can switch between distinct modes of operation [6]. Another name for this type of configuration is a *hybrid dynamical system*. Each mode of operation defines a different type of continuous evolution. When switching between modes the continuous variables of the system can jump to new values. By definition, there is a non-trivial interaction between continuous and discrete dynamics. Canonical examples include bouncing balls, thermostats and switched circuits.

Example 1.1 (Simplified Hybrid System)**.** A thermostat has two modes of operation, heating (on) and cooling (off). In Figure 1.1a the trajectory of a thermostat is plotted, with temperature (T) versus time. The initial temperature is set to $25°$C. When the temperature rises above $30°$C the thermostat shuts off; when the temperature decreases to below $20°$C, the thermostat turns on. In each mode, continuous evolution is described by a differential equation.

[3] In this dissertation when unqualified the word *continuous* refers to a *continuous-time* dynamical system with a smooth vector field



(**a**) Thermostat trajectory (**b**) Simplified hybrid automaton

Figure 1.1: Hybrid system

A common formalism used for modelling hybrid systems is the *hybrid automaton* [5]. An example of one that models the thermostat is shown in Figure 1.1b. Hybrid automata are quite general and can express many complex dynamical systems; their verification is however quite difficult. A well known theoretical result is that the reachability question, *"Does there exist a run of the hybrid automaton that can reach an unsafe state?"*, is undecidable [106]. Standard transition system verification techniques will therefore not work.

One alternative approach to verify the reachability properties of dynamical systems is based on discretising the continuous state space into distinct cells. Then, using properties of the underlying vector field, identify transitions between cells. This abstraction process produces a finite-state automaton that admits efficient formal verification algorithms. Several examples of this methodology, which differ in how the cells are constructed, include: hyper-rectangles (Maler and Batt [139], Carter and Navarro-López [40], Carter [41]), piecewise-linear functions (Asarin et al. [17]), linearisations (Dang et al. [60]), qualitative abstractions (Tiwari [209]), Lyapunov functions (Sloth and Wisniewski [198]) and many others. In this dissertation, I describe a similar discretising approach for hybrid automata, but with two distinguishing features.

- The cell partitioning functions are allowed to contain nonpolynomial terms.
- The feasibility of cells and the transitions between them are determined and verified using automated theorem proving.

Allowing nonpolynomial terms greatly expands the class of systems that can be verified via this type of abstraction method. Being able to handle nonpolynomial functions is particularly important since hybrid systems usually interface with the physical world. Transcendental and special functions naturally arise in modelling their behaviour. Angular measurements might involve sine, cosine and related transcendental functions. Several

types of friction or drag can involve the exponential function. Only a handful of existing techniques can deal with this class of *nonpolynomial hybrid systems*.[4] This makes the methods described in this dissertation highly relevant and important to the field of dynamical system verification. Furthermore, when compared to other state-of-the art methods, employing a discretising method allows for a time unbounded verification result rather than a bounded one.

1.4 Choice of Automated Theorem Prover: MetiTarski

MetiTarski [4] is an automated theorem prover for arithmetical conjectures involving transcendental functions. It has been previously successful at proving arithmetical theorems that arise from the verification of analogue circuits [70], aircraft stability [72] and linear hybrid systems [3]. Based on its successful track record, I have chosen to use MetiTarski as the verification engine for constructing abstractions of hybrid systems.

A difficulty to overcome when using an automated theorem prover for dynamical systems verification is the *dimensionality gap*. Real (nonlinear) systems can generally have tens to hundreds of continuous real variables, but automated theorem provers tend to struggle on conjectures with more than four or five continuous variables. This dissertation attempts to bridge this gap by developing sound abstraction methods to reduce the complexity of the models while maintaining the behaviours of interest. I also investigate the development of techniques that can aid MetiTarski in finding a proof for high dimensional problems.[5]

1.5 Research Methodology

The goal of this dissertation is to leverage the power of automated theorem proving for the development of dynamical system verification techniques. To be able to properly use MetiTarski for this task, I have conducted several preliminary experiments to quantitatively measure its capabilities and limitations. These initial experiments are particularly important for estimating the size (in terms of the number of continuous variables) of dynamical systems that can be verified.

1.5.1 Preliminary Experiments

The first part of this dissertation presents two preliminary experiments that are used to evaluate MetiTarski's ability to verify properties of dynamical systems (see Figure 1.2).

[4]This is the class of switched and discontinuous continuous-time hybrid systems that are defined by systems of differential equations that contain nonpolynomial terms in their right hand sides.

[5]For automated theorem provers, a high dimensional problem contains 8-11 continuous variables of state

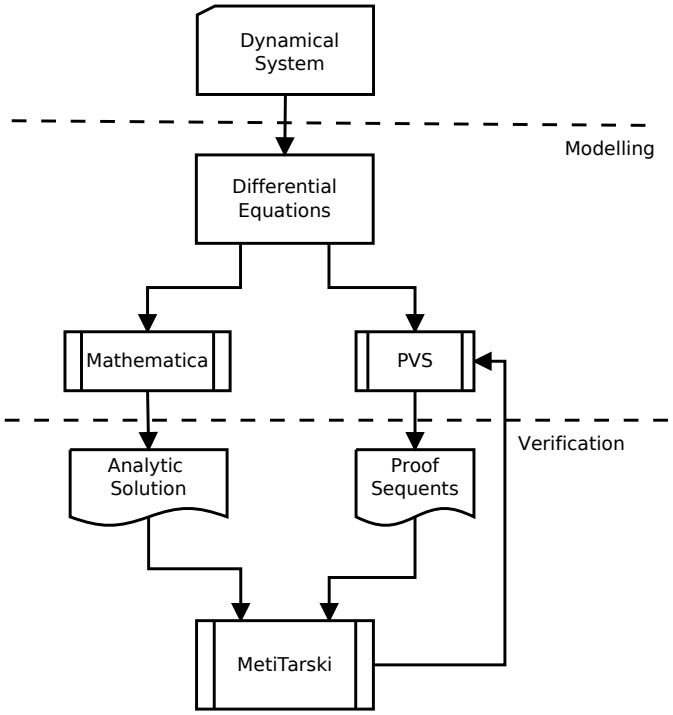


Figure 1.2: Methodology for continuous system verification

The first way to obtain experimental problems is by using the computer algebra system Mathematica to symbolically calculate the solutions of linear dynamical systems. These solutions typically contain transcendental and special functions. MetiTarski is used to prove the safety of the systems by showing, for ranges of initial conditions, that all unsafe states are unreachable. This type of analysis can been seen as a form of exhaustive simulation.

The second way to obtain experimental problems is from an integration of MetiTarski and the interactive theorem prover PVS. During a PVS proof session, sequents containing transcendental terms can be sent to MetiTarski for analysis. If successfully proved by MetiTarski, the result is fed back into PVS so the construction of the proof can continue. A single PVS proof script can make many invocations of MetiTarski. The integration of MetiTarski and PVS also opens access to the wide range of interesting problems contained in the NASA PVS library [1].

1.5.2 Hybrid System Verification

The second part of this dissertation deals with the verification of nonlinear continuous-time hybrid systems (see Figure 1.3). Since it is generally not possible to obtain closed form solutions for these types of systems, MetiTarski cannot be used to directly verify properties of these types of systems. I propose extensions of a qualitative abstraction method initially developed by Tiwari [209] to use nonpolynomial functions to cut up the state space into discrete cells.

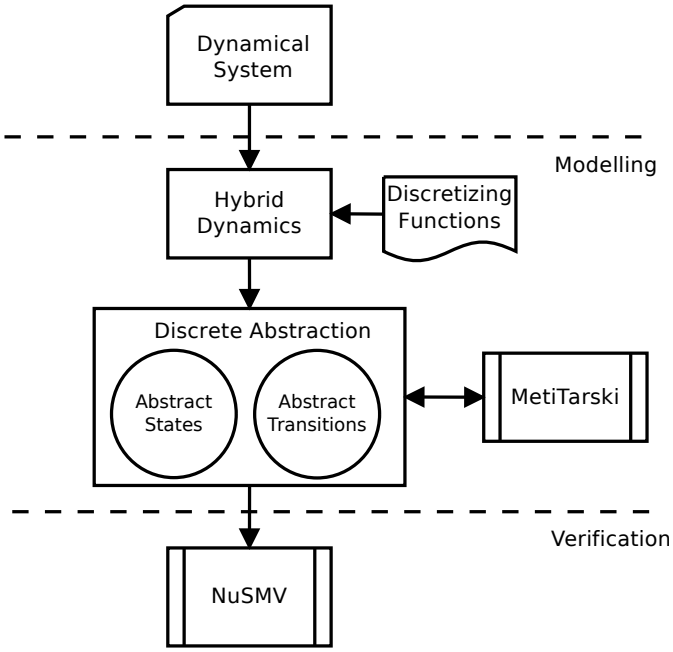


Figure 1.3: Methodology for hybrid system verification

Several methods are used to generate the functions that discretise the state space. One method takes higher order derivatives of the underlying vector fields to generate a series of increasingly precise functions. The other is based on using convex optimisation to generate a family of invariant functions that greatly reduce the size of the abstract state space. Once the abstraction functions are chosen, MetiTarski is used to determine if the generated cells, described by a conjunction of first order predicates, are feasible and to determine all transitions between cells.

Once the abstract system is constructed and all infeasible states and transitions are removed, verification can proceed. The model checker NuSMV [47] is used to prove temporal safety properties of the discrete state abstraction. Since the qualitative abstraction process is provably sound, the resulting finite state transition system is guaranteed to be an over-approximation of the original hybrid system. When NuSMV is able to prove a safety property of the abstract system, the original system is guaranteed to be safe as well.

1.6 Contributions

This dissertation is concerned with the application of automated theorem proving to the verification of some types of dynamical systems. Highlighted contributions are:

1. A set of techniques that simplify the proof search for conjectures arising from linear and nonlinear dynamical system verification problems.

2. An integration of automated and interactive theorem proving for conjectures over the real numbers.

3. Extending a qualitative abstraction algorithm to the automated verification of non-polynomial hybrid automata.

1.7 Dissertation Outline

The remainder of the dissertation is organised as follows:

Chapter 2

Presents the necessary mathematical background for this dissertation. It covers continuous and hybrid dynamical system theory, which sets the stage for the following chapters.

Chapter 3

Presents the initial experiments on continuous system verification. The focus is on the analysis of linear systems that admit closed form solutions. An aircraft collision avoidance example is analysed in depth.

Chapter 4

Presents the integration of MetiTarski as a decision procedure with the PVS interactive theorem proving environment. Includes an analysis between the internal decision procedures of PVS and MetiTarski.

Chapter 5

Presents the methodology for hybrid system verification. It details the background of the qualitative abstraction method developed by Tiwari, followed by the extensions developed as part of this dissertation. Important highlights include an analysis of the soundness proof of the abstraction method. I demonstrate several ways to generate the functions used by the abstraction algorithm.

Chapter 6

Presents three case studies of the application of the abstraction methodology on nonlinear hybrid systems, and the results are discussed.

Chapter 7

Presents the related work on hybrid dynamical system verification techniques and related modelling methods.

Chapter 8

Summarises and presents some conclusions and ideas for future work.

CHAPTER 2

Background

This chapter reviews the relevant background theory for this dissertation. The first two sections provide a high level overview of continuous and hybrid dynamical systems, with emphasis on showing how real world phenomena can be represented by a mathematical model. This leads naturally to the introduction of two important functions from control engineering theory, Lyapunov functions and barrier certificates, which play an important role in the abstraction methods described in Chapter 5. The chapter concludes with a discussion on the architecture of the automated theorem prover MetiTarski, which is used in the dynamical system verification experiments of this dissertation.

Only previous experience with calculus, especially knowledge of total and partial derivatives, is assumed. More in-depth coverage of dynamical system theory can be found in books by Khalil [123], Boyce and DiPrima [31] and Sastry [190].

2.1 Dynamical Systems

A dynamical system is a term used to classify any process that changes over time. This definition, being quite abstract, allows a wide variety of phenomena to be described: chemical reactions, aeroplane dynamics, a beating heart and electronic circuits are all examples of dynamical systems.

There are two classes of dynamical systems that are relevant to this dissertation: continuous-time and hybrid.[1] Continuous-time systems define the behaviour of a system as it changes smoothly with respect to time (e.g. the flow rate of a river). Hybrid systems have both continuous and discrete behaviours present (e.g. a thermostat controlling the temperature of a room can switch discretely between two distinct modes of continuous operation, heating and cooling). A hybrid system can both flow smoothly with respect to time and jump discretely with respect to its state. In this dissertation, I have chosen to focus on hybrid systems, because they are an appropriate model for many

[1] Hybrid refers here to both continuous-time and discrete-time hybrid systems.

macroscopic systems (e.g. machinery, vehicles, spacecraft) that operate in the real world. The Newtonian dynamics of such systems can be represented as a continuous-time system. The control decisions can be modelled as a discrete-event finite state machine. This combination of continuous-time dynamics plus discrete control therefore motivates the use of a hybrid system model.

There are particular dynamical systems, such as the class of continuous linear systems, that admit closed form analytical solutions. The solutions are exact and can be plotted to verify the behaviour of the system. These closed form solutions will generally be expressed in terms of elementary functions. It follows that this class of systems can be verified automatically by tools, such as MetiTarski, that can reason about elementary, transcendental and other special functions.

Most real world phenomena cannot be modelled by linear equations alone; their behaviour can only be properly represented using nonlinear terms. For example, a swinging pendulum is one of the simplest dynamical systems, it however requires a sine component in its model. In direct contrast to linear systems, nonlinear systems cannot in general be solved analytically and therefore other methods must be used for verification. Two such methods are numerical simulation and qualitative analysis.

Numerical simulation can be used to algorithmically compute the solutions to nonlinear dynamical systems. In this method a single initial condition is chosen and a single trajectory of the system is computed. This is repeated for many initial conditions, until the number of trajectories obtained describe the behaviour of the system to the required degree.

Care must be taken when using numerical simulation. Errors may build up because of the limited precision of real numbers in a computer. This can cause the computed solutions to deviate from the true value of the system's trajectories. Therefore, the results from numerical simulation of nonlinear systems are generally only valid for short intervals of time [92]. Consequently, numerical simulation cannot be completely trusted when trying to verify the long-term behaviour of nonlinear dynamical systems.

Instead of trying to numerically compute solutions, qualitative methods alternatively take into account the general evolution of a dynamical system rather than focusing on specific trajectories. One qualitative analysis method is linearisation. In this method, the nonlinear dynamics are approximated by a linear model at a specific point called an equilibrium point. Dynamical system theory shows that in a close neighbourhood of that point, the long-term stability of the linearised system will be equivalent to the original nonlinear system [31]. Closed form solutions of the linearised model can be calculated and used to simulate the behaviour of the original system within this neighbourhood.

One potential problem is that behaviours critical to the operation of the original system might occur far away from the linearisation point and subsequently can be lost during the process [123]. This means drawing conclusions about the global behaviour of a non-

linear system via linearisation, is impossible. An alternative qualitative method, based on searching for energy-like functions (Lyapunov functions), can provide global guarantees on the trajectories of the system and thus is more useful in a verification context.

2.2 Continuous Systems

2.2.1 Mathematical Model

A dynamical system can be viewed as a box with a set of inputs and and a set of outputs. Its internal state is defined by a vector $\boldsymbol{x}$ that holds the current values of the variables of the system (also called the *state variables*). A state modification function f takes the current state vector as an input and returns the next value of each of the state variables.

For continuous-time dynamical systems, evolution progresses as a smooth function of time. In this case, the function f describes how the system *flows* from one time instance to the next. The simplest way to model this change is by using *ordinary differential equations* (ODEs). The prefix *ordinary* is used since the differential equations vary with respect to a single independent variable (time in this case) and its derivatives.

Definition 2.1 (Continuous Dynamical System)**.** A n-dimensional continuous dynamical system is represented by the state vector $\boldsymbol{x}(t) \in \mathbb{R}^n$, input-vector $\boldsymbol{u} \in \mathbb{R}^m$ and a set of coupled first-order differential equations of the form

$$\begin{aligned} \dot{x_1} &= f_1(\boldsymbol{x}(t), t, \boldsymbol{u}), \\ \dot{x_2} &= f_2(\boldsymbol{x}(t), t, \boldsymbol{u}), \\ &\vdots \\ \dot{x_n} &= f_n(\boldsymbol{x}(t), t, \boldsymbol{u}), \end{aligned} \tag{2.1}$$

where $\dot{x_i}$ denotes the derivative of x_i with respect to time t and $f_i : \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}$, where f_i is a smooth.

In this dissertation, two standard assumptions are made about the type of systems under analysis.[2] First, the systems are *autonomous* or *time invariant*: the f_i definitions do not explicitly depend on time. Second, it is assumed that there are no inputs $\boldsymbol{u}$ to the system or the inputs are held constant. These restrictions are commonly used to simplify the analysis of dynamical systems. They represent the natural assumption that a device should always behave in the same way regardless of when it starts functioning. Taking

[2]The resulting models are adequate for representing the operation of simple parts of more complex systems.

these assumptions into consideration reduces the system of equations (2.1) to

$$\begin{aligned} \dot{x_1} &= f_1(\boldsymbol{x}), \\ \dot{x_2} &= f_2(\boldsymbol{x}), \\ &\vdots \\ \dot{x_n} &= f_n(\boldsymbol{x}). \end{aligned} \tag{2.2}$$

Definition 2.2 (Solution of a System of Differential Equations, [31, p. 342])**.** A system of differential equations (2.2) with initial conditions $\boldsymbol{x}_0 = (x_{10}, x_{20}, \ldots, x_{n0})$ is said to have a solution on the interval I : $\alpha < t < \beta$, with $\alpha < 0 < \beta$, if there exists a set of n functions

$$x_1 = \phi_1(t), x_2 = \phi_2(t), \ldots, x_n = \phi_n(t) \tag{2.3}$$

that are continuously differentiable at all points in the interval I, satisfying $\phi_1(0) = x_{10}, \phi_2(0) = x_{20}, \ldots, \phi_n(0) = x_{n0}$ for each equation in system (2.2) for all $t \in]\alpha, \beta[$. The solution can also be referred to as a trajectory or flow, which is the path followed by the variables of the system.

Just because a system of differential equations can be defined does not guarantee that a solution, in the sense of Equation (2.3), exists. Systems of differential equations, like algebraic systems, can have a single solution, multiple solutions or no solution at all. Even if there is a single solution, it might behave in ways that are not physically realisable (e.g. blowing up to infinity in finite time). Fortunately, strict continuity conditions can be enforced on a system's vector fields to ensure that a solution exists, is unique and is well behaved.

Definition 2.3 (Lipschitz continuity [123, p. 88])**.** A function $f : \mathbb{R}^n \to \mathbb{R}^n$ is called Lipschitz continuous if there exists a λ such that for all $x, \hat{x} \in \mathbb{R}^n$

$$\|f(x) - f(\hat{x})\| < \lambda \|x - \hat{x}\| \tag{2.4}$$

This definition enforces a strong form of continuity on the function f, ensuring that it will not vary faster than a real constant λ. For instance, this implies any f that has an infinite slope is not Lipschitz continuous [123].

Theorem 2.1 (Existence and uniqueness, [123, p. 93])**.** If each f_i is Lipschitz continuous, then the system defined by Equations (2.2) has a unique solution $\phi(t) : [0, T] \to \mathbb{R}^n$ for all $T \geq 0$ and all $x_0 \in \mathbb{R}^n$.

In all the examples in this dissertation I only consider such *well behaved* dynamical systems that have Lipschitz continuous vector fields. By assuming that the models of interest accurately capture the behaviour of the system under analysis, I can focus on verification research questions, rather than on modelling ones.

2.2.2 Linear Dynamical Systems

Linear dynamical systems can take the form of

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + \boldsymbol{b} \tag{2.5}$$

where A is an $n \times n$ matrix and $\boldsymbol{b}$ is an n vector. It is convenient[3] in our discussions here to consider the class of linear homogeneous systems $\dot{\boldsymbol{x}} = A\boldsymbol{x}$, where the $\boldsymbol{b}$ vector is equal to zero.

The solutions[4] of linear homogeneous systems are linear combinations of $e^{\lambda_1 t}$, ..., $e^{\lambda_n t}$, where λ_1, ..., λ_n are the eigenvalues of the matrix A. Recall that the eigenvalues of a matrix are the roots of the characteristic equation $\det(A - \lambda I)$, where det is the determinant of a matrix.

Example 2.1 (Linear Circuit)**.** Consider for example the electric circuit in Figure 2.1. R, L and C represent a resistor, inductor and capacitor respectively. The state vector $\boldsymbol{x} = [V, I]^T$ represents the voltage across and current through the capacitor.

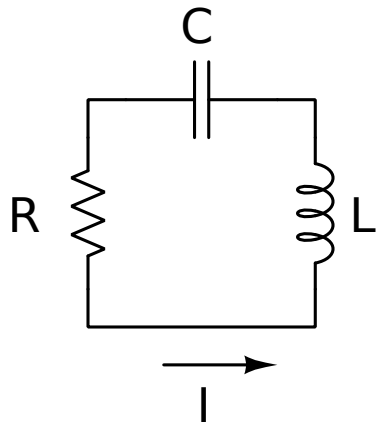


Figure 2.1: Linear RLC circuit

Solving the circuit using Kirchoff's current and voltage laws gives the following definition of the system in the form of Equation (2.5), with $b = 0$:

$$\begin{bmatrix} \dot{V} \\ \dot{I} \end{bmatrix} = \begin{bmatrix} 0 & -1/C \\ 1/L & -R/L \end{bmatrix} \begin{bmatrix} V \\ I \end{bmatrix}$$

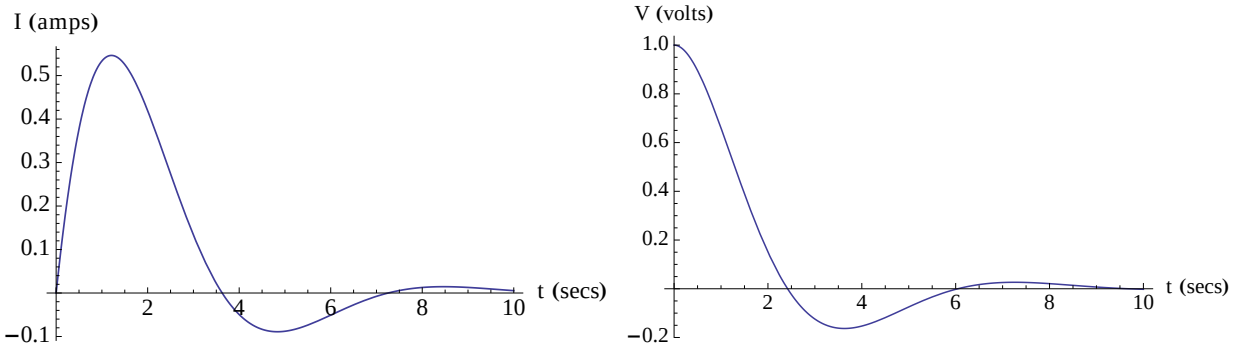
[3]The long-term qualitative behaviour of a non-homogeneous system is similar to that of the equivalent homogeneous system. However, the solutions will have a different form. We assume here that the matrix A is diagonisable. This assumption does not change the long-term qualitative nature of the linear system.

[4] $e^{At}x_0$ is the solution of a linear system of differential equations of the form $\dot{\boldsymbol{x}} = A\boldsymbol{x}$. Assuming A is diagonisable then $e^{At} = Se^{\Lambda t}S^{-1}$, where the columns of the matrix S are n linearly independent eigenvectors of A. The matrix Λ is a diagonal matrix of the eigenvalues of the system.

Taking $R = 1\Omega$, $L = 1H$, $C = 1F$ and the initial values as $I_0 = 0A$ and $V_0 = 1V$, a computer algebra system, such as Mathematica, can be used to compute a closed form solution. The computed solutions of the system are

$$V(t) = \frac{1}{3}e^{-t/2}\left(\sqrt{3}\sin\frac{\sqrt{3}t}{2} + 3\cos\frac{\sqrt{3}t}{2}\right),$$
$$I(t) = \frac{2e^{-t/2}\sin\frac{\sqrt{3}t}{2}}{\sqrt{3}}.$$

Looking at the plots below, we see that the solutions can be visualised as the trajectories of the state variables. For the specific circuit parameters, the circuit in Figure 2.1 starts to oscillate, but then dies out. As there is no external supply of energy (voltage or current), this is the expected behaviour.



(**a**) Solution for the current $I(t)$ (**b**) Solution for the voltage $V(t)$

Figure 2.2: Trajectories of the current and voltage

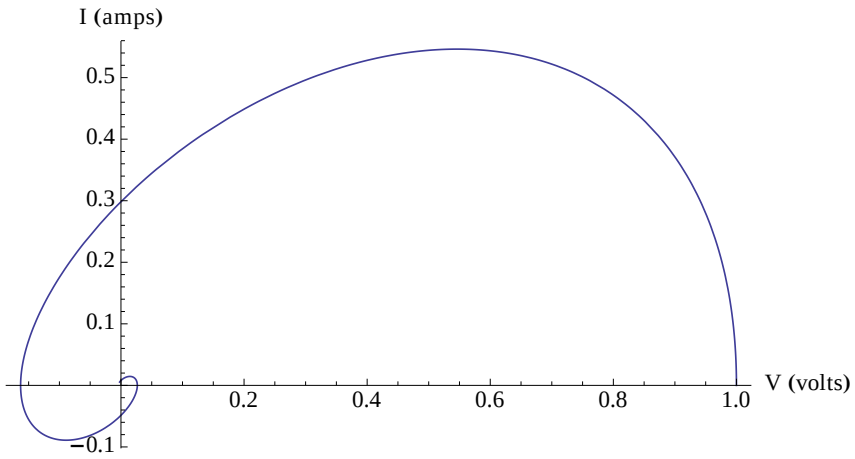


Figure 2.3: Phase plot of $I(t)$ vs $V(t)$

The sign of the real part of the eigenvalues, $Real\{\lambda_i\}$, of a linear system governs the long-term qualitative behaviour of the system's trajectories. When the eigenvalues are complex, $\lambda_i = a \pm bi$, then the solutions will be of the form $e^{(a \pm bi)t}$, which can be expressed equivalently in terms of $e^{at} \sin bt$ and $e^{at} \cos bt$. Several important cases are as follows:

1. If $\forall i : Real\{\lambda_i\} < 0$, then the $e^{\lambda_i t}$ terms in the solutions decrease to zero as $t \to \infty$ and the system is therefore guaranteed to converge to a specific point. We will see in the next section that this specific point is called an *equilibrium point*.

2. If $\exists i : Real\{\lambda_i\} > 0$, then the $e^{\lambda_i t}$ terms in the solutions will, under most circumstances, grow unbounded as $t \to \infty$.

3. If $\forall i : Real\{\lambda_i\} = 0 \wedge \exists i : Imag\{\lambda_i\} \neq 0$, the solutions will contain sin and cos terms causing the trajectories to oscillate, neither approaching zero or infinity.

The eigenvalues of the system described in Example 2.1 are:

$$\left\{ -\frac{1}{2} - \frac{i\sqrt{3}}{2}, -\frac{1}{2} + \frac{i\sqrt{3}}{2} \right\}.$$

As each eigenvalue has a negative real part, then the system should head to zero, which is the equilibrium point of the system. Since each eigenvalue has a non-zero imaginary part, it is expected that the solutions will oscillate. The phase plot in Figure 2.3, which is a parametric plot of the state variables, clearly shows the expected qualitative behaviour of the system.

The information provided by the analysis of the eigenvalues is an integral part of qualitative methods used for the analysis of nonlinear systems that do not admit closed form solutions. When a nonlinear system is approximated by a linear system, then the sign of the eigenvalues of the approximation can give a reasonable characterisation of the behaviour of the original system. This is discussed in the next section.

2.2.3 Nonlinear Dynamical Systems

An example of a nonlinear dynamical system is the pendulum of Figure 2.4. A rod of length L is attached to a ball of mass m. As the ball swings, the angle θ between the rod and the vertical changes. The angular velocity (rotational speed in the tangential direction) $\omega(t)$ is equivalent to the change of the angle θ or $\frac{d\theta}{dt}$. Acceleration, velocity and position of the ball are related by $a = \dot{v} = \ddot{x}$. The arc-distance travelled by the ball is $x = \theta L$. The effective force returning the ball to the center is $mg \sin\theta$ minus an air friction term dependent on ω. The differential equations of the system can be derived

from Newton's Second Law $F = ma$. Taking $\frac{F}{m} = a$, $a = \ddot{x} = (\ddot{\theta L}) = \dot{\omega} L$ gives the system in state space form

$$\dot{\theta} = \omega \tag{2.6a}$$

$$\dot{\omega} = -\frac{g}{L}\sin\theta - b\omega \tag{2.6b}$$

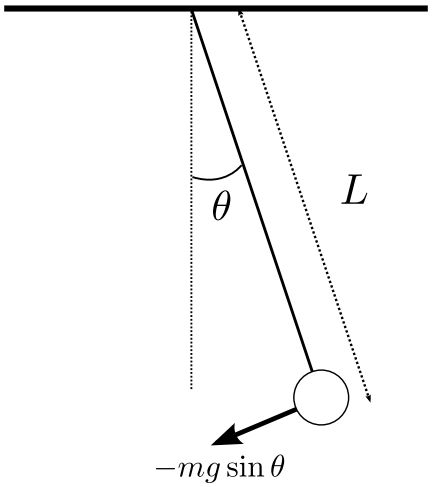


Figure 2.4: Regular pendulum

The pendulum is concisely described by two simple differential equations. Due to the nonlinear term in Equation (2.6b), it is not possible to obtain an exact analytic solution with respect to time for either of the state variables $\theta(t)$ or $\omega(t)$.

Since it is not possible to obtain closed form analytical solutions to nonlinear systems, alternative qualitative methods can be used to obtain an understanding of the general long-term behaviour of the system. This qualitative analysis is usually concerned with the equilibrium points of the system. By investigating the properties of equilibrium points, much can be understood about the properties of the system without resorting to numerical methods for computing solutions.

Definition 2.4 (Equilibrium Point)**.** An equilibrium point of a system of the form $\dot{x} = f(x)$, is a location in the continuous state space $\widetilde{x} \in \mathbb{R}^n$ where $f(\widetilde{x}) = 0$. It is called *stable* if every trajectory of the system beginning near the equilibrium point remains near the equilibrium point for all time. It is called *asymptotically stable* if it is stable and every trajectory beginning near the equilibrium point tends to it as $t \to \infty$. It is considered *unstable* otherwise.

It was discussed in Section 2.2.2 how the long-term behaviour of a linear system could be determined from an analysis of its eigenvalues. It can now be seen that if all the eigenvalues have a negative real part, then the system converges to an equilibrium point.[5] It is possible to characterise the type of equilibrium point (stable, asymptotically stable or unstable) just by analysing a linear system's eigenvalues.

2.2.4 Lyapunov Functions

The analysis of the eigenvalues of a system can provide a general idea of the behaviour of linear or linearised dynamical systems. However, as previously shown, there are cases where this process fails (i.e. when $\forall i$: Real$\{\lambda_i\}$ = 0). In this case, another qualitative method is used that is based on the following simple concept: if a mechanical system is losing energy, it must eventually stop. The following example demonstrates the general reasoning behind this method.

Example 2.2. Consider the nonlinear pendulum in Figure 2.4 that experiences air resistance as it rocks back and forth. Take a function $V(x)$ to be the total (kinetic plus potential) energy of the system. It is clear that when the pendulum is displaced, energy is put into the system causing $V(x)$ to increase and because of this, $V(x) > 0$. The energy of the system will only be zero when the pendulum has stopped swinging and is hanging straight down at position 0, therefore $V(0) = 0$ and $\dot{V}(0) = 0$. The only other position where $\dot{V}(0) = 0$ is when the pendulum changes direction. In this case the system will immediately enter a state where the energy continues to decrease. It cannot stay in the state $\dot{V}(x) = 0, x \neq 0$ forever. In all other positions, the system continuously loses energy due to friction, therefore $V(x)$ is always decreasing, which implies that $\dot{V}(x) < 0$. From this description we can conclude that the pendulum will eventually stop and end up hanging straight down.

Theorem 2.2 (Lyapunov Function [123, p. 114])**.** Let the system $\dot{x} = f(x)$. For an equilibrium point located at the origin $(x = 0, x \in \mathbb{R}^n)$, let $V(x)$ be a continuously differentiable function $V : \mathbb{R}^n \to \mathbb{R}$ such that

$$V(x) > 0 \qquad \text{for } x \neq 0 \tag{2.7a}$$

$$V(0) = 0 \tag{2.7b}$$

$$\frac{dV(x)}{dt} \leq 0 \qquad \text{for all } x \tag{2.7c}$$

[5] In the linear case, when all the eigenvalues have a negative real part the equilibrium point is guaranteed to be asymptotically stable.

Then, $x = 0$ is a stable equilibrium point. If $\frac{dV(x)}{dt} < 0$ for all $x \neq 0$, then $x = 0$ is an asymptotically stable equilibrium point. A continuously differentiable function $V(x)$ satisfying equations (2.7a) and (2.7b) is called a candidate Lyapunov function. The existence of a candidate Lyapunov function is a sufficient condition for the stability of an equilibrium point.

There is no constraint forcing the Lyapunov $V(x)$ to be the energy of the system. Any function satisfying the conditions of Theorem 2.2 can be used. The caveat is that finding a Lyapunov function in general can be quite difficult and can even be more difficult than solving the original system of differential equations. For nonlinear systems, finding a Lyapunov function usually requires a search. Fortunately, there are several advanced methods based on the sum of squares decomposition and convex optimisation that make the search for Lyapunov functions of nonlinear systems tractable [163]. These methods have been implemented in a MATLAB package called SOSTOOLS [162] and are used in Chapter 5 for constructing abstractions of hybrid systems.

2.2.5 Barrier Certificates

Lyapunov functions can be used to understand the qualitative nature of equilibrium points and to determine the stability of these equilibria. They are useful because no computation of the trajectories of the system is required, which is essential when the system of differential equations contains nonlinear terms. *Barrier certificates* generalise Lyapunov functions to arbitrary locations in the state space. This method is particularly useful for proving the safety of dynamical systems.

Theorem 2.3 (Barrier Certificate [174])**.** A dynamical system is defined over a state space $\chi \subseteq \mathbb{R}^n$. There is a region $\chi_u \subseteq \mathbb{R}^n$ of unsafe states and a region $\chi_o \subseteq \mathbb{R}^n$ of initial states. Consider a continuous function of state $B(x) : \mathbb{R}^n \to \mathbb{R}$ that is differentiable with respect to its argument. If,

$$B(x) > 0 \qquad \text{for all } x \in \chi_u \tag{2.8a}$$

$$B(x) \leq 0 \qquad \text{for all } x \in \chi_o \tag{2.8b}$$

$$\frac{dB(x)}{dt} \leq 0 \qquad \text{for all } x \in \chi \tag{2.8c}$$

then all trajectories of the system starting in χ_o will never reach the states in χ_u and therefore the system is safe.

Consider the example of the nonlinear pendulum, with its phase plot in Figure 2.5. Each blue arrow line represents a separate trajectory of the system. For all trajectories starting in the green box, the existence of the black solid line (a barrier certificate that meets all the conditions of Theorem 2.3), is a guarantee that the red box of unsafe states

is unreachable. Notice how each trajectory crosses the dark line in one direction, heading inwards. This behaviour further highlights the fact that it is impossible for any trajectory starting inside the barrier to leave the region through the barrier.

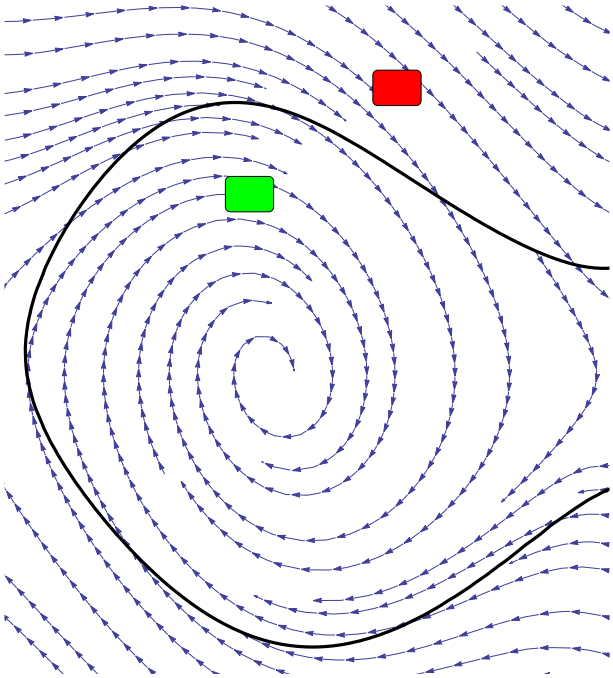


Figure 2.5: Barrier certificate

Since the form of a barrier certificate matches closely that of a Lyapunov function, the search for barrier certificates can also be performed efficiently with the SOSTOOLS software [173].

2.3 Hybrid Systems

There are many physical processes that cannot be modelled by a purely continuous-time dynamical system that has a smooth trajectory. Take for example an electric circuit that contains a switch. When the switch is opened or closed, the structure of the circuit is altered and consequently the solutions of the system change abruptly. Such a *switched system* is an example of a *hybrid dynamical system*. Hybrid systems, in practice, arise from the combination of finite state logic which governs a continuous process. The finite state logic models the computerised control and the physical phenomena is modelled by differential or difference equations.

There are a wide variety of hybrid systems that can be defined by the type of switching and jumps that can take place. Switching here refers to how the continuous-time vector field changes discontinuously when moving into certain regions of the state space. Jumping[6] refers to how the continuous-time trajectory changes discontinuously when moving

[6]Also known as a reset.

into a specific region of the state space. A hybrid system might have both or just one of the types of discontinuity. For example, systems with friction can have discontinuities in the vector field only. While an impact system, such as a bouncing ball, will be defined by a single vector field but its velocity will jump to a new value with each bounce.

2.3.1 Mathematical Model

Hybrid systems involve the interaction of continuous-time and discrete-time dynamics. On the one hand, there is a continuous flow that is best described by differential equations. On the other, there is the possibility of discrete jumps that would be best described using a finite state machine. One modelling framework (several others are described in Chapter 7) for hybrid systems which properly captures the evolution in time of a set of discrete and continuous variables is the *hybrid automaton* [7, 135].

Definition 2.5 (Hybrid Automata, Lygeros [135])**.** A hybrid automaton H is a collection $H = (Q, X, f, Init, Inv, E, G, R)$ where

- $Q = \{q_1, q_2, \dots\}$ is a set of discrete states
- $X \subseteq \mathbb{R}^n$ is a set of continuous states
- $f : Q \times X \to \mathbb{R}^n$ is a vector field
- $Init \subseteq Q \times X$ is a set of initial states
- $Inv : Q \to 2^X$ is an invariant set for each discrete state
- $E \subseteq Q \times Q$ is a set of edges
- $G : E \to 2^X$ provides guard conditions for each edge
- $R : E \times X \to 2^X$ is a reset map

For each discrete state q_i there is a set of continuous states $Inv(q_i)$ assigned to it. The function f defines the set of differential equations governing the flow of the continuous variables for each discrete state q_i. $(q_i, x_i) \in Q \times X$ is called the state of the hybrid automaton.

Example 2.3 (Bouncing Ball on a Sine Curve, Ishii et al. [117])**.** The hybrid automaton in Figure 2.6 models a ball bouncing on a sine shaped curve. A single simulated trajectory of the bouncing ball hybrid automaton is shown in Figure 2.7.

- $Q = \{q_1\}$, the bouncing ball has one discrete mode q_1 = "falling".
- $X = \mathbb{R}^4 = (p_x, p_y, v_x, v_y)$, where p_x and p_y are the position of the ball, v_x and v_y are the respective velocities.
- $f(q_1, (p_x, p_y, v_x, v_y)) = (\dot{p_x}, \dot{p_y}, \dot{v_x}, \dot{v_y})$, are the respective velocities and acceleration in the x and y dimensions.

- $Init = (q_1, \{(p_x, p_y, v_x, v_y) \in \mathbb{R}^4 \mid \sin p_x - p_y < 0, p_x = p_{x0}, p_y = p_{y0}, v_x = v_{x0}, v_y = v_{y0}\})$, the system begins operating in the falling state, with some initial positions and initial velocities. This is indicated by the single incoming arrow to the falling state.

- $Inv(q_1) = \{(p_x, p_y, v_x, v_y) \in \mathbb{R}^4 \mid \sin p_x - p_y \leq 0\}$ this constraint is termed an *invariant*. It must always evaluate to true while the system is in the specified discrete state. If remaining there would cause the invariant to be violated, a transition is forced.

- $E = \{(q_1, q_1)\}$, there is only one possible transition.

- $G(q_1, q_1) = \{(p_x, p_y, v_x, v_y) \in \mathbb{R}^4 \mid \sin p_x - p_y = 0\}$, the guard is satisfied when the bouncing ball hits the sine curve. Switching can potentially occur when the guard is satisfied.

- $R((q1, q1), (p_x, p_y, v_x, v_y)) =$

$$\{v_x := \frac{(1 - 0.8\cos^2 p_x)v_x + 1.8v_y \cos p_x}{1 + \cos^2 p_x},$$

$$v_y := \frac{1.8v_x \cos p_x + (-0.8 + \cos^2 p_x)v_y}{1 + \cos p_x}\}$$

The velocities in the x and y dimension get updated when the ball hits the sine curve.

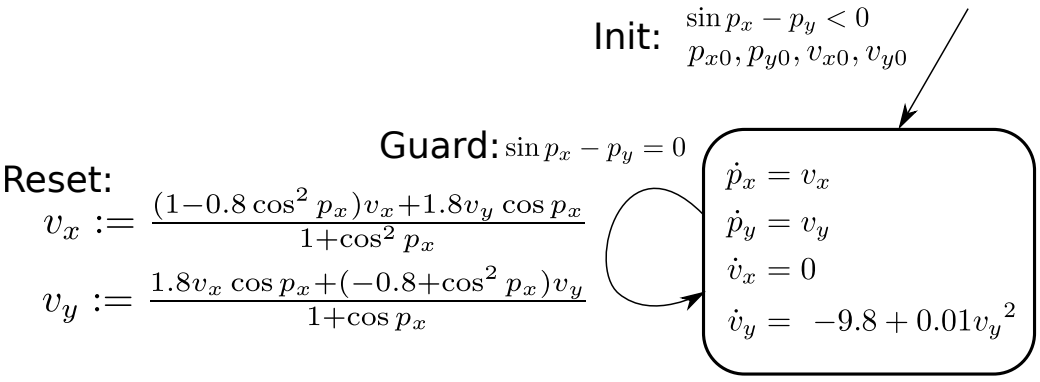


Figure 2.6: Hybrid automaton for a bouncing ball

2.3.2 Solutions of Hybrid Systems

Just as the case with continuous dynamical systems, there is a practical concern whether a hybrid automaton model is well-posed. The question is, for each state of the hybrid system in the set *Init*, does there exist a solution and is it unique? Since the model of the system now contains both continuous flows and discrete jumps, the notion of a solution

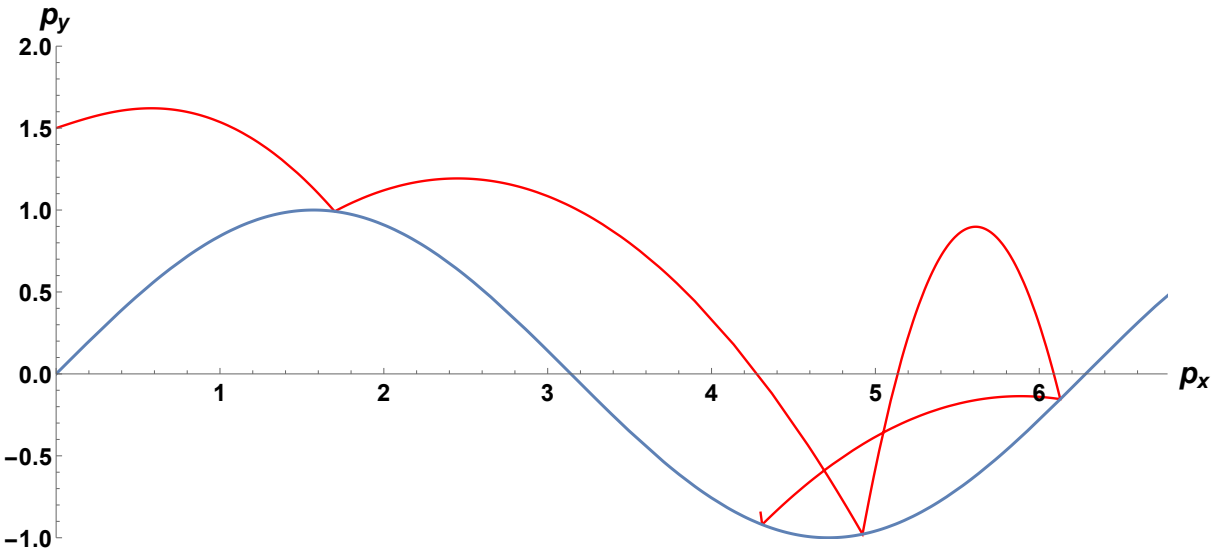


Figure 2.7: Simulation trace of the bouncing ball

cannot be defined as a single continuous trajectory. A *hybrid time trajectory* gives a precise representation of how a hybrid system evolves over time that takes into account both continuous and discrete behaviours. We start by defining an abstraction of time.

Definition 2.6 (Simplified hybrid time set)**.** A hybrid time set $\tau = \{I_i\}_{i=0}^{N}$ is a finite or infinite sequence of intervals of the real line, such that

$$\text{for all } 0 \leq i < N, I_i = [\tau_i, \tau_i'] \text{ with } \tau_i \leq \tau_i' = \tau_{i+1}$$

Each interval represents the time over which the hybrid system flows continuously in its separate modes. In Example 2.3, this is the time taken for the ball to fall between bounces. The right hand side τ_i' are the times when discrete transitions take place. These are the time instances when the ball hits the curve. We now define the specific meaning of a trajectory or solution of a hybrid system, defined in terms of a hybrid time set.

Definition 2.7 (Hybrid system trajectory, Johansson et al. [119])**.** A hybrid trajectory over the set of hybrid states $Q \times X$ is a triple (τ, q, x), where τ is a hybrid time set, $q : I_i \to Q$ and $x : I_i \to X$, satisfying

1. Initial condition: $(q(\tau_0), x(\tau_0)) \in Init$
2. Discrete evolution: for all i, e :
   - $(q(\tau_i, \tau_i'), q(\tau_{i+1}, \tau_{i+1}')) \in E$.
   - $x(\tau_i') \in G(e)$.
   - $x(\tau_{i+1}) \in R(e, x(\tau_i'))$.
   - $x(\tau_{i+1}) \in Inv(q(\tau_{i+1}, \tau_{i+1}'))$.

3. Continuous evolution: for all i, such that $\tau_i < \tau_i'$

   - $q_i : I_i \to Q$ is constant over $t \in I_i$.
   - $x_i : I_i \to X$ is the solution to the differential equation

     $$\frac{dx_i}{dt} = f(q_i(t), x_i(t))$$

     over I_i starting at $x_i(\tau_i)$.
   - for all $t \in [\tau_i, \tau_i'), x_i(t) \in Inv(q_i(t))$.

the function q assigns to each hybrid time set interval I_i a discrete state q_i. The function x assigns to each hybrid time set interval I_i values from the continuous state space X.

The first condition enforces that the hybrid trajectory start time $\tau_0 = 0$ (the very first point of the hybrid time set) must map to a corresponding state in the set $Init$ of the hybrid automaton. The second condition enforces that any edge e leaving a discrete mode q_i must be *enabled* by the corresponding guard G and that the new continuous state vector is assigned a value by the reset map R. Finally, the third set of conditions enforce that the system flows according to the solution of the system of differential equations f assigned to state q_i and must remain in the invariant set Inv. A transition from this state will be *forced* at a time τ_i' if the trajectory leaves Inv.

The formal definition of a hybrid system trajectory describes the behaviour of a hybrid automaton with respect to time. We say that a hybrid trajectory (τ, q, x) is infinite if τ is an infinite sequence. The hybrid trajectory is maximal if no other trajectories of the hybrid automaton are longer. Finally, we can now state the conditions for the uniqueness of a hybrid system trajectory.

Definition 2.8 (Uniqueness of a hybrid system trajectory, Johansson et al. [119])**.** A hybrid automaton H is called non-blocking if for all initial states $(q_0, x_0) \in Init$ there exists an infinite hybrid trajectory starting at (q_0, x_0). It is called deterministic if for all initial states (q_0, x_0) there is at most one maximal trajectory. A hybrid automaton H has a unique hybrid trajectory if it is both non-blocking and deterministic.

Just as with continuous systems, all the examples considered in this dissertation are *well behaved* hybrid dynamical systems that have a unique trajectory. This assumption again is one of convenience, so that the focus of this dissertation can be on verification, rather than on modelling.

2.4 MetiTarski : An Automated Theorem Prover

There are few methods or tools that can automatically prove statements containing the real valued transcendental functions that are typically found in dynamical system verification problems. Take for instance the solutions of linear systems that, as discussed

in Section 2.2.2, can potentially contain exponential, sine and cosine terms. One verification strategy is to perform a reachability analysis. The goal here is to show that all solutions $\phi(t)$, starting in some initial region I of the state space, will never reach an unsafe region U. It is quite natural to define this condition as a first order formula over real inequalities of the type $\forall t : \phi(t) \notin U$. To perform a reachability analysis of a dynamical system requires being able to reason not only about transcendental functions but also about statements involving inequalities between such functions.

The automated theorem prover MetiTarski [4] perfectly meets the requirements necessary for performing a reachability analysis. It combines a resolution theorem prover and a decision procedure to prove first-order sentences of real number inequalities that contain transcendental and other special functions.

In the resolution framework [181], logical sentences are represented as clauses that are a disjunction of literals (an atomic formula or its negation). Resolution performs a proof by refutation. The conjecture is negated, it is combined with any available axioms, and the resolution process attempts to reduce the set of clauses to the empty clause. The empty clause represents false, indicating a contradiction has been found and proving that the original conjecture is true.

Within MetiTarski, the standard resolution framework is modified as follows: occurrences of special functions are isolated by the resolution procedure and are replaced by a series of upper and lower polynomial bounds that are provided via axiom files. This substitution reduces the problem to the theory of real closed fields (RCF), which is decidable [205]. External RCF decision procedures are called to delete algebraic clauses that are inconsistent with other derived facts. The ideal case is that each call to the RCF decision procedure moves the proof towards the empty clause.

The RCF decision procedures are the bottleneck in this process. Although the theory of first order formulas over polynomials containing real numbers (the real closed fields) admits quantifier elimination, it has a doubly exponential run time in the number of variables [62]. This is the main cause of the dimensionality gap problem discussed in Section 1.4. Over the course of the development of MetiTarski, three decision procedures have been integrated with MetiTarski for making RCF decisions:

QEPCAD

QEPCAD [37] is an implementation of cylindrical algebraic decomposition (CAD), which is a real algebraic geometric method that can perform quantifier elimination over the reals. It is very efficient on single variable problems. Unfortunately, when using it, MetiTarski times out for most problems of more than 3 variables. Consequently, it has not been used in any of the experiments discussed in this dissertation.

Mathematica

The computer algebra system Mathematica contains implementations of several highly optimised CAD algorithms, which allow MetiTarski to handle problems up

to 4 or 5 variables. However, to be able to take advantage of the power of the CAD algorithms requires a deep knowledge of quantifier elimination, real algebra and the numerous input parameters (many of which are undocumented). I have therefore avoided using Mathematica as the back-end to MetiTarski. One interesting feature of note, is the ability of the Mathematica CAD algorithms to handle coefficients that are real exact transcendental numbers such as e and π.

Z3

The SMT solver Z3 [65] has an internal module called *nlsat* [121] that implements an efficient method for deciding purely existential RCF sentences. Combined with strategies tailored to the types of RCF problems generated by MetiTarski, it has been used to successfully prove problems of up to 11 variables. Since Z3 does not require any manual tweaks to its algorithms and because it vastly outperforms both QEPCAD and Mathematica, it is the RCF decision procedure that has been used for the experiments of this dissertation.

2.4.1 High Dimensional Problems

The default parameters to MetiTarski have been chosen to give the best performance on the problem set that is included with it. However, the default parameters are not optimised for high dimensional problems (8-10 continuous-time variables). The success of the RCF decision procedures gradually decrease as the number of continuous variables increase. With this in mind, there is an important command line argument to MetiTarski that can have a drastic effect on the proof times of high variable problems.

RCFtime

If a difficult RCF problem is encountered by the decision procedures it can potentially block any further progress of the proof. This is especially bad in the case when there are easier RCF problems that are waiting to be analysed. In the worst case scenario, the global timeout will be hit and the proof will fail. It is possible to limit the time spent on individual RCF problems using the "--RCFtime" argument. By limiting the amount of time on individual RCF problems, MetiTarski can minimise the time wasted on difficult RCF problems and focus on easier ones, potentially moving the proof forwards. Care must be taken as setting the RCFtime too low can cause all RCF problems to timeout causing the proof to fail.

The experiments in this dissertation try various RCF timeouts in an attempt to find the best value of the RCFtime input parameter.

2.4.2 Experimental Results

MetiTarski can be downloaded from:

`https://code.google.com/p/metitarski/`

All the relevant files from the experiments described in this thesis are located in the subdirectory `tptp/Denman-Thesis/` of the full MetiTarski distribution. Information about the qualitative abstracter tool QUANTUM is provided in Appendix A. The experiments in this dissertation were all performed on a dual quad-core 2.6 GHz Intel Core i7 with 8GB of RAM.

CHAPTER 3

Linear Continuous System Verification

The broad goal of this dissertation is the development of a methodology for the automated verification of nonlinear hybrid dynamical systems modelled by hybrid automata. Hybrid automata are essentially finite state machines, where each discrete state defines the behaviour of a separate continuous vector field. As a first step towards a complete hybrid system verification framework, this chapter investigates the performance of MetiTarski on the verification of purely linear continuous dynamical system problems.

The MetiTarski theorem prover was chosen for these initial experiments primarily because it can reason automatically about transcendental and special functions. This is important for analysing the solutions of linear systems because they will generally contain sine, cosine and exponential functions (see Section 2.2.1 for more details). Under certain assumptions and conditions, a nonlinear system can be transformed into an equivalent linear system by replacing the nonlinear terms with new variables. This recasting process increases the dimension of the verification problem, adding to its complexity. The recasting process and its effect on verification times will also be described in this chapter.

The purpose of the initial experiments, which were restricted to the verification of continuous systems, was to identify the problem size (in terms of the number of continuous variables required) prior to moving on to the the more complex hybrid system verification experiments in Chapter 5. The initial experiments described in this chapter have also been used to evaluate the integrated decision procedure *nlsat* [121] that is used by MetiTarski for making RCF calls. It has proved to be particularly useful for the verification of problems with many variables generated by the recasting process. Finally, the continuous system experiments explored the limitations of MetiTarski and motivated the development of techniques to overcome them.

There are three techniques that are analysed and developed in this chapter. Each addresses a particular shortcoming of MetiTarski that was encountered during the experimental analyses. The end goal of the researched techniques was to reduce the time required to find proofs for continuous dynamical systems of eight to ten continuous variables. The first technique progressively limits the time limit of the RCF decision procedure. The second recasts away transcendental functions, when possible, to reduce the number of algebraic variables processed by the resolution loop. Finally, the last technique iteratively splits the range of continuous variables to convert difficult problems into several easier ones. These three techniques enable MetiTarski to prove dynamical system problems with as many as 11 continuous variables. Proofs of such problems were previously beyond its reach.

This chapter begins with the definition of the *safety* of a dynamical system. This is followed by an overview of the methods available for the verification of continuous dynamical systems. The automated theorem proving methodology is introduced through a motivating example and then applied to a case study taken from the domain of flight collision avoidance protocols.

3.1 Preliminaries

An engineer's most important concern is safety. A system under design is considered safe if it cannot, under any circumstances, do something wrong. This is particularly important for safety critical systems, where unsafe behaviour can potentially lead to casualties and loss of life. Recall from Section 2.2.1 that a continuous dynamical system is defined by a system of differential equations and its time dependent transient behaviour is represented by a trajectory (solution). The verification of a dynamical system safety is then best defined as a *reachability* problem.

Definition 3.1 (Reachability)**.** A dynamical system is safe if no trajectory (solution) starting in a *safe* initial state can reach an *unsafe* state.

For example, an aeroplane's rudder and ailerons should not be able to move into a position that causes the engines to stall. Consider a lift bridge that raises for a passing ship, it must never close when a ship is still below. In the next sections I will describe several methods that can be used to perform a reachability analysis of continuous dynamical systems.

3.1.1 Simulation Methods

The set of trajectories beginning in a safe region can be quite large due to the number of variables, parameters and initial conditions of the dynamical system under analysis. Therefore, it is not usually possible to numerically simulate all possible trajectories. En-

gineers will typically try to isolate worst-case corner values and simulate the system with them. If any unsafe states are reached by the simulation then this clearly demonstrates the violation of the reachability safety definition. However, if no violation is found, then it cannot be concluded that the dynamical system is safe with respect to Definition 3.1.

Monte Carlo methods [143] are a group of statistical techniques for analysing the sensitivity of dynamical systems to a variation of parameters. By randomly picking values for the initial states and variables of the system, a group of representative trajectories can be found. This is continued until it is deemed that a sufficient finite number of trajectories have been obtained. These statistical based techniques can also be used for performing the integration[1] used by the simulation algorithms, which is especially important when simulating over complex domains [97]. Although Monte Carlo methods can find safety violation cases that may have been missed by the engineer, they suffer from the same drawback as standard simulation. Conclusions about the safety of a system cannot be made if there is no trajectory that violates the reachability conditions.

Figure 3.1 demonstrates how simulation can fail. An unsafe region, represented by a red circle, has not been reached by any of the simulated trajectories (the solid lines) starting from an initial safe region, defined here by a green ellipse. The system appears to be safe; however, there exists a trajectory that reaches the unsafe region (the dotted line) which has been missed.

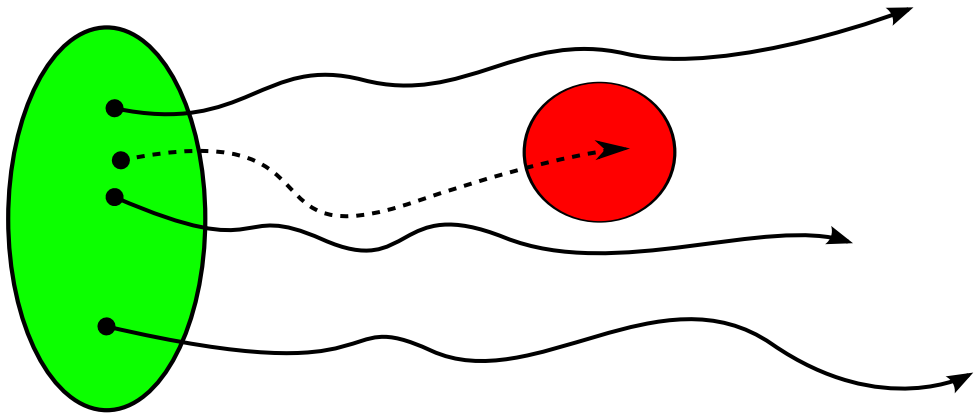


Figure 3.1: Reachability with simulation

3.1.2 Validated Solutions to Dynamical Systems

As a step towards the complete verification of continuous systems, techniques using interval arithmetic [145] have been developed for calculating enclosures to the trajectories of systems of ordinary differential equations. These methods produce so-called *verified* bounds of the solutions to the system that are *guaranteed* to exist within a certain interval.

[1] By randomly picking the integration step-size, for instance.

A representative implementation of this method is VNODE-LP [158], which uses Taylor series expansions at each individual integration step to compute tight over-approximations of the trajectories. It can efficiently compute the bounds on solutions to systems that operate over short time intervals and those that begin within a small set of initial conditions. Another example is the COSY package [137], which can handle larger initial sets. VSPODE [133] specifically targets systems with varying parameters, which can cause computability issues for VNODE-LP and COSY. Much research in this domain is focused on finding ways to limit the *wrapping effect*, where the intervals grow too large to convey any information about the underlying solution. Other notable work on rigorously verified ODE solvers is the Computed Assisted Proofs in Dynamics (CAPD) library [218].

An important question is whether the interval analysis methods for enclosing solutions to dynamical systems are correct. The proofs for these algorithms are generally done at a high level and by hand [179]. This issue of correctness has been tackled by the author of the VNODE software by using the Literate Programming method [126] (the LP in VNODE-LP). Code and documentation are interleaved so that the reasoning of the programmer can be better followed. The motivation behind LP is to take advantage of natural language to write higher quality algorithms and to make it easier to manually check the correctness of programs. This justification of correctness is incredibly weak. A move in a better direction is the recent work on formalising the interval solution methods in the interactive theorem prover Isabelle [115]. This stronger formal correctness will be key to motivating the use of the validated ODE solvers in safety verification frameworks.

In summary, interval differential equation solvers can produce tight enclosures of the solutions to systems of differential equations and there is some support to show that these algorithms are correct. However, the parameter and initial value ranges must be kept small to avoid inaccuracies induced by the interval representation of the variables. Furthermore, the methods only produce usable results over a short and bounded time domain. Consequently, it is difficult to verify the long-term transient behaviour of dynamical systems with these methods.

Figure 3.2 shows how an interval reachability method can enclose the solutions to a system of differential equations. Note how if the algorithm does not run long enough, trajectories leading to an unsafe state can be missed.

3.1.3 Algorithmic Reachability

Interval-based integration methods have led to the development of more general techniques for computing the reachable states of dynamical systems. Interval analysis techniques use hyper-rectangles. Algorithmic reachability methods, on the other hand, use

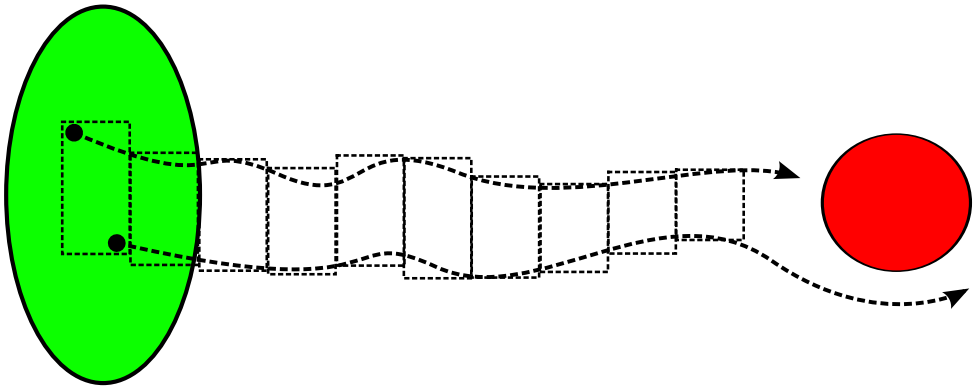


Figure 3.2: Reachability with verified interval methods

more efficient representations that are easier to work with and can provide tighter bounds on the reachable space. Examples of such representations include ellipsoids [215], convex polytopes [45], zonotopes [88] and support functions [131].

The most advanced work to date on computing reachable states of linear continuous dynamical systems has been implemented in the SpaceEx tool [83]. Systems of 100 variables and more have been successfully analysed [130]. However, the algorithms used for such many-variable problems are not numerically sound (floating point computations are used) [131] and therefore SpaceEx cannot be used to certify safety in this case. As with interval based techniques, these algorithmic methods are usually constrained to calculating the reachable set only for a finite time.

Figure 3.3 gives an abstract view of how algorithmic reachability methods work. At each iteration, the set of reachable states up to some time bound are approximated (in this 2-dimensional case polygons are used). However, it can happen that the over-approximations of the trajectories intersect with unsafe states (the top circle) when the actual trajectories of the system do not, leading to a false positive. As well, if the algorithm does not run long enough, certain unsafe states can be missed (the right circle).

3.1.4 Theorem Proving and Deductive Reasoning

Simulation and algorithmic reachability-based methods are constrained by the fact that they rely on iterative algorithms to calculate the set of reachable states. Not only do they use numerical methods (which can be unsound), but the results are limited to finite time horizons which are inadequate for showing a dynamical system is safe under all operating conditions. Instead of trying to compute approximations of trajectories, the reachability problem can be expressed instead as a set of inequalities that must be proved true using first principles such as calculus and deductive reasoning. The safety of a dynamical system is then just a logical conjecture that when proved is guaranteed to hold for all time.

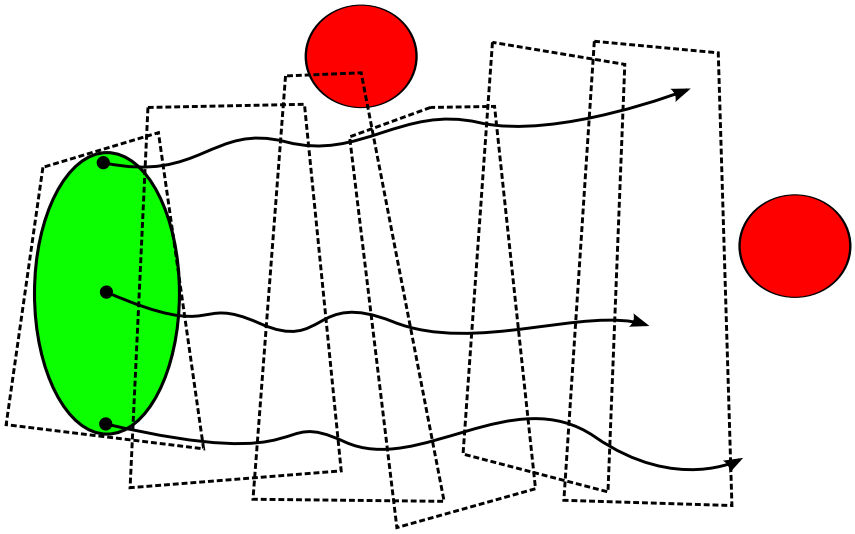


Figure 3.3: Algorithmic reachability methods

Although the conjectures defining a reachability property can be proved by hand, this is time-consuming and potentially error-prone. Instead, a theorem prover can be used to formally verify the conditions over analytical functions and real variables without the need for floating point computations. The main benefit here is that each step of the proof is mechanically checked ensuring that all reasoning is sound. A notable application of this type of verification via formal proof is the Flyspeck Project [95] that aims to formalise Hales' proof of the Kepler Conjecture on sphere packing [94]. The proof using hand calculations and linear programming was of such great complexity that even after several years a panel of twelve referees was not able to completely certify it [96]. This was the motivation for formalising the proof in a theorem prover to remove any uncertainty of the proof's correctness.

The Prototype Verification System (PVS) [160] is one such theorem prover that contains a specification language and a deductive reasoning engine that can be used to verify continuous dynamical systems. For example, Hardy [99] developed and implemented a decision procedure that uses PVS to reason about continuous functions that have a finite number of inflection points. This decision procedure, the Nichols plot Requirements Verifier (NRV), performs an automated formal analysis to categorise the stability of linear continuous dynamical systems. The tool also uses the symbolic methods of the computer algebra system Maple and the quantifier elimination procedure QEPCAD [37]. NRV was successfully used to verify the stability of several classic linear control system examples including an inverted pendulum and a disk drive reader [29].

Several formalised theories for real numbers are available in the NASA PVS library [1]. These include specifications for intervals and vectors that are essential for reasoning about continuous systems. PVS can also be interfaced with trusted external oracles when

its internal methods are not capable of solving a difficult problem directly. This situation is addressed in Chapter 4, where MetiTarski is integrated as an external decision method to PVS.

Although PVS is quite powerful and the theories for the analysis of conjectures over the reals are well developed, proving theorems about dynamical systems still requires a great deal of expertise to guide the proof assistant. More specialised theorem provers have been built to directly target dynamical systems. KeYmaera [171] is one such interactive theorem prover built specifically for the verification of hybrid systems. It can also be used for verifying purely continuous systems. Its automatic methods rely on identifying inductive invariants (similar to Lyapunov functions or barrier certificates, see Section 2.2.5) that guarantee that unsafe states of the system cannot be reached. Although many of the deductive steps implemented in KeYmaera are automatic, it still requires a significant amount of manual guidance.

MetiTarski, on the other hand, is a theorem prover that performs each of its deductive steps automatically. The dynamical system reachability problem can be defined as a first-order formula over unbounded parameters, including large or infinite time ranges. After the conjecture is defined, it is sent to MetiTarski and no further interaction is required. If the verification problem is too difficult, then it can be split into several MetiTarski problems each covering a different region of the state space.

Figure 3.4 demonstrates how MetiTarski is able to cover the entire reachable set. All possible trajectories out of the initial safe set, represented as a green ellipse cannot reach the unsafe red set. If the solution of the system is represented in a symbolic form with the initial conditions and parameters represented by variables, the unsafe states can be represented as bounds on the solutions. In this case, MetiTarski will attempt to find a mathematical proof guaranteeing that the unsafe states indicated here by unbounded red zones are never reachable.

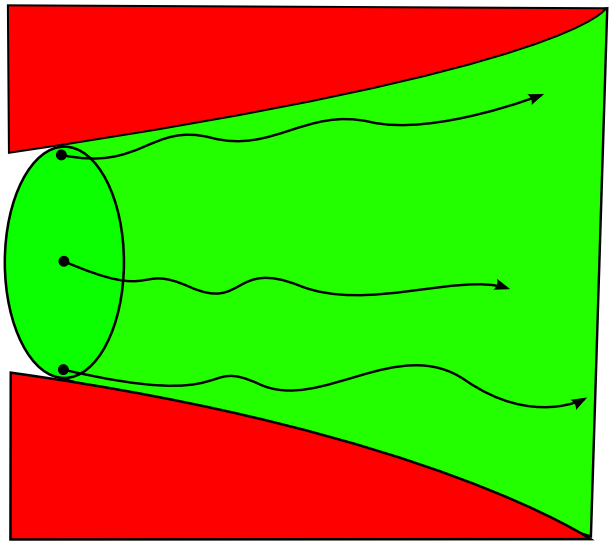


Figure 3.4: Reachability with theorem proving

Although iterative methods can work on much higher dimensional systems, lack of soundness precludes a safety result. MetiTarski has previously had success at verifying reachability properties generated from the techniques described above. For instance, the Nichols Plot analysis with PVS described by Hardy was reimplemented and automated using MetiTarski [72]. The following experiments in this chapter further demonstrate how MetiTarski can be used to automatically verify the safety of continuous dynamical systems.

3.1.5 MetiTarski Syntax

MetiTarski uses a variant of the Thousands of Problems for Theorem Provers (TPTP) format [203] to define conjectures to be proved (though it will also accept the SMT-LIBv2 format [19]). An example is shown in Figure 3.5. The label *"fof"* indicates that the formula uses first order logic. The proof is given the name "Tunnel". The keyword *"conjecture"* indicates that the following formula is to be proved. The conjecture can be understood as follows: For all (!) X between 0 and 2.39×10^{-9} this implies (=>) that the formula $-0.0059 - 0.000016e^{-2.55\times10^8 X} + 0.031e^{-5.49\times10^7 X}$ is always less than 0.03.

```
fof(
  Tunnel,conjecture, ! [X] :
  (
    (0 <= X & X <= 2.39*10^(-9)) =>
     -0.0059 - 0.000016*exp(-2.55*10^8*X) + 0.031*exp(-5.49*10^7*X)
     < 0.03
  )
).

include('Axioms/general.ax').
include('Axioms/trans.ax').
include('Axioms/exp-general.ax').
include('Axioms/exp-lower.ax').
include('Axioms/exp-upper.ax').
```

Figure 3.5: MetiTarski syntax

3.1.5.1 Axioms

Axiom-include-statements must be used to indicate to MetiTarski which upper and lower bounds to use when replacing the transcendental functions during the resolution process.

The include statements can be automatically provided to MetiTarski using the *"--autoInclude"* command line argument. However, this process is not perfect and more axioms than necessary might be chosen for inclusion, which can have a negative effect on proof times. This is because each additional axiom set increases the proof search space.

For example, there are two sets of axiom declarations for the exponential function. One for *regular bounds* and one for *extended bounds*. There are cases where including the extended bounds will make the inequality under analysis unsolvable within a reasonable time. Removing the extra axioms will enable MetiTarski to complete the proof. The converse is also true, if for instance the TPTP description contains trigonometric functions and those axioms are not included, then the conjecture will not be provable and may fail quickly. For the examples contained in the rest of this dissertation it will be assumed that the correct axiom files have been chosen and included.

3.2 Experimental Methodology

The experiments in this chapter are based around proving the reachability properties of continuous dynamical systems with the automated theorem prover MetiTarski. The methodology is shown in Figure 3.6.

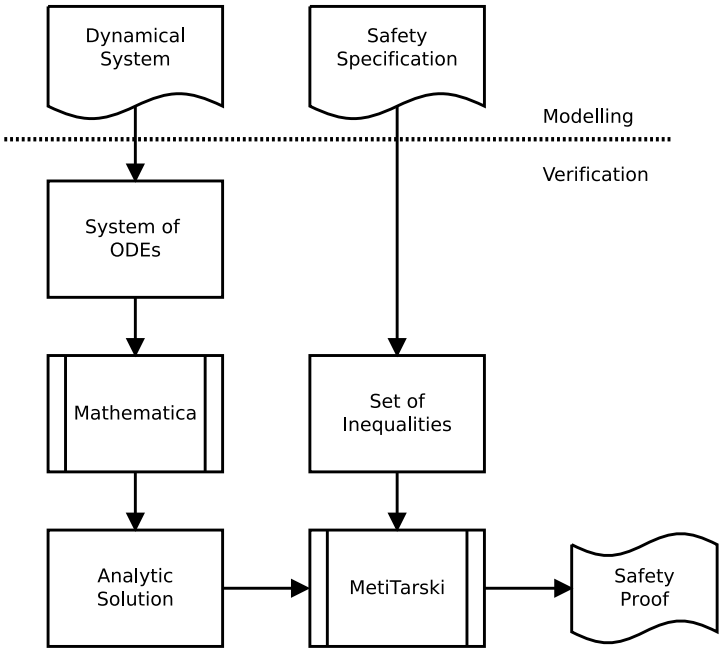


Figure 3.6: Continuous system verification methodology

We start with the analysis of linear systems that admit closed form solutions. The system of differential equations defining the behaviour of the dynamical system is encoded into a form that can be input to the computer algebra system Mathematica.[2] Mathematica is then used to solve the system and return an analytical solution. The solution will contain a variable for each initial condition of the system. The initial states and unsafe states are each converted into a set of inequalities in terms of the symbolic solution, where each variable ranges over a bounded or unbounded interval. Finally, the reachability

[2]Mathematica is used here to solve the system of differential equations and is not used as a back-end to MetiTarski

problem is encoded into a series of problems in the format given in Section 3.1.5 and sent to MetiTarski for analysis. If MetiTarski is able to prove each conjecture then the system is labeled safe with respect to the safety specification.

The second type of dynamical system under analysis are nonlinear systems that can be transformed into a linear system by recasting the nonlinear terms to new variables. The resulting linear system can then be analysed as before, with Mathematica producing an analytical solution and MetiTarski performing the reachability analysis.

The goal of these experiments is to see how well MetiTarski can handle the verification of continuous dynamical systems. They will be deemed successful if MetiTarski can, within a reasonable amount of time[3], prove the required conjectures. Any failure to meet this criteria, will motivate the subsequent development of proof techniques reported in this chapter. A technique will be deemed acceptable if a previously unprovable conjecture is proved within the reasonable time limit defined above.

3.3 Example: Linear System Reachability

This section presents an example originally taken from work on abstracting continuous dynamical systems to a discrete state model [196]. This is in contrast to the continuous system verification methodology developed in this chapter that works on the system's equations and solutions directly.

3.3.1 System Definition

A three dimensional system, modelling the motion of an oscillating particle, may be defined by the following system of differential equations

$$\dot{x}_1 = -0.1x_1 - x_2 \tag{3.1a}$$

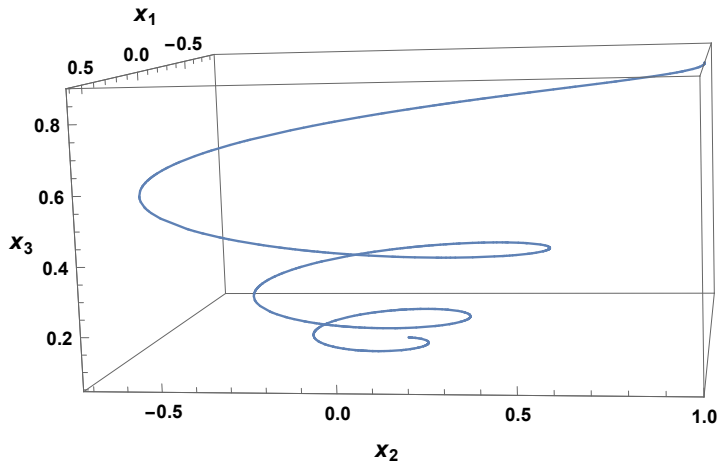
$$\dot{x}_2 = x_1 - 0.1x_2 \tag{3.1b}$$

$$\dot{x}_3 = -0.15x_3 \tag{3.1c}$$

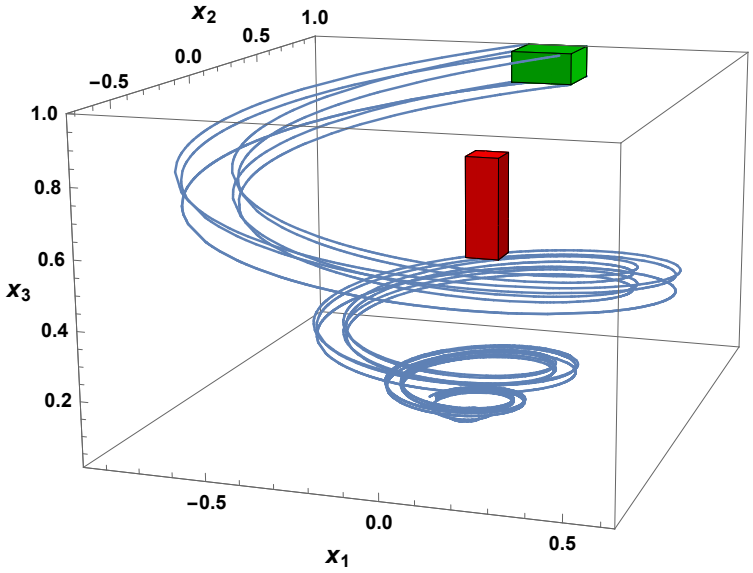
The x_1, x_2, x_3 variables represent the coordinates of the particle in 3 dimensions. The variables are each implicitly a function of time t. A simulation trace of the system is shown in Figure 3.7a, with initial conditions $x_1 = -0.1, x_2 = 1, x_3 = 0.9$.

In this example, the system must satisfy the following specification: all trajectories starting in the safe (green) box defined by $x_1 = [-0.1, 0.1], x_2 = [0.8, 1], x_3 = [0.9, 1]$ must avoid an unsafe (red) box defined by $x_1 = [0, 0.1], x_2 = [0, 0.1], x_3 = [0.5, 0.8]$. Several simulation traces starting from the edges of the safe box are shown in Figure 3.7b.

[3]Reasonable here is within 600 seconds or 10 minutes



(a) Simulation trace



(b) System specification

Figure 3.7: Trajectories of the system

3.3.2 Analytical Solution

The analytical solution to this system, obtained with Mathematica, is

$$x_1(t) = c_1 e^{-0.1t} \cos t - c_2 e^{-0.1t} \sin t \tag{3.2a}$$

$$x_2(t) = c_1 e^{-0.1t} \sin t + c_2 e^{-0.1t} \cos t \tag{3.2b}$$

$$x_3(t) = c_3 e^{-0.15t} \tag{3.2c}$$

with c_1, c_2, c_3 representing the initial locations $x_1(0), x_2(0), x_3(0)$.

3.3.3 MetiTarski Input

The bounds on the initial states are combined with the symbolic solution to represent the safety reachability property as a first-order formula in the MetiTarski syntax. The specification can be read directly from Figure 3.8. For all time (denoted by variable T) the constraints on the initial conditions (variables C1, C2, C3) imply that the trajectories of the system never[4] enter the unsafe box. Unfortunately, MetiTarski was initially unable to prove this conjecture with a global timeout of 600 seconds. This motivated a deeper analysis of the problem.

```
fof(p1,conjecture,(! [T,C1,C2,C3] :
(
  (T > 0 &
   C1 > -0.1 & C1 < 0.1 &
   C2 > 0.8 & C2 < 1 &
   C3 > 0.9 & C3 < 1)
   =>
   ~(
        ((C1*cos(T) - C2*sin(T))*exp(-0.1*T)) < 0.1 &
        ((C1*cos(T) - C2*sin(T))*exp(-0.1*T)) > 0 &

        ((C2*cos(T) + C1*sin(T))*exp(-0.1*T)) < 0.1 &
        ((C2*cos(T) + C1*sin(T))*exp(-0.1*T)) > 0 &

        (C3*exp(-0.15*T) > 0.5) &
        (C3*exp(-0.15*T) < 0.8))
))).
```

Figure 3.8: MetiTarski syntax

3.3.4 Limiting the RCF time

Analysing the initial failure of MetiTarski in more detail, it was observed that the amount of time taken up by the RCF decision procedure for this example was 603.19 seconds out of 604.102 seconds (over 99% of the total proof time). Analysis of the proof trace showed that the resolution procedure was blocked on a difficult RCF subproblem, causing the global time limit to be exceeded.

The first approach to solve this issue was to limit the amount of time spent on individual RCF subproblems. The following table shows the effect of different timeout values on the MetiTarski input of Figure 3.8. In Table 3.1 the label "timeout" refers to hitting the

[4]This is indicated with the logical *not* symbol ~.

default global limit of 600 seconds, while "gave up" refers to the resolution loop running out of clauses to process. The breakdown column shows the individual timings of the resolution theorem prover Metis and the RCF decision procedure nlsat.

RCF Timeout (ms)	Proof Time (s)	Breakdown
Unlimited	timeout	8.913 (Metis) + 603.189 (RCF)
10000	413.342	8.024 (Metis) + 405.318 (RCF)
7500	326.87	7.943 (Metis) + 318.927 (RCF)
5000	269.369	7.908 (Metis) + 261.461 (RCF)
1000	77.827	7.769 (Metis) + 70.058 (RCF)
750	74.84	7.706 (Metis) + 57.137 (RCF)
500	51.455	7.881 (Metis) + 43.575 (RCF)
250	35.575	7.900 (Metis) + 27.675 (RCF)
100	23.745	8.133 (Metis) + 15.612 (RCF)
50	20.670	8.467 (Metis) + 12.203 (RCF)
10	gave up	6.180 (Metis) + 7.149 (RCF)

Table 3.1: RCF timeout experiment results

The experimental results show that limiting the time spent on individual RCF problems can have a drastic effect on proof times. Specifically, in this example, MetiTarski went from working for over 10 minutes and obtaining no proof to proving the conjecture in just 20 seconds. As the time limit only applies to the RCF decision procedure, the time spent by Metis is held relatively constant. To investigate further, MetiTarski was run on its entire problem set of 839 problems with a varying RCF timeout. The results of this experiment are shown in Table 3.2.

RCF Timeout (ms)	Total Proof Time (s)	# Proved	Gave Up	Timed Out
None	815.7	664 (79%)	44	131
10000	1259.0	675 (80%)	49	115
1000	1024.8	664 (79%)	59	116
100	621.2	615 (73%)	116	108
10	486.2	522 (62%)	154	143

Table 3.2: RCF timeout experiments on the full problem set

Applying an RCF timeout of 10,000 milliseconds or 1000 milliseconds did not greatly alter the total number of problems proved. However, as the time limit was lowered further, there was a noticeable decrease of problems proved and an increase in those that were given up on. This behaviour can be explained by the fact that some inferences

in the proof require a minimal amount of RCF time. A lower individual RCF timeout causes a higher number of subproblems to be discarded, leading to more clauses being dropped during the resolution process.

The lack of universal improvement on MetiTarski's standard problem set can be explained by several facts. The problem set is composed of problems taken from a variety of sources (e.g. mathematical inequalities and identities) that can already be proved quite quickly. Other problems come from simple hybrid and continuous system verification problems where there are no more than five real variables. This is primarily due to the fact that the performance of the decision procedure degrades quickly for high variable problems. The majority of these problems were added before the availability of the RCF time limit as an option. Therefore, it is unsurprising that the proof times are not significantly affected (positively or negatively) by restricting the time of RCF subproblems.

The resulting number of total problems proved is misleading as several difficult problems that timeout within 600 seconds are successfully proved when a RCF time limit is in place. These successful proofs are hidden by some of those that are timed out on. Table 3.3 highlights several interesting examples that showcase this behaviour. The "Problem Name" column identifies the corresponding name of the first-order formula problem in the MetiTarski problem set.

	RCF Timeout (ms)			
Problem Name	None	10000	1000	100
atan-problem-2-weakestT	timed out	timed out	49.5	23.9
Chua-2-VC1-L	timed out	timed out	51.3	4.4
cos-problem-9-weak	73.1	58.9	13.6	gave up
CONVOI2-sincos	80.7	31.4	6.0	2.4
log-nest-exp-twovars-weak1	timed out	23.9	5.8	4.4
polypaver-bench-sqrt	timed out	timed out	25.6	timed out
sin-problem-7-weak	timed out	31.0	7.5	timed out
sqrt-1mcosq-7-weak	timed out	timed out	52.2	35.7
sqrt-cos-problem-2-2vars	111.9	38.1	11.2	5.5
tan-2-2var-weak2	timed out	21.1	3.1	gave up

Table 3.3: Isolated MetiTarski problem proof times

The results show that for particularly difficult conjectures that cannot be proved within a timeout of 600 seconds, imposing an RCF time limit in these cases can lead to a proof. However, as demonstrated by the experiments on the full problem set, arbitrarily setting the time limit can cause a performance hit. Several of the isolated problems that do show a significant improvement, for instance Chua-2-VC1-L and CONVOI2-sincos, come from dynamical system verification problems.

The RCF timeout of 1000 milliseconds is particularly important, as shown in Table 3.3, to prove problems that were previously impossible to verify in a reasonable amount of time. This specifically meets the criteria for success outlined for the experiments of this chapter. Although the RCF timeout technique is particularly good for high dimensional problems, beyond 11 continuous variables the doubly exponential complexity of the underlying decision procedures is hit. Therefore, other techniques will be presented that address this issue in particular.

In conclusion, an RCF timeout should not be used until it is required. One such use case is the safety verification of continuous dynamical systems that cannot be proved using the default MetiTarski parameters. The experimental results also indicate that an RCF timeout of 1000 milliseconds should give an appropriate balance between proof times and probability of a proof being found.

3.3.5 Recasting Technique

There might be cases where adding an RCF time limit has no effect. An alternative strategy for improving proof times is to reduce the number of transcendental functions in the conjecture to be proved. This can be done through a recasting process. For example, consider a conjecture of the form

$$\forall X, Y : \sin X + \cos X > Y + X$$

It can be recast by replacing $\sin X$ and $\cos X$ by two variables S and C and including a constraint from the identity $\sin^2 X + \cos^2 X = 1$ resulting in

$$\forall X, Y, S, C : S^2 + C^2 = 1 \wedge S + C > Y + X$$

This recasting process can sometimes make the proof easier for MetiTarski to complete, especially in cases of nested functions and problems that contain many variables. Take for instance the present moving particle example. Applying this recasting process results in the MetiTarski input in Figure 3.9 and is proved in 0.270 seconds. Recall that even with an RCF timeout of 50 milliseconds it took MetiTarski 20.67 seconds to prove this conjecture. The recasting process decreased the proof time by a factor of 75.

The recasting technique described in this section can be broadly applied to many systems. The main difficulty is choosing the correct functions to recast away. As well, the process of recasting and adding extra constraints is a manual one. It is conceivable that the technique could be automated to a certain degree.

```
fof(p1,conjecture,(! [T,C1,C2,C3,C,S] :
(
  (T > 0 &
   C1 > -0.1 & C1 < 0.1 &
   C2 > 0.8 & C2 < 1 &
   C3 > 0.9 & C3 < 1 &
   C^2 + S^2 = 1) =>

   ~(
        ((C1*C - C2*S)*exp(-0.1*T)) < 0.1 &
        ((C1*C - C2*S)*exp(-0.1*T)) > 0 &
        ((C2*C + C1*S)*exp(-0.1*T)) < 0.1 &
        ((C2*C + C1*S)*exp(-0.1*T)) > 0 &
           (C3*exp(-0.15*T) > 0.5) &
        (C3*exp(-0.15*T) < 0.8))
   ))).
```

Figure 3.9: Particle example recasted

3.3.6 Verification of Timed Reachability Properties

So far, our only concern has been on the verification of safety properties. That is, we want to ensure that a dynamical system never enters an unsafe state. Of further interest are timed properties, where we want to show that a dynamical system will enter a target state within a certain amount of time.

Taking the same moving particle example, we now want to show that for all trajectories starting in the initial region $x_1 = [-0.1, 0.1], x_2 = [0.8, 1], x_3 = [0.9, 1]$, the following properties hold.

- Property 1: The particle will pass through the orange plane, defined by $x_3 = 0.5$, within 10 seconds.
- Property 2: The particle will reach the blue box defined by $x_1 = [-0.2, 0.2]$, $x_2 = [-0.2, 0.2]$, $x_3 = [-0.2, 0.2]$ within 20 seconds and stay there for all time.

The plane and box described by the properties are shown in Figure 3.10. The initial set is shown as a green cube and the target set is shown as a blue cube. The trajectories originating from the end points of the initial set are shown in light blue.

Taking into account the techniques developed above, the timed verification specifications were encoded into the proper format and sent to MetiTarski. The results are shown in Table 3.4. Property 1 was proved with the default parameters; property 2 required the use of the recasting technique (labelled SC) to obtain a proof.

Table 3.4 highlights several interesting results. In particular, for property 2, using only the RCF timeout technique fails. This demonstrates that in many cases several proof techniques must be used to obtain a positive results. As well, the results show that in

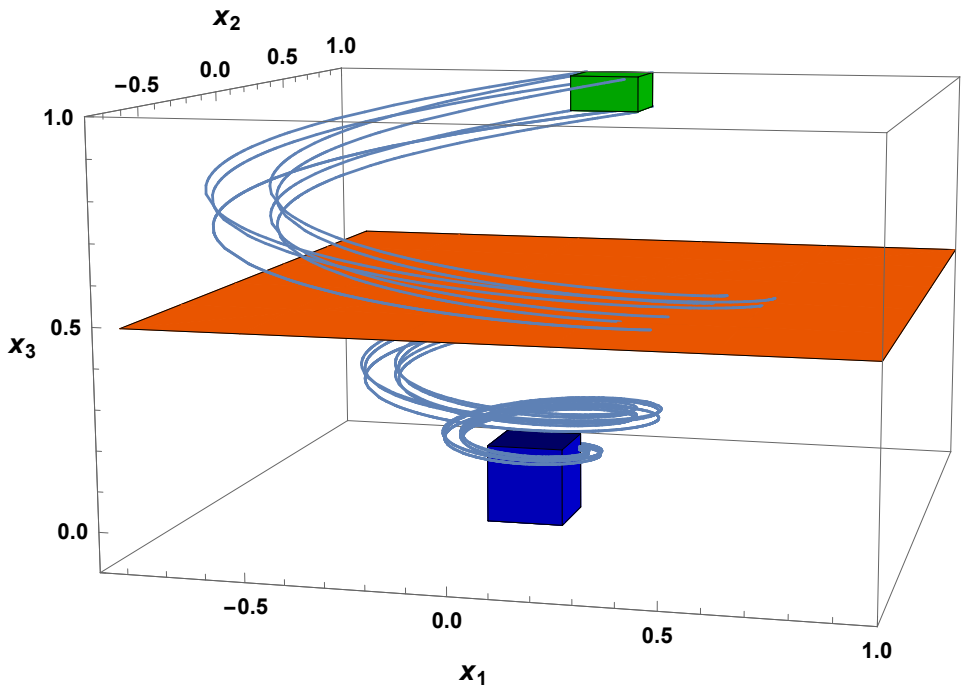


Figure 3.10: Verification of timed properties

Experiment	Proof Time (s)
Property 1	0.09 - 0.023 (Metis) + 0.068 (RCF)
Property 2 - Original	gave up
Property 2 - RCF time (1000ms)	timeout
Property 2 - SC Technique	12.05 - 0.378 (Metis) + 11.674 (RCF)

Table 3.4: Timed property proof times

comparison with the work of Sloth and Wisniewski [196], no abstraction is required to prove the timed properties. In the cited work, combination of discretisation and convex optimisation is used to construct a timed automaton that is then fed into a model checker. No information is given on the time it took to generate this abstraction or an estimation on its precision. I can only assume that its omission implies that its construction takes a significant amount of computational effort. The recasting technique has successfully allowed for the proof of a property which previously could not be proved. This again meets the criteria of success outlined for the experimental analysis of this chapter.

3.3.7 Beyond Simple Regions

The safe, unsafe and reachable regions have so far been defined by intervals over the state variables, resulting in cubes. In this section we show how it is possible to define regions over more complicated sets which can be verified using MetiTarski.

Let us consider a more complicated safety specification (the region shown in Figure 3.11). All trajectories starting in the green safe box defined above must avoid the set of unsafe states enclosed by the sphere $\frac{x_1^2}{0.09} + \frac{x_2^2}{0.09} + \frac{(x_3-0.5)^2}{0.09} < 1$. MetiTarski proved the conjecture describing this safety property in 14.74 seconds.

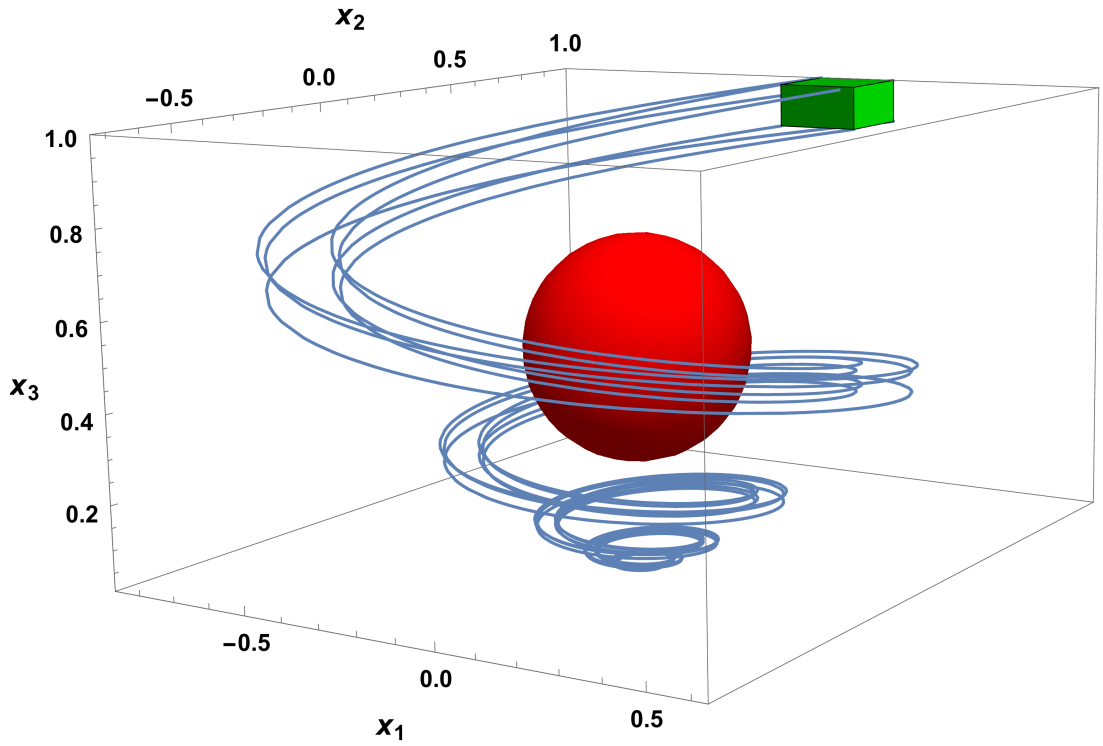


Figure 3.11: Verification of trajectories over a complex region

The importance of this result cannot be understated. Other work on verifying reachability properties depend on unsafe regions defined by simple linear inequalities. Using MetiTarski, no such restriction is required. For example, in the work of Sloth and Wisniewski [196], the developed abstraction techniques cannot work over the unsafe sphere shown in Figure 3.11.

3.3.8 Discussion

In this section, I have investigated the application of MetiTarski to the verification of linear continuous dynamical systems. Limiting the RCF time has shown to be a powerful technique for proving conjectures that are otherwise unprovable. A timeout of 1000 milliseconds gives a reasonable performance for the proofs. Another important technique is the recasting of sine and cosine terms to new variables. Combining the new variables with a single transcendental identity gives a significant performance boost. The use of these techniques has shown to be essential when attempting to verify the safety of continuous linear dynamical systems over complex regions of the state space.

3.4 Case Study: Flight Collision Avoidance

Conservative forecasts indicate that the number of aeroplane passengers flying each year will increase by a factor of 3, from 750 million to over 2.1 billion passengers over the next ten years [80]. This influx will push the current air transportation system beyond its safe limits. As the number of simultaneous flying aircraft increases, air traffic controllers (ATCs) who manually schedule and route the path of aircraft will be unable to handle the increased congestion. Currently, the only way to ensure that the entire system is safe is to keep aircraft grounded, potentially causing major delays [182]. This unacceptable situation has motivated the development of so called "free-flight" systems [217] that allow pilots to modify their scheduled routes without intervention from a ground based controller. Taking the ATCs out of the loop does have the potential to increase the throughput of airports. The automated support required by such a system will however need to be thoroughly verified before it is put into use.

Current collision detection and resolution systems that are installed in most commercial aircraft, such as the Traffic Collision Avoidance System (TCAS) [102], involve communication between two aircraft through transponders. If the system detects a potential midair collision, it will give instructions (to climb or descend) to each pilot and what path should be followed to resolve the situation. The main restriction is that the system only works between two aircraft. TCAS is therefore inadequate to deal with the amount of congestion expected in the near future.

Alternate collision avoidance algorithms, based on roundabout manoeuvres, that direct the aeroplanes to use curved flight paths have been proposed to address the scalability problem of TCAS. Initially put forward by Tomlin et al. [212] and then formally proved correct by Platzer and Clarke [170] using KeYmaera, the fully flyable tangential roundabout manoeuvre algorithm (FTRM) can handle up to five conflicting aircraft.

Another group of conflict detection methods deal with the problem of simultaneously landing multiple aircraft on narrowly separated runways during periods of reduced visibility. The Airborne Information for Lateral Spacing (AILS) [2] algorithm has been proved formally correct using several geometrical theories developed in PVS [38]. A more recent and complex algorithm is the Adjacent Landing Alerting System (ALAS) [167] which uses nonlinear projections to detect potential landing conflicts. Its implementation has been verified using a combination of switched models and dynamic analysis techniques relying on the guaranteed solutions of ODEs [76].

Due to the complexity of the proposed automated free-flight algorithms, simulation and testing are wholly inadequate for showing that they are free from errors [38]. To be able to prove the correctness of curved flight algorithms requires reasoning about the non-trivial nonlinear dynamics of aeroplane trajectories. Theorem proving solutions using systems such as PVS and KeYmaera have been successful at providing sound safety

results, however they rely on the skill of the user to develop the proofs. Under the right assumptions, the dynamical equations representing the trajectories of aircraft can be solved symbolically. The experiments described below show how MetiTarski can efficiently automate the verification of safety conditions on aircraft trajectories, which make up a significant part of certifying collision avoidance algorithms.

3.4.1 Flight Dynamics

The dynamics of an aircraft flying in the xy plane, shown in Figure 3.12, can be described by the following system of differential equations

$$\dot{x} = v\cos\theta \tag{3.3}$$

$$\dot{y} = v\sin\theta \tag{3.4}$$

$$\dot{\theta} = \omega \tag{3.5}$$

with v representing the linear velocity, ω the angular velocity and θ the counter-clockwise angular orientation of the aircraft with respect to the x-axis. (x, y) are the coordinates of the aircraft on the x axis and y axis respectively.

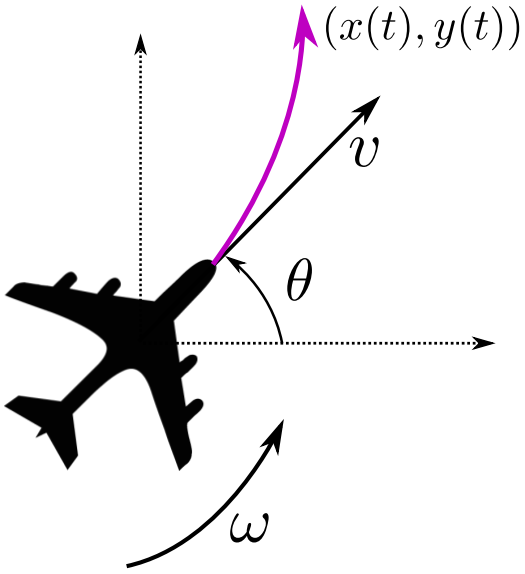


Figure 3.12: A flying aeroplane

3.4.2 Differential Axiomatisation

Using a recasting process (also called differential axiomatisation by Platzer [168, p. 150]) the nonlinear system can be transformed into a linear system which can then be solved symbolically. This is done by representing the velocity in each direction using the linear speed vector d,

$$d = (d_x, d_y) = (v\cos\theta, v\sin\theta)$$

assuming that an aircraft is flying at a constant linear velocity ($\dot{v} = 0$)[5] the following derivation holds

$$\dot{d_x} = (v\dot{\cos}\,\theta) = \dot{v}\cos\theta + v(-\sin\theta) = -(v\sin\theta)\omega = -\omega d_y \tag{3.6}$$

$$\dot{d_y} = (v\dot{\sin}\,\theta) = \dot{v}\sin\theta + v(\cos\theta) = v(\cos\theta)\omega = \omega d_x \tag{3.7}$$

The system of differential equations is now

$$\begin{aligned} \dot{x}(t) &= d_x(t) & \dot{y}(t) &= d_y(t) \\ \dot{d_x}(t) &= -\omega d_y(t) & \dot{d_y}(t) &= \omega d_x(t) \end{aligned}$$

with initial conditions

$$\begin{aligned} x(0) &= x_0 & y(0) &= y_0 \\ d_x(0) &= d_{x,0} & d_y(0) &= d_{y,0} \end{aligned}$$

3.4.3 Analytical Solution

Solving the system gives the following set of positional equations for the trajectories of two aircraft (labelled with subscripts 1 and 2 respectively) travelling in the xy plane. Their locations are (x_1, y_1) and (x_2, y_2), their respective linear speed vectors are labelled d and e, with the subscript x or y indicating the direction of each component. The initial values of each variable are subscripted with the number 0.

$$x_1(t) = x_{1,0} + \frac{d_{y,0}\cos(\omega_1 t) + d_{x,0}\sin(\omega_1 t) - d_{y,0}}{\omega_1} \tag{3.8}$$

$$y_1(t) = y_{1,0} - \frac{d_{x,0}\cos(\omega_1 t) - d_{y,0}\sin(\omega_1 t) - d_{x,0}}{\omega_1} \tag{3.9}$$

$$x_2(t) = x_{2,0} + \frac{e_{y,0}\cos(\omega_2 t) + e_{x,0}\sin(\omega_2 t) - e_{y,0}}{\omega_2} \tag{3.10}$$

$$y_2(t) = y_{2,0} - \frac{e_{x,0}\cos(\omega_2 t) - e_{y,0}\sin(\omega_2 t) - e_{x,0}}{\omega_2} \tag{3.11}$$

3.4.4 Safety Property Definition

To ensure that a particular collision avoidance manoeuvre is safe, each aeroplane is considered to be surrounded by a protected region that no other aircraft is allowed to enter. The requirement can be represented as a safety property over the trajectories that specifies the two aircraft always maintain a minimum separation distance p. This can be described

[5]This assumption, as highlighted by Platzer [168, p. 150], is reasonable since during curved turns it is not fuel-efficient to vary the linear speed of the aircraft.

mathematically as

$$\forall t : (x_1(t) - x_2(t))^2 + (y_1(t) - y_2(t))^2 \geq p^2$$

In the experiments below, the appropriate safe distance between two aircraft is chosen to be 2 nautical miles. The property is verified over a time interval of 2 minutes. This is an adequate amount of time for a potential collision to be detected and for corrective measures to be taken.[6]

3.4.5 Experimental Setup

By combining the safety property and the closed form solution of the trajectories of the two aircraft, the resulting first-order formula is easily translatable into a form that can be proved by MetiTarski. The first experiment starts by assigning the initial conditions ($x_{1,0}$, $y_{1,0}$, $x_{2,0}$, $y_{2,0}$, $d_{x,0}$, $d_{y,0}$, $e_{x,0}$, $e_{y,0}$) and angular velocities (w_1, w_2) to specific values, reducing the problem to one variable (time in this case). Figure 3.13 shows a plot of this single trajectory. The resulting one variable collision avoidance problem can be expressed as the MetiTarski input of Figure 3.14, which was proved in 0.082 seconds. The subsequent experiments assign to each previously fixed initial condition an interval. The number of continuous variables is incremented by one from one experiment to the next. This process makes the verification condition progressively more general by considering an entire set of trajectories, rather than a single one.

$$\begin{aligned} w_1 &= -0.01 \\ w_2 &= 0.01 \\ d_{x,0} &= 0.1 \\ d_{y,0} &= 0.1 \\ e_{x,0} &= -0.1 \\ e_{y,0} &= -0.1 \\ x_{1,0} &= -9 \\ x_{2,0} &= 10 \\ y_{1,0} &= -1 \\ y_{2,0} &= 10 \end{aligned}$$

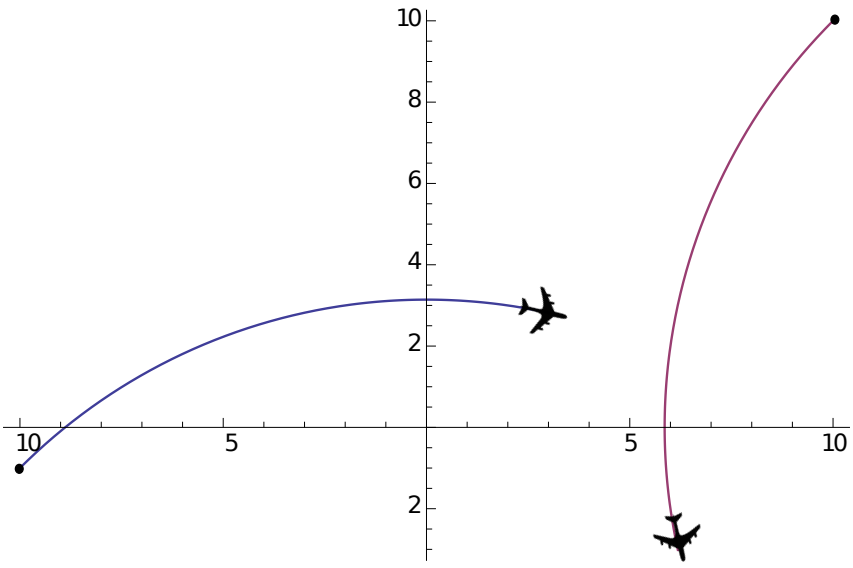


Figure 3.13: Flight trajectories

[6] Appropriate bounds on the response times are given by the TCAS family of standards

```
fof(airplane_easy,conjecture,
  (! [T] :
    (
      ( T>0 & T<120 =>
        882 - 400*cos(0.02*T) - 1200*sin(0.01*T) > 4)))).
```

Figure 3.14: MetiTarski input for flight collision verification

3.4.5.1 Range Splitting

I have already presented two techniques that can greatly improve proof times: limiting the RCF timeout and the recasting of transcendental functions. There is a third technique that can be used in cases when the conjecture to be proved is bounded over a time interval. If the conjecture proves too difficult for the RCF decision procedure, then consider splitting the problem into several, hopefully easier, overlapping subproblems.[7] If each separate conjecture is proved then this implies that the conjecture arising from the original problem is true over the entire original range.

For instance, in the aircraft collision example presented above, when the problem moves from one variable to two (t ranging from $[0, 120]$ and $x_{1,0}$ ranging from $[-9, -10]$), MetiTarski times out in 600 seconds. If instead, the conjecture is limited over the three intervals $[0, 60]$, $[60, 90]$ and $[90, 120]$, MetiTarski can prove each interval respectively in 0.092, 28.96 and 0.183 seconds.

The technique works as follows: if a conjecture cannot be proved over an entire time range, the interval is split in half resulting in two sub-problems. MetiTarski can then attempt to prove each separate problem. If either of the new problems fails (times out or gives up), then the failing range is split again in half. This can continue until some predefined minimum time interval width is reached. If the conjecture is still not proved at this point, then use the other developed techniques.

One benefit of splitting the conjecture into many subproblems is that it allows for the analysis of separate problems with different proof options. For instance, one range might need more precise upper and lower bounds or perhaps a higher RCF timeout. If any of these options were used on the original time interval then it is likely that the global timeout would be hit.

3.4.6 Experimental Results

The results of applying the range splitting technique to the aeroplane flight collision avoidance problem are shown in Table 3.5. The experiment name is suffixed with the number of variables that appear in the problem file. The proof times are in seconds. The max

[7] This strategy was inspired by interval analysis techniques [145].

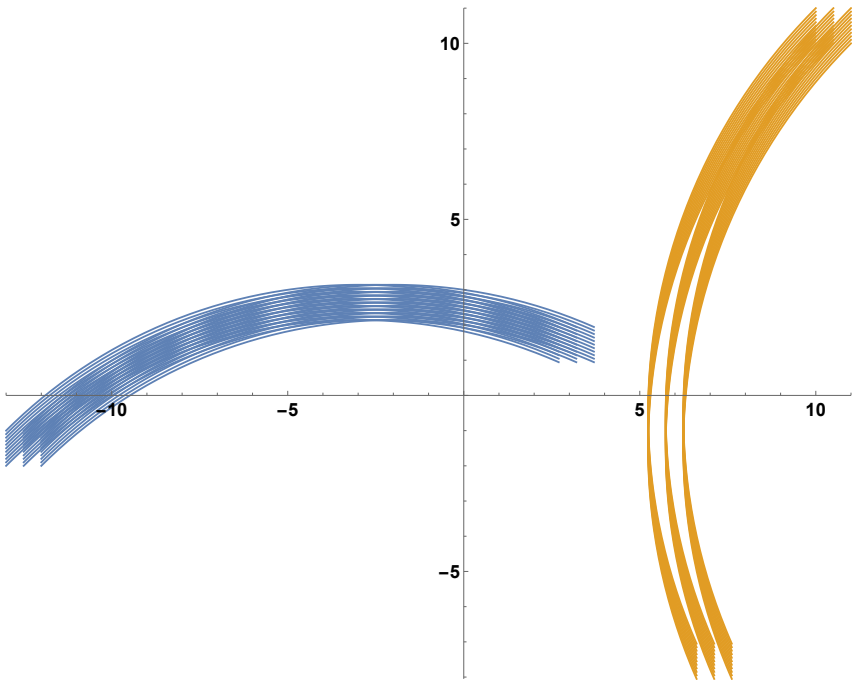


Figure 3.15: Multiple flight trajectories

and min columns list the proof times for the easiest and hardest interval. A visual representation of the analysis is shown in Figure 3.15, where several trajectories of the aircraft are superimposed.

Experiment	# of Splits	Min Proof time	Max Proof time	Total Proof time
AA-2p-1v	2	0.085	0.085	0.170
AA-2p-2v	3	0.092	28.96	29.235
AA-2p-3v	3	0.121	74.612	74.73
AA-2p-4v	3	2.184	18.125	32.369
AA-2p-5v	5	2.479	39.128	176.37
AA-2p-6v	4	15.69	97.981	278.22
AA-2p-7v	4	24.93	26.087	126.7
AA-2p-8v	4	26.04	27.144	107.11
AA-2p-9v	3	25.86	29.772	82.337

Table 3.5: Aircraft avoidance experimental results

The results show that as the number of continuous variables increased, MetiTarski and the RCF decision procedure nlsat had to work harder to find the proof. This behaviour is expected when using procedures to decide the satisfiability of nonlinear arithmetic. The complexity of the proof search rises rapidly with an increase in the number of variables.

There was however an unexpected decrease in total proof time when the number of variables was increased from six to seven. It would appear that the problem became easier for MetiTarski in some way. This can be explained by the architecture of RCF decision procedures and their performance on high dimensional problems. They generally depend

on heuristics for applying the computationally expensive sections of their algorithms. This can drastically change the movement through the search space. It is possible that in this case the additional variable might have affected the underlying quantifier elimination process. The order of variables can be the differentiating factor of the success of RCF decision procedures such as Cylindrical Algebraic Decomposition (CAD) [113].

It is important to note that the flight collision avoidance experiments presented in this section are the highest variable count problems successfully proved with the MetiTarski theorem prover to date. This positive result was the primary motivation behind using MetiTarski and nlsat in developing the hybrid system verification framework of Chapter 5.

3.5 Chapter Summary

This chapter has provided an introduction to the MetiTarski theorem prover and discussed its applicability to verify the safety of a restricted class of linear continuous dynamical systems. The experiments have guided the development of several techniques which are essential for proving reachability properties of problems with more than 5 continuous variables. The techniques are:

- Progressively limiting the amount of time used by the RCF decision procedure.
- Recasting transcendental terms with polynomial variables and coupling them with extra polynomial constraints.
- Iteratively splitting a variable range to convert a single difficult problem into several easier subproblems.

Each technique developed in this chapter had a specific goal: to allow MetiTarski to prove conjectures in a reasonable amount of time. They were each developed to address different difficult parts of the proof search of the flight collision avoidance protocol example of this chapter. The three techniques each satisfy the criteria for the success of the experimental effort. Each can reduce the amount of time required by MetiTarski to find a proof for higher dimensional verification problems.

As the number of continuous variables increases, it becomes difficult to predict whether MetiTarski will be successful. The best strategy therefore relies on using a combination of the three proposed techniques when a proof cannot be found. Although the classes of systems and properties analysed in this chapter have been restricted to linear continuous systems, the next chapters will demonstrate that the developed techniques will be useful and in many cases necessary for verifying the safety of nonlinear and hybrid systems with MetiTarski.

CHAPTER 4

PVS/MetiTarski Integration

In the previous chapter I described how the automated theorem prover MetiTarski was used to verify safety reachability properties of *linear dynamical systems*. Some systems can be accurately represented by a linear model or approximation. However, most real systems, which are affected by phenomena such as friction and vibrations, can be best modelled using nonlinear functions. Furthermore, models with nonlinear terms rarely admit a closed form symbolic solution and consequently the approach presented in Section 3.6 cannot be used.

One verification method that can handle nonlinear models is *interactive theorem proving*. The dynamics of the system are formalised within a specified logic and then a proof assistant is guided to construct theorems concerning the system's trajectories. The Prototype Verification System (PVS) contains several theoretical developments and strategies for proving theorems over the theory of the reals, allowing for the specification and verification of nonlinear systems. When encountering particularly difficult theorems that cannot be proved by its internal methods, PVS can alternatively invoke *external back-ends* including model checkers, SAT solvers, SMT solvers and various other decision procedures to discharge proof goals.

This chapter describes the design, implementation and testing of the integration of MetiTarski and PVS, which greatly extends the automated capabilities of PVS for dealing with real numbers. This development takes the form of a proof strategy that designates MetiTarski as a trusted oracle, which allows individual logical sequents to be isolated and analysed during an interactive proof session. When the strategy is invoked, PVS sequents containing real-valued formulas are automatically translated into first order formulas and submitted to MetiTarski. If the formula is successfully proved, then the corresponding sequent in the PVS proof is closed. Otherwise, the sequent is returned to the proof assistant unchanged.

I begin with a brief description of the architecture of PVS. This is followed by an overview of some of the currently available back-ends. I then present the new `metit` strategy and demonstrate how it works internally. Finally, the results of applying the proof procedure to several examples originating from the NASA PVS Library [1] are presented.

The results show that MetiTarski is considerably faster and more powerful than other strategies for nonlinear arithmetic that are available to PVS. The experiments of this chapter not only compare the abilities of PVS to MetiTarski, but they further reinforce the choice of using MetiTarski and the RCF decision procedure nlsat for the hybrid system verification framework of Chapter 5.

4.1 Preliminaries

4.1.1 Prototype Verification System

The Prototype Verification System (PVS) [160] is a formal logic based verification environment that consists of a specification language and an interactive theorem prover. The PVS specification language is strongly typed and supports predicate subtyping. In particular, the numerical types are defined such that `nat` (natural numbers) is a subtype of `int` (integers), `int` is a subtype of `rat` (rationals), `rat` is a subtype of `real` (reals), and `real` is a subtype of the primitive type `number`. All numerical constants, including fractions and numbers in decimal notation, are members of `number`. The subtyping hierarchy of numerical types and the fact that rational arithmetic is built-in makes PVS well suited for proofs involving real numbers. In particular, ground numerical expressions are automatically simplified by the PVS theorem prover. For example, the numerical expression `1/3+1/3+1/3` is simplified to `1` and this simplification does not require a proof.

The PVS theorem prover is based on a sequent calculus, whose rules of inference define a form of natural deduction. Each logical statement is represented as a *sequent* that contains a set of hypotheses (antecedents) and a set of conclusions (consequents). The goal is to build a proof tree where each leaf is a sequent that is shown to be true. PVS will start with a single sequent at its root. Proof rules are then applied interactively and will potentially generate several children sequents that are added to the proof tree. The focus of the prover then shifts to the newly generated sequents, which must each be proved true (closed) to complete the proof. PVS provides *strategies* that combine frequently used proof steps and decision procedures into a single command. For example, the strategy `grind` performs many automated steps including term re-writing, propositional simplification and calls to a decision procedure for linear arithmetic.

The NASA PVS Library contains several strategies for manipulating and simplifying real number formulas. The Manip package [74] for instance, provides rules for performing non-trivial manipulation of real-valued expressions. This includes operations for swapping, negating, grouping terms, multiplication and factoring. The Field [148] package provides support for the simplification of real valued goals, most importantly providing a grind proof rule that can be applied to sequents containing real numbers (`grind-reals`).

The most advanced proof strategies for nonlinear real number proving are `interval` [61, 153], `bernstein` [149] and `sturm` [154]. These strategies are based on provably correct interval arithmetic, Bernstein polynomial approximations and applications of Sturm's Theorem for univariate polynomials, respectively. The strategy `interval` automatically discharges sequent formulas involving transcendental and other special functions. The strategy `bernstein` automatically discharges simply-quantified multivariate polynomial inequalities. Finally, the strategy `sturm` can discharge existential and universal univariate polynomial inequalities.

4.1.2 Back-ends for Interactive Theorem Provers

The theorem proving approach employed by PVS (and other proof environments such as Coq and Isabelle/HOL) is based around describing a system and its safety properties using logical formulae. A proof checker is then manually guided to show that the properties logically follow or are implied by the original system definition. PVS in particular, can make use of decision procedures to efficiently automate many of the trivial steps required for the construction of proofs. This may significantly reduce the number of sequents that must be closed manually, making it much easier to develop complex proofs.

The integration of external tools is a commonly used technique for aiding the development of proofs inside interactive theorem provers. For instance, Paulson and Blanchette [166] developed Sledgehammer that connects Isabelle with the automatic theorem provers E [192], SPASS [216], Vampire [180] and Z3 [65]. Powerful proof reconstruction is provided by Metis [114], which translates the results of the external tools back into a series of sound Isabelle proof steps. Sultana et al. [202] have recently extended Sledgehammer support for the higher order theorem provers LEO-II [22] and Satallax [36]. Meng et al. [142] developed a C program termination checker for Isabelle's Hoare logic that can be called either as a trusted oracle or a proof method.

Similarly, PVS can use external back-end tools to handle fragments of its logical theories that are easily mechanised. For example, the μ-calculus was defined in the PVS specification language to tightly integrate a CTL model checker with the PVS proof checker [176]. In this case, a strategy was implemented to rewrite proof goals into the proper model checker input format. Similarly, the SMT solver Yices [77] can be called on se-

quents that contain logical combinations of uninterpreted functions, linear real and integer arithmetic. The RAHD system [164] can be called to decide the satisfiability of existential formulas containing real multivariate polynomials.

Other interesting extensions to PVS include the ground evaluator, the implementation of semantic attachments [57] and the PVSio library [147]. These components generate executable Lisp code from PVS specifications making it easy to rapidly build and simulate prototype implementations. PVSio has recently been integrated with Simulink [140], allowing for the verification of discrete logic controllers defined in the Stateflow language from within PVS. Another powerful extension is the random test generator built into PVS [159]. Its goal is to minimise the time wasted trying to prove false theorems, by finding counterexamples as early as possible in the development process.

Decision procedures that are built and formally verified to be correct within PVS are sound by design. However, external tools including SMT solvers and bespoke provers such as RAHD and MetiTarski must be trusted as external oracles and are assumed to implement sound methods. In a safety verification framework, such assumptions are dangerous. One practical solution to this problem is the Evidential Tool Bus (ETB) proposed by Rushby [184]. The ETB is a general framework for integrating verification tools with the broad goal of managing the assertions (e.g. an inequality is false or a propositional sentence is satisfiable) made by each individual component.

An example of where the ETB could be used is in the case of two competing external solvers that operate on the same domain. Consider one to be the *reference* that has been formally verified but is less efficient than the other, which uses heuristics to gain a speed up. The ETB allows proof goals to be closed with the more efficient tool, while keeping track of the sequence of judgments made. When certification is required, the reference solver would drop in seamlessly from the tool bus. The architecture of the ETB allows it to easily handle the replaying of judgments without any interaction from the user.

In summary, PVS is well suited for interfacing with external tools that act as decision procedures for simplifying proof goals. Currently, we must trust these tools to give the correct answer. The ETB can provide a framework to keep track of claims and evidence for aiding the certification of the results.

4.2 The PVS Strategy metit

The proof strategy that integrates the automated theorem prover MetiTarski into the PVS theorem prover is called `metit`. This strategy is currently available as part of the NASA PVS Library for PVS 6.0 [1].

```
  |-------
{1}   FORALL (v, phi:real): abs(phi) <= 35 AND v ## [|200, 250|] IMPLIES
        abs(180*9.8*tan(phi*pi/180)/(pi*v*0.514)) < 3.825

Rule? (metit)
Metitarski Input =
fof(pvs2metit,conjecture, (![V1, PHI2]: (((abs(PHI2) <= 35) & (200 <=
V1 & V1 <= 250)) => (abs((((180*(98/10))*tan(((PHI2*pi)/180)))/((pi*V1)
*(514/1000)))) < (3825/1000))))).
SZS status Theorem for tr_35.tptp
Processor time: 0.680 = 0.184 (Metis) + 0.496 (RCF)
Trusted source: MetiTarski.
Q.E.D.
```

Figure 4.1: Automated proof of formula 4.1 using metit

In its simplest form, the strategy `metit` can be used to prove universally-quantified formulas involving real numbers such as

$$\forall v \in [200, 250], |\phi| \leq 35 : \left| \frac{180\, g}{\pi v\, 0.514} \tan(\frac{\pi \phi}{180}) \right| < 3.825 \tag{4.1}$$

where $g = 9.8$ (gravitational acceleration in meters per second squared) and π is the standard transcendental constant. This formula, which appears in the formal verification of an alerting algorithm for parallel landing [151], states that for an aircraft flying at a ground speed of between 200 and 250 knots and a maximum bank angle of 35 degrees, the angular speed is less than 3.825 degrees per second.

Figure 4.1 shows Formula 4.1 as a sequent in PVS. The double hash symbol "`##`" is the inclusion operator of closed intervals, which are denoted using the parenthesis operator "`[| |]`". The sequent, which consists of one universally-quantified formula in the consequent, is automatically discharged by the proof strategy `metit` in less than one second. The strategy uses PVS's internal utilities to parse the sequent. If the sequent is recognised as a set of first order formulas involving real numbers, the strategy translates the sequent into a TPTP formula and submits it to MetiTarski. If MetiTarski returns *Theorem*, the result is trusted by PVS and the sequent is closed. If MetiTarski returns *Timeout* or *GaveUp* then the sequent in question is returned back to PVS unchanged. Application of other proof strategies would be required at this stage.

Although universally-quantified real-number formulas such as Formula 4.1 occur in the verification of complex systems, a more common use case for the strategy `metit` is in the context of an interactive proof of a large theorem where multiple formulas appear in a sequent. The strategy `metit` only considers sequents that are sets of first order formulas containing real-number inequalities between transcendental and special functions.

However, the user may optionally specify formulas of interest in a given sequent. Other formulas in the sequent will be ignored by the strategy. The user can also specify the formulas of interest that are to be sent to MetiTarski.

Moreover, in an interactive theorem prover such as PVS, sequent formulas may also involve data structures such as records, arrays, tuples, and abstract data types. For example, the sequent in Figure 4.2 appears in a lemma that characterises aircraft trajectories that are repulsive [155]. This sequent consists of 12 antecedent formulas and one consequent formula. All of the formulas are quantifier-free, but free-variables (Skolem constants, in PVS terminology) occurring in the sequent can be understood as universally-quantified variables. In addition to the real variable `eps`, this sequent involves record variables `v`, `rd`, `dv`, and `mps`, which represent vectors in a 2-D Euclidean space.

The strategy `metit` does not directly deal with data structures. However, it recognises that an expression such as `v`x`, which accesses the field `x` of 2-D vector variable `v`, denotes a real-number variable. The strategy will appropriately translate record and tuple access expressions to variables in the TPTP syntax. Furthermore, the strategy `metit` allows the user to specify the formulas of interest that are to be sent to MetiTarski. The proof command `(metit *)`, where the asterisk symbol "`*`" specifies all formulas in the sequent, translates the 13 formulas of the sequent into a TPTP formula involving 9 variables. This particular TPTP formula is discharged by MetiTarski in less than 0.2 seconds.

Further analysis of the sequent in Figure 4.2 reveals that all the formulas in the sequent are necessary to discharge it. For instance, invoking the strategy with `(metit (^-1))`, which uses all formulas except the first one in the antecedent, fails to prove the sequent. In total, the proof of the lemma where this particular sequent appears requires 171 invocations of `metit`. Including all the other proof rules, the lemma is proved in 37 seconds. The largest sequent discharged by `metit` in this proof involves 13 variables. None of these sequents can be discharged by any other automated strategies available to PVS.

The use case for the PVS strategy `metit` is ideally for lemmas containing transcendental functions and special functions. However, proofs of purely polynomial problems can also take advantage of the integration of PVS, MetiTarski and Z3. What distinguishes Z3 from other state-of-the-art SMT solvers is that its proof heuristics for nonlinear real arithmetic are customisable through a strategy language. MetiTarski itself also implements its own set of strategies that work in combination with those of Z3.

```
repulsive_criteria_iterative_reduces_seq_divergent_special.3.1.1.1 :
[-1]  eps = 1 OR eps = -1
[-2]  v`y*eps <= 0
[-3]  rd`y*eps < 0
[-4]  ((v`x = 0 AND v`y = 0) IMPLIES rd`x >= 0)
[-5]  ((v`x /= 0 OR v`y /= 0) IMPLIES rd`x > v`x)
[-6]  rd`x*v`y*eps-rd`y*v`x*eps <= 0
[-7]  mps`y*eps+rd`y*eps < 0
[-8]  v`x >= 0
[-9]  (dv`x /= 0 OR dv`y /= 0)
[-10] mps`x*rd`y*eps-mps`y*rd`x*eps <= 0
[-11] -1*(dv`x*mps`y*eps)-dv`x*rd`y*eps+ dv`y*mps`x*eps+dv`y*rd`x*eps < 0
[-12] ((rd`x*mps`x+rd`x*rd`x+rd`y*mps`y+rd`y*rd`y < 0 AND
      dv`x*rd`y*eps-dv`y*rd`x*eps < 0) OR (rd`x*mps`x+rd`x*rd`x+
      rd`y*mps`y+rd`y*rd`y >= 0 AND dv`x*mps`x+dv`x*rd`x+dv`y*mps`y+
      dv`y*rd`y > rd`x*mps`x+rd`x*rd`x+rd`y*mps`y+rd`y*rd`y
      AND dv`x*rd`y*eps-dv`y*rd`x*eps <= 0))
  |-------
[1] (dv`x /= 0 OR dv`y /= 0) AND dv`y*eps < 0 AND ((v`x = 0 AND v`y = 0)
    IMPLIES dv`x >= 0) AND ((v`x /= 0 OR v`y /= 0) IMPLIES dv`x > v`x)
    AND dv`x*v`y*eps-dv`y*v`x*eps <= 0
```

Figure 4.2: Sequent involving 13 formulas and 9 variables

4.2.1 Strategy Application Example

The first stage of the `metit` strategy checks to see if it is being called on a single quantified sequent. In this special case, the variables will be contained in a list bound to the quantifier. Each variable that is further constrained by a predicate subtype will be lifted to an equivalent propositional form. For instance, consider the case of using the strategy on the following sequent

```
 |-------
{1}   FORALL (x: real, y: posreal, z: posreal):
        z*abs(x * y) <= 1 IMPLIES x * y <= abs(x) * y
```

the variables y and z, which have the type `posreal` (a positive real number), are lifted to $y > 0$ and $z > 0$. They are added as antecedents to the sequent, giving

```
  {-1} y > 0
  {-2} z > 0
  |-------
  {1} z*abs(x * y) <= 1 IMPLIES x * y <= abs(x) * y
```

This is the state that most proofs are in before calling `metit`. Now that the quantifiers have been removed, the next stage of the strategy involves converting each sequent into a propositional form. If `metit` has been called directly on a list of antecedents and consequents, then only those that are specified by the optional argument are used in the strategy call. If the argument is omitted, then they are all considered. Since the sequent calculus specifies that the conjunction of the antecedents imply a disjunction of the consequents, this is a simple matter of adding `AND`, `OR` and `IMPLIES` symbols to the expression. After the conversion step, the sequent expression now has the following propositional form

```
(y > 0 AND z > 0) IMPLIES (z*abs(x*y) <= 1 IMPLIES x*y <= abs(x)*y)
```

Next, each PVS variable is converted to uppercase and appended with a distinct numerical label, creating a list of valid MetiTarski variables. This step is required because PVS variables are case sensitive; MetiTarski variables must only contain uppercase characters and numbers. A conversion routine is then called recursively on the propositional expression. Each operator is isolated, its interpretation is confirmed[1] and it is translated into the correct TPTP syntax. For example, the first operator to be isolated on the expression above would be `IMPLIES`. This would be converted to the TPTP symbol for implication (`=>`), then each of its arguments would be converted in turn. On the left hand side the operator `AND` would be isolated next and on the right it would be `IMPLIES`. When a PVS variable is encountered, it is replaced with its correct MetiTarski interpretation. The previous expression produces the following MetiTarski input

[1] This is because every operator in PVS can be overloaded.

```
fof(pvs2metit,conjecture,
  (![X1, Y2, Z3]: (((Y2 > 0 & Z3 > 0) =>
    ((Z3*abs((X1*Y2)) <= 1) => ((X1*Y2) <= (abs(X1)*Y2)))))).
```

4.3 Experimental Results

To test the capabilities of the integration of MetiTarski with PVS that I built, the proof strategy `metit` was run on several examples originating from the PVS contribution `interval_arith`. These examples involve trigonometric and other special functions that are not supported by the strategies `bernstein` or `sturm`. The results are displayed in Table 4.1. Each row is a separate attempt to prove the specified lemmas. The second and third columns list the total proof time for the respective proof strategy.

On average, the reduction in proof times was by a factor of 18. In an interactive proof where multiple sub-problems of the type listed in Table 4.1 occur, the potential reduction in overall proof time is substantial. However, it should be noted that while `interval` is a proof-producing strategy (i.e. `interval` preserves the soundness of the PVS proof system), `metit` integrates MetiTarski and its RCF decision methods as trusted oracles into the PVS theorem prover.

Lemma	`interval` (s)	`metit` (s)	Speed up
sqrt23	1.39	0.154	9.27
sin6sqrt	1.76	0.120	14.67
sqrtx3	1.65	0.195	8.46
tr_35	1.97	0.680	2.77
tr_35_le	1.87	0.113	16.55
A_and_S	1.38	0.036	38.30
atan_implementation	2.55	0.154	16.56
ex1_ba	1.59	0.073	21.78
ex2_ba	1.51	0.049	30.82
ex3_ba	1.65	0.059	27.97
ex4_ba	1.71	0.078	21.92
ex5_ba	1.84	0.075	24.53
ex6_ba	1.60	0.105	15.24
ex7_ba	1.54	0.111	13.87

Table 4.1: Interval versus metit strategy run-times

4.4 Chapter Summary

Proving theorems over the reals with proof assistants such as PVS can require a significant amount of manual and computational effort. Sending difficult sub-problems to trusted oracles is an already accepted method for decreasing proof times. Providing MetiTarski as a back-end to PVS gives an efficient alternative for closing difficult sequents. Furthermore, it provides a way to quickly ascertain whether a proof is worth attempting with the verified methods built within PVS.

Since MetiTarski is able to use several external arithmetic decision methods (Mathematica, QEPCAD or Z3) itself to decide the satisfiability of RCF sentences, the strategy `metit` greatly expands the number of options available to PVS for automatically handling problems from the theory of the reals. The experiments show that the new strategy is considerably better than other methods currently available to PVS for closing sequents containing real-valued functions. The positive results further support the choice of using MetiTarski for verifying linear and nonlinear continuous dynamical systems. Consequently, it will play an important role in the abstraction framework presented in Chapter 5, where it will be used to abstract the continuous state space of nonlinear hybrid systems.

CHAPTER 5

Hybrid System Verification

This dissertation has been concerned so far with the verification of *continuous* dynamical systems modelled by ordinary differential equations. This is a valid starting point because a wide range of interesting applications (e.g. aircraft trajectories, particle motion) can be described by continuous functions that vary over $\mathbb{R}^n$. The experiments of Chapters 3 and 4 have shown that the automated theorem prover MetiTarski is well suited for handling complex high dimensional conjectures[1] that arise from the safety analysis of continuous dynamical systems.

There are certain types of dynamical system behaviour that cannot be properly represented by a purely continuous model. Consider, for example, systems that evolve both continuously and discretely due to instructions from an embedded controller or from interactions with the environment (e.g. unmanned aerial vehicles, self-driving cars and colliding particles). These are examples of *hybrid* dynamical systems. One popular modelling framework for hybrid systems is the hybrid automaton (see Section 2.3.1), which properly takes into account both continuous and discrete behaviours. The verification of reachability properties of hybrid automata is however quite difficult and in all but the simplest cases computationally intractable [106].

One common approach to solving the reachability problem of hybrid automata is via abstraction. The general idea is to algorithmically extract a finite discrete state system (the abstract system) from the original hybrid system (the concrete system) which then allows for the application of efficient model checking methods to perform an automated and exhaustive safety analysis.

When constructing an abstraction of a dynamical system the primary objective is to ensure that all properties of interest are preserved. The danger is the process can potentially remove safety violating behaviours from the model. For the abstraction to be useful it must be *sound*: all safety reachability properties verified with the abstraction must hold on the concrete model. Soundness is generally guaranteed by over-approximating

[1] As highlighted several times in this dissertation, high dimensional means 8 to 10 continuous variables

the reachable state space. Unfortunately, this can introduce behaviours that do not exist in the concrete system. To address this, the abstraction can be further restricted to be *complete*: all safety properties violated in the abstraction must also be violated in the concrete model. Soundness is much simpler to guarantee than completeness, due to the availability of sound but incomplete decision methods for deciding the type of conjectures generated during the construction of the abstraction. We will see that soundness is sufficient for proving safety properties of hybrid systems.

There have been several abstraction methods proposed for continuous and hybrid systems [11, 40, 41, 60, 139, 198], but they are generally restricted to those that have polynomial vector fields. The current methods therefore have difficulty handling commonly encountered physical effects such as vibrations, drag and friction that appear as transcendental functions in the system definitions. In this chapter, I present an enhanced abstraction method for nonlinear systems that has no such restrictions: the vector fields, guards and invariants can all contain arbitrary combinations of transcendental functions.

The hybrid system verification framework is based on the qualitative abstraction algorithm developed by Tiwari [209, 211] and subsequently implemented in the HybridSAL tool [208]. In his method, the continuous state space is discretised by continuous functions into qualitatively distinct regions. The algorithm generates proof obligations for checking the feasibility of the abstract states and to determine transitions between them. This is done by analysing how the derivatives of the discretising functions change with respect to the trajectories of the concrete system. A sound and incomplete decision procedure for the theory of RCF is used to prove the conjectures.

The development of the abstraction framework presented in this chapter began with the idea of using MetiTarski in place of the RCF decision procedure called by Tiwari's abstraction algorithm, allowing the construction of finite state abstractions of nonpolynomial hybrid systems. However, it was not known whether MetiTarski would be able to handle the type and size of conjectures being generated. Furthermore, even if MetiTarski could successfully be used to construct a sound abstraction, questions remained whether the qualitative abstraction process would be efficient enough to complete within an appropriate amount of time to be useful.

The positive results from the initial continuous system experiments supported the choice of using MetiTarski as a back-end decision procedure. I designed and implemented the hybrid system verification framework QUANTUM (Qualitative Abstractions for Nonpolynomial Models), which uses MetiTarski to decide the conjectures generated by the abstraction process. The initial version constructed a full discrete state abstraction that was then passed to the model checker NuSMV to verify safety reachability properties. This proved to be inefficient, primarily because of time wasted on conjectures that were irrelevant to the property of interest.

The next version of QUANTUM implemented a *lazy* form of abstraction that immediately terminated if a predefined safety property was invalidated (by transitioning into an unsafe state). This allowed the abstracter to focus completely on conjectures that contribute to disproving the required safety property. An additional improvement to the original algorithm was to distribute the calls to MetiTarski across several processes. This had a significant effect on decreasing the abstraction times, although it required extra care in keeping track of proved conjectures. The results of using QUANTUM on several benchmark hybrid system problems have shown that qualitative abstraction with MetiTarski is competitive with other verification methods for nonlinear hybrid automata [68]. These case studies will be presented in Chapter 6.

This chapter begins with a discussion of how abstraction is used to simplify verification tasks. This is followed by an introduction to the field of *qualitative reasoning*, which is the basis of the methods implemented in HybridSAL and QUANTUM. I then briefly discuss some of the external methods used to construct and verify discrete state abstractions. Next, I provide an overview of the Tiwari abstraction algorithm, showing how MetiTarski is used to verify the types of conjectures that are generated. I then describe the development of techniques that improve on the original implementation of QUANTUM.

5.1 Background

This section describes additional background material required to understand the qualitative abstraction methods used by HybridSAL and QUANTUM. The main contribution of this dissertation, which includes implementation details, begins in Section 5.2.

5.1.1 Verification by Abstraction

Abstractions are commonly used to simplify the understanding and operation of complex systems. Take for instance moving an automobile: all that must be known is that pressing the accelerator pedal will make it move forward. This abstraction allows an operator to ignore all other processes including electrical and mechanical signals controlling the flow of fuel to the engine, the chemical combustion process and the distribution of power to the wheels. Similarly, when operating a computer, a user does not have to consider the steps taken by the hardware, operating system and applications to make a letter appear on the screen when a key is pressed. These operational abstractions ignore all details that are irrelevant to the resulting behaviour. This general idea of neglecting features from a model, while maintaining only the behaviours of interest to make it easier to understand, is the basis of *verification by abstraction*.

In the verification by abstraction approach, a system under analysis that proves too complex for the available verification methods is abstracted into a simpler form, which should ideally be easier to verify than the original system. The critical requirement is that the abstraction maintains enough details to preserve all properties of interest. Figure 5.1 shows the relation between a concrete system and its abstraction. The abstraction procedure is denoted by α and the concretisation procedure, which returns the abstraction to its original domain, is denoted by γ. As behaviours of the original system are lost, the γ function will usually return an over-approximation of the original system. Finding accurate abstraction procedures and proper abstract domains is undoubtedly quite difficult. Several abstract domains are relevant to both the verification of computer programs and dynamical systems.

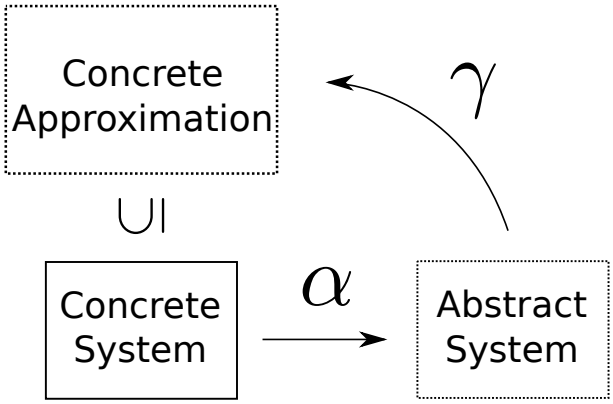


Figure 5.1: Verification by abstraction

Abstraction plays an important role in automating the deductive verification of software. It deals with proving properties (such as the absence of overflows) of the execution of computer programs. An abstract representation of a program and a property is constructed so that when validated, proves that a specific concrete property holds on a concrete program. One example of this type of method is *abstract interpretation* [56] for the static analysis of software. Sets of program executions are over-approximated by changing the domain of the program variables. For instance, integer variables can be abstracted over intervals, octagons or polyhedra. This was extended further to programs operating over an infinite state space by Graf and Saidi [91] with *predicate abstraction* that uses a Boolean abstract domain for variables. Predicates in the concrete domain are replaced by a conjunction of Boolean variables in the abstract domain. Predicate abstraction has been shown to be important for combating the state space explosion problem encountered in model checking [108] and has found applications in other domains including the verification of hybrid systems [13].

The same types of abstract domains can be used for the abstraction of continuous and hybrid dynamical systems. In this case, the trajectories or solutions of the system can be viewed as the "program executions". Here the general behaviour or flow of the vector field is analysed rather than the specific trajectories. The abstraction method for hybrid

systems investigated in this chapter, which was developed by Khanna and Tiwari [211], abstracts the system into discrete zones using continuous functions evaluated over their signs $(-, +, 0)$. This is a form of *data abstraction* and comes from the area of *qualitative reasoning* [128].

5.1.2 Qualitative Reasoning

Qualitative reasoning[2] (QR) is concerned with modelling and predicting the behaviour of complex dynamical systems that are incompletely specified. It is motivated by the fact that reasoning about physical processes can be performed quite naively and with incomplete knowledge. For instance, if we know that the rate of water flowing into a bathtub is greater than the rate of water flowing out, it can be inferred that the bathtub will eventually overflow. This conclusion is made without knowing the size of the bathtub, the current water height, the temperature of the water, the specific flow rates, etc. A qualitative abstraction allows us to ignore the specifics of a system that do not change the overall behaviour of the system. There are several QR methodologies that provide a modelling framework and sound inference methods that formalise this type of common sense reasoning.

De Kleer and Brown [64] developed a system called ENVISION that represents a dynamical system, such as an electrical circuit, as a group of components and conduits connected by terminals. *Components* represent specific objects that are governed by a linear function (e.g. a resistor), *conduits* passively transport information between components (e.g. a wire) and a *terminal* connects components and conduits. All real variables are evaluated over the *quantity space* $Q = \{-, 0, +\}$ and the qualitative value is derived by a function $QV : \mathbb{R} \to Q$, such that

$$QV(x) = \begin{cases} - & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ + & \text{if } x > 0, \end{cases} \tag{5.1}$$

The behaviour of the components are given in terms of *confluences* that impose constraints on the signs of the component's state variables. For example, a resistor, which is governed by the function $V = I \times R$, would correspond to the confluence $QV(V) = QV(I)$ because $QV(R) = +$. Qualitative values are added and multiplied according to the rules in Figure 5.2, where ? represents an undefined quantity (all three signs would be assumed in this case).

The possible sign combinations of all qualitative variables represent distinct states of the system. The qualitative model is "simulated" by first determining all possible solutions to the confluences according to rules shown in Figure 5.2. Then a transition analysis

[2]It is also known as qualitative physics.

$\oplus$	$-$	0	$+$
$-$	$-$	$-$	$?$
0	$-$	0	$+$
$+$	$?$	$+$	$+$

$\otimes$	$-$	0	$+$
$-$	$+$	0	$-$
0	0	0	0
$+$	$-$	0	$+$

Figure 5.2: Qualitative addition and multiplication

infers how the signs of the variables can change, while satisfying each confluence. The main assumption here is that we are dealing with continuously differentiable functions that admit only several possible transitions from one qualitative state to another. The result is a finite state machine with edges between abstract states.

Kuipers' [127] qualitative simulation (QSIM) method starts with abstractions of dynamical systems called qualitative differential equations (QDEs). The QDEs contain partially specified functions that reside in different monotonicity classes such as M^+ and M^-. M^+ is the class of monotonically increasing functions where for every $f \in M^+$ and $y(t) = f(x(t))$ we have $\frac{df}{dt} > 0$. M^- is the class of monotonically decreasing functions where for every $f \in M^-$ and $y(t) = f(x(t))$ we have $\frac{df}{dt} < 0$. The QDEs are further separated into several arithmetic primitives that impose constraints on the qualitative variables. Several of these qualitative constraints are shown in Table 5.1.

Qualitative constraints	Equational constraints
EQUAL(x, y)	$y(t) = x(t)$
ADD(x, y, z)	$z(t) = y(t) + x(t)$
D/DT(x,y)	$\dot{y}(t) = x(t)$
M+(x,y)	$y(t) = f(x(t)), f \in M^+$
M-(x,y)	$y(t) = f(x(t)), f \in M^-$

Table 5.1: Qualitative constraints

In direct contrast to ENVISION, which imposes a fixed structure on its qualitative model and variables, the QSIM framework allows continuous variables to be abstracted over several intervals that can be further subdivided when new qualitative behaviours are detected. To determine all reachable qualitative states, a constraint satisfaction problem is constructed, based on the initial state of the system and the constraints imposed by the QDEs.

QSIM can conveniently handle situations when a system's operating conditions (e.g. initial states, parameter values) might range over a set of values or have no specific numerical value (e.g. a chemical reaction might define concentrations as being "weak" or "strong") [63]. Non-numerical descriptions are used to capture the fundamental behaviour of the system. A qualitative model is constructed by abstracting each real variable by its own *quantity space*, which is a series of intervals with real endpoints called *landmark* values. The sign of the first derivative of the variables across the quantity space

determines how the system changes with respect to time. Time is represented as a series of *significant* time points $t_0 < t_1 < t_2 < \ldots t_k$ where the qualitative behaviour is designated to change.

The qualitative value of a real variable at time t is represented in terms of the landmark values and the direction that the system is moving at that point. Consider a real variable v, with a quantity space $l_1 < l_2 < \cdots < l_k$, its qualitative value $QV(v,t)$ is defined by the *qualitative magnitude* (qmag) and *direction* (qdir). If the system is currently directly on a landmark value then $qmag = l_j$, otherwise it is defined as the open interval between landmarks $qmag = (l_j, l_{j+1})$. The qualitative direction is defined in term of whether the first derivative of the variable is increasing (inc), steady (std) or decreasing (dec),

$$qdir = \begin{cases} inc & \text{if } \dot{v} > 0 \\ std & \text{if } \dot{v} = 0 \\ dec & \text{if } \dot{v} < 0 \end{cases}$$

The *qualitative state* of a system is a tuple of QV values for each variable of the system. The *qualitative behaviour* of a system is a sequence of qualitative states.

Example 5.1. Consider a simple model of an undamped spring in Figure 5.3 given by the system first order differential equations,

$$\dot{x} = v$$
$$\dot{v} = -\frac{k}{m}x$$

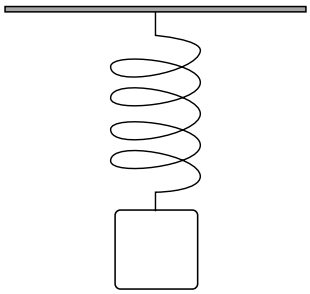


Figure 5.3: Simple undamped oscillating spring

where m is the mass on the spring, k is the spring elasticity and x is the vertical position of the mass with respect to the equilibrium position at rest. An example simulation trace (along with the parameter values) is shown in Figure 5.4.

The spring system is represented by the QDE in Figure 5.5, with $a = \dot{v}$, $v = \dot{x}$. The left-hand side column shows the quantity space and the corresponding landmark values. The right-hand side column are the qualitative constraints.

$$m = 2$$
$$k = 5$$
$$x(0) = 0$$
$$\dot{x}(0) = 1$$

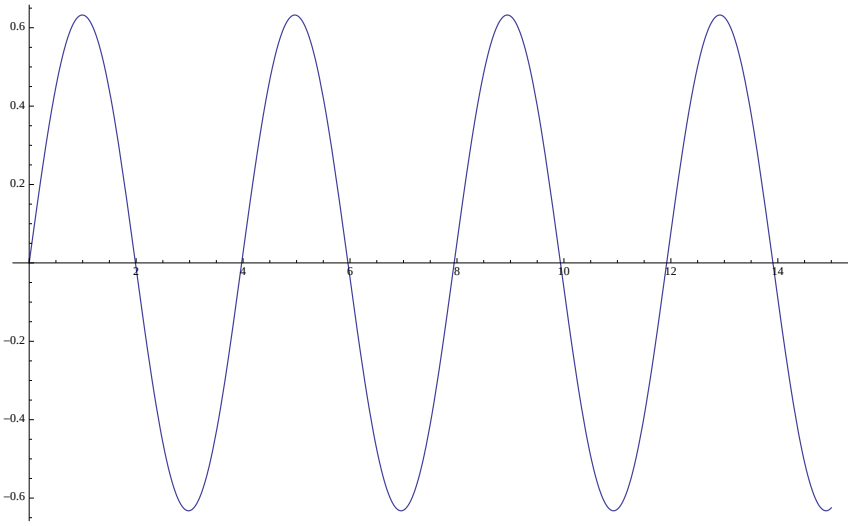


Figure 5.4: Simulation of the spring

$a : -\infty < 0 < \infty$	$d/dt(x, v)$
$v : -\infty < 0 < \infty$	$d/dt(v, a)$
$x : -\infty < 0 < \infty$	$M^-(a, x)$

Figure 5.5: Qualitative differential equation for the simple spring

5.1.2.1 Qualitative Simulation with QSIM

To analyse the resulting model, the QSIM algorithm produces a tree of all potential qualitative behaviours called an *envisionment*. QSIM must take into account all constraints[3] on the qualitative variables defined by the QDE. The qualitative simulation algorithm performs the following steps:

1. An initial state of the concrete system is converted into potentially several qualitative states, which are then put on the *agenda*.
2. Pop a state off the agenda.
3. For each qualitative variable of the popped state, determine all possible sign values that the variable can transition to.
4. Check whether the potential successor states are consistent with the constraints of the system and if so, add the transition to the behaviour tree.
5. Add each consistent successor state to the agenda.
6. If the agenda is empty, terminate the qualitative simulation loop. Otherwise, go to step 2.

[3]Similarly to the *confluences* in the ENVISION framework.

Example 5.2. Consider the above spring example with initial states $\{x(0) = 0, \dot{x} = v = 1\}$. The qualitative initial state at time t_0 is determined to be,

$$QS_{init} = \{QV(x, 0), QV(v, 0), QV(a, 0)\}$$
$$QS_{init} = \{x : \{0, inc\}, v : \{(0, \infty), std\}, a : \{0, dec\}\}$$

To determine the next qualitative state of the system at time t_1, all possible transition states in the interval (t_0, t_1) are generated. Since x is increasing, $QV(x, (t_0, t_1))$ is necessarily $\{(0, \infty), inc\}$. Since a is decreasing and v is steady, then $QV(a, (t_0, t_1))$ and $QV(v, (t_0, t_1))$ are respectively $\{(-\infty, 0), dec\}$ and $\{(0, \infty), dec\}$. The set of possible values for all qualitative variables in state t_1 deduced from those in the (t_0, t_1) interval are,

$$x = [\{\infty, inc\}, \{\infty, std\}, \{(0, \infty), inc\}, \{(0, \infty), std\}]$$
$$v = [\{0, dec\}, \{0, std\}, \{(0, \infty), dec\}, \{(0, \infty), std\}]$$
$$a = [\{-\infty, dec\}, \{-\infty, std\}, \{(-\infty, 0), dec\}, \{(-\infty, 0), std\}]$$

The QSIM algorithm then filters out all possible combinations of the qualitative values that violate the equational constraints from Table 5.1. For instance, the combination $[v : \{0, dec\}, x : \{\infty, inc\}]$ can be removed since it violates the constraint $D/DT(v, x)$ that enforces $QV(\dot{x}) = QV(v)$. The velocity of the mass cannot be both zero and increasing (greater than zero) at the same time. Similarly, the combination $[v : \{0, std\}, a : \{-\infty, dec\}]$ can be removed since it violates the constraint $D/DT(a, v)$. The acceleration of the mass cannot be both negative and steady (equal to zero) at the same time. The combination $[a : \{-\infty, std\}, x : \{\infty, inc\}]$ is removed since it violates the requirement $M^-(a, x)$: that the acceleration decreases as the velocity increases. Further filtering removes qualitative states with overlapping variables that do not share any sign values. Finally, the QSIM algorithm outputs the only feasible state at t_1 to be $[x : \{(0, \infty), std\}, v : \{0, dec\}, a : \{(-\infty, 0), std\}]$. This represents the mass at its highest position, with the spring compressed. The next state t_3 is determined to be $[x : \{0, dec\}, v : \{(-\infty, 0), inc\}, a : \{(-\infty, 0), inc\}]$. This process continues until no additional qualitative behaviour is found. The full discrete state qualitative abstraction of the undamped spring is shown in Figure 5.6.

Example 5.3. Consider now an undamped spring model that takes into account its *stiffness*. When a real spring is elongated or compressed, the force it exerts depends on the material it is made of. A hard spring will become stiffer as it is perturbed and consequently will exert a greater force. A soft spring will start to deform and will exert a weaker force. This type of behaviour can be modelled by adding a nonlinear term βx^2 to

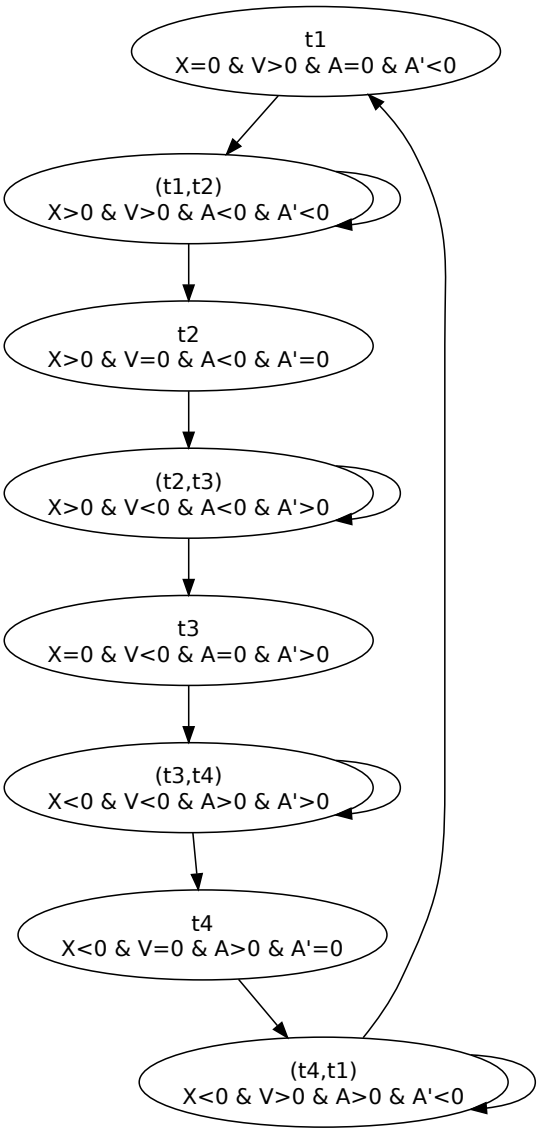


Figure 5.6: Discrete state abstraction of a simple spring

the system of differential equations,

$$\dot{x} = v$$
$$\dot{v} = -\frac{k}{m}x + \frac{\beta}{m}x^2$$

This simple addition of a nonlinear term to the problem causes problems for the QSIM algorithm. Take for instance the qualitative magnitude of $QV(x)$ being $(0, \infty)$. It follows that the magnitude of $QV(x^2)$ is $(0, \infty)$. Unfortunately, the qualitative constraint $ADD(x, -x^2, v)$ does not give a unique value for the magnitude of $QV(v)$ (see the $\oplus$ operator in Table 5.2), it is undefined and therefore can be any of $[(-\infty, 0), 0, (0, \infty)]$ regardless

$a : -\infty < 0 < \infty$	$d/dt(x, v)$
$v : -\infty < 0 < \infty$	$d/dt(v, a)$
$x : -\infty < 0 < \infty$	$M^{-}(a, x)$
$x^2 : -\infty < 0 < \infty$	$ADD(x, -x^2, v)$
	$MULT(x, x, x^2)$

Figure 5.7: Qualitative differential equation for the nonlinear spring

of the qualitative direction. In a full qualitative simulation this type of deduction would produce *spurious states* and *spurious transitions* that represent no real concrete trajectory of the original system of differential equations.

5.1.2.2 QSIM Extensions

There have been several improvements to the QSIM framework that aim to reduce the number of spurious states and transitions generated. For instance, new landmark values can be automatically introduced if it is detected that the continuous functions change signs at certain points (the roots of functions contained in the ODEs can be used). In some cases, taking the higher-order derivative can eliminate choices of qualitative values that violate continuity properties. Global constraints on the energy of the system can be used to remove qualitative behaviours that violate fundamental physical properties of the underlying system. One such property is that the sum of the potential and kinetic energy must remain constant (in an ideal system) or decrease (due to friction). This type of global constraint is particularly useful for dynamical systems such as the spring example above, where it will filter out transitions that move the mass higher or lower than its initial position when released from rest.

The behaviours predicted by qualitative simulation can be further refined if some numerical information of the underlying system is known. Qualitative simulation has been combined with interval constraint solving in the Q2 [129], Q3 [23] and NSIM [122] semi-qualitative methods. With Q2 and Q3, landmark values and other terms such as constants are given interval bounds. The qualitative constraints of a QDE imply a series of algebraic equations, where the landmark values are symbolic variables, that can be used to construct an interval constraint satisfaction problem. Interval arithmetic can be used to tighten the bounds that are found to be consistent between each significant time point. Intervals that are found to be inconsistent will result in the associated qualitative state being pruned from the envisionment. NSIM uses interval bounds as well as *dynamic envelopes* obtained via numerical simulation, which are upper and lower bounds of the trajectories of the underlying system of ordinary differential equations.

The guaranteed coverage theorem [128, p. 118] ensures that QSIM will predict all possible behaviours of the system of QDEs. The theorem implies that the resulting envisionment will be an over-approximation of the trajectories of the dynamical system. Based on this theoretical result, Shults and Kuipers [194] built the model checker TL for proving universal CTL* properties over the envisionments produced by QSIM. The model checker is used in two ways: to verify a property over a complete behaviour tree, and to discard generated branches that do not satisfy a temporal property during qualitative simulation. One weakness of this framework is due to how the QDEs are obtained. Generally they are constructed by hand, which can be a tedious process since there are restrictions on the types of functions (and the relations between them) that can be converted into a QDE. This conversion process is particularly problematic when the underlying ODEs contain transcendental functions [42]. The HybridSAL and QUANTUM approaches instead work with the system of differential equations directly. Consequently, it is easier to manipulate the type of models that are commonly encountered in engineering domains. Uncertain parameters of the system can remain as symbolic constants, thus retaining a similar level of abstraction as QDEs.

The Tiwari algorithm can be viewed as an extension of the basic qualitative simulation methodology that adds the ability to use arbitrary polynomials as landmark values instead of constants and intervals. This can result in the creation of much finer abstractions. Additionally, it allows for the definition of global constraints explicitly in terms of original state variables rather than on qualitative variables. Consequently, the abstraction algorithm is more general and simpler to implement. All decisions regarding how the sign of qualitative variables change are decided via calls to a theorem prover. The QUANTUM system takes this one step further by allowing arbitrary nonpolynomial functions to be used as landmarks.

Qualitative simulation and qualitative reasoning in general might seem weak because the generated discrete state models will often contain spurious behaviours. Even with this limitation, several recent developments have shown that qualitative methods can help reduce the complexity of difficult verification problems. Hinrichs et al. [109] developed a system that combines qualitative reasoning, geospatial data and probabilistic simulation to determine the movements of units in military battle planning. These *courses of actions* are an integral part of war-games and are usually determined and analysed manually. Qualitative data measured by satellites concerning the position of troops, their intended movement and the topology of the environment constrains the space of possible outcomes, reducing the overall analysis time. Bobrow et al. [27], Klenk et al. [124] developed a qualitative simulator that integrates with the OpenModelica [84] design tool that allows for *design exploration*. It is used to support the design of devices operating over different physical domains (electrical and mechanical). Qualitative analysis can be applied early in the design process when parameter variables are unknown. This integration was further extended to hybrid systems defined by differential algebraic equations [125].

The main contribution was the development of automatic methods to extract qualitative constraints and QDEs from Modelica models. This development provides designers with easy access to qualitative simulation, without requiring an extensive background knowledge in qualitative reasoning techniques.

5.1.3 Types of Abstraction

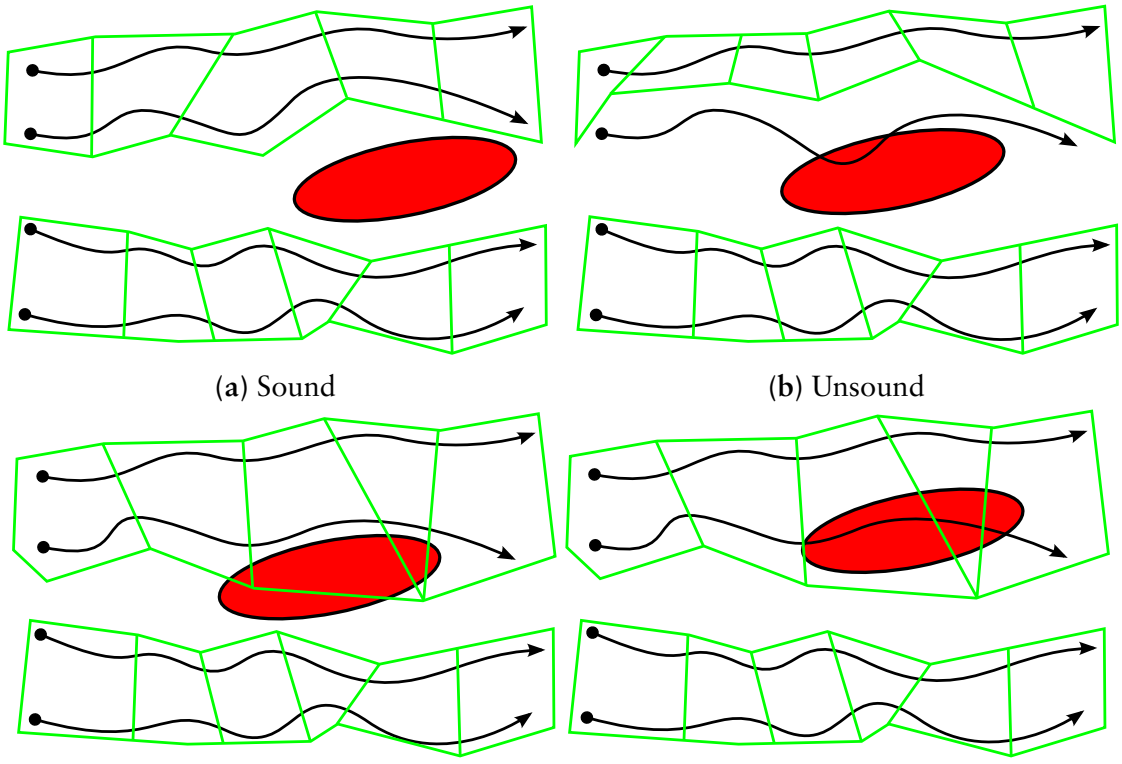
It is useful to characterise the dynamical system abstractions by the types of properties that can be verified. Let DS be a dynamical system and RP be a reachability safety property. α is the abstraction algorithm, $\alpha(DS)$ is the resulting discrete state abstract model and $\alpha(RP)$ is the reachability property defined in terms of the abstract variables.

Definition 5.1 (Sound Abstraction, [111, p. 259])**.** An abstraction procedure α is sound with respect to DS and RP if for any trajectory of DS that violates RP there exists a corresponding abstract trajectory of $\alpha(DS)$ that violates $\alpha(RP)$

The trajectories of a dynamical system are shown in Figure 5.8. The green polygons represent an abstraction of all possible trajectories of the system and the red ellipse represents the set of unsafe states. In Figure 5.8a the green polygon does not intersect the red ellipse. The abstraction is *sound* because no concrete trajectories of the system intersect the unsafe region. In Figure 5.8b the red ellipse is still not reachable by the abstraction. However, there is a trajectory of the concrete system that reaches the unsafe region. The abstraction is *unsound* because the reachability property is validated in the abstract model, but is in fact false. In summary, a sound abstraction ensures that there are no *false positives*; the safety of the discrete state abstraction always implies the safety of the original dynamical system.

Definition 5.2 (Complete Abstraction, [111, p. 259])**.** An abstraction algorithm α is complete with respect to DS and RP if for any abstract trajectory of $\alpha(DS)$ that violates $\alpha(RP)$, there exists a corresponding concrete trajectory of DS that violates RP.

An incomplete abstraction is shown in Figure 5.8c. In this case, no concrete trajectories actually reach the unsafe region, however the system is still incorrectly determined to be unsafe because the abstraction intersects with the red ellipse. Finally, in the complete abstraction in Figure 5.8d, all invalidated properties have a corresponding concrete trajectory entering an unsafe state. In summary, a complete abstraction ensures that there are no *false negatives*; an unsafe discrete state abstraction always implies that the original dynamical system is unsafe.



(a) Sound (b) Unsound

(c) Incomplete (d) Complete

Figure 5.8: Dynamical system abstraction types

5.1.4 Verification of Discrete State Abstractions

There are three main components to a formal verification framework. First, there must be an appropriate way to *model* how the system operates. Second, there must be a way to precisely define properties or specifications regarding the behaviour of the system. Third, there must be a way to show that the properties either hold or are violated by the model. A system will be deemed *verified* if it can be shown that all specifications concerning its behaviour are met. Of primary importance to dynamical systems are *safety* properties which, if proved, give a guarantee that some bad behaviour will never happen. This is an example of an *invariant*, which is a type of property that is true in all initial states and remains true along all trajectories of the system. Formal verification aims to exhaustively check that all safety specifications are invariants of the model.

For continuous and hybrid dynamical systems, the models can be defined respectively by differential equations and hybrid automata. The specifications are concerned with possible (continuous or hybrid) trajectories of the system. The properties of interest can be stated using natural language such as, "The air speed velocity of the aircraft will never fall below the stalling threshold", but in this form the requirements can be easily misunderstood. This can lead to the incorrect properties being analysed, possibly resulting in an *unsafe* system being labelled *safe*, and consequently negating the verification effort. Instead, a *formal* specification can be given in terms of a temporal logic, which augments

propositional logic with temporal operators making the specifications precise and unambiguous. The next section will introduce a method that uses temporal logic to properly specify properties of transition systems that can be verified automatically.

The qualitative reasoning methods described in Section 5.1.2 give a process to soundly abstract a continuous dynamical system into a finite structure (the envisionment). Since the abstraction process produces a discrete state over-approximation of the set of trajectories defined by a system of ODEs, verification techniques such as model checking can be used. When a reachability property is proved on the abstraction then it is guaranteed that the property holds on the original system.

5.1.4.1 Model Checking

Model Checking [52] is an automated and exhaustive method for verifying temporal properties of finite state models. The analysis begins in an initial state and then visits all reachable states while checking to see if the property under analysis is invalidated. If the property is proved false, then a *counterexample* is returned providing a trace listing the steps taken to reach the state that invalidates the property. The model checking problem can be formally stated as follows,

Definition 5.3 (Model Checking Problem)**.** Given a finite state system M containing a set of states S, an initial state $s \in S$ and a property ϕ specified in a temporal logic, the model checking problem is to find out whether $M, s \models \phi$, that is whether M satisfies (models) ϕ.

There are several types of temporal logics that can be used to specify properties that can be proved via model checking, including: linear temporal logic (LTL) [172], computational tree logic (CTL) [52] and their extensions. The temporal logics are separated into different classes based on their expressive power; there are some properties that can be defined in LTL that cannot be expressed in CTL and vice-versa. Certain extensions to the languages subsume both LTL and CTL allowing for more general properties to be specified. The different logics also require different model checking algorithms of varying complexity [183]. Only a restricted class of temporal operators are required when verifying the safety properties of dynamical systems.

LTL is a type of *linear time modal logic*. It describes linear traces, where a condition is checked to be true along a single path. LTL defines the operators in Table 5.2 regarding the propositional variables x and y.

CTL on the other hand is a *branching time modal logic*. Each moment in time can be split into a finite number of one or more traces. It adds to the LTL logic two path operators shown in Table 5.3, which must always be paired with an LTL operator.

Operator	Example
G: globally	Gx, x is always true
F: eventually	Fx, x is eventually true
U: until	xUy, x is true until y is true
X: next time	Xy, y is true in the next instant

Table 5.2: LTL operators

Operator	Example
A: always	AGx, x is true in all paths
E: exists	EFy, there is a path where y is eventually true

Table 5.3: CTL operators

CTL* [79] is a more expressive version of CTL that removes the restriction that path quantifiers must appear paired with LTL operators. Restricted subsets exist, for example: ACTL* and ECTL* each respectively drop one of the branching path operators. It is important to note that both LTL and CTL logics are proper subsets of CTL*, which contains all combinations of LTL and CTL formulas.

The most important improvements to standard model checking are methods that manage the *state explosion* problem. Real systems will generally have a large state space and consequently a large number of transitions. In many cases, it is impossible to hold all this information explicitly in memory. This greatly limits the sizes of systems that can be analysed. Symbolic model checking [49] addresses this problem by representing the automaton and the specification to be checked by Boolean functions. By using data structures such as Reduced Ordered Binary Decision Diagrams (ROBDDs), the corresponding systems representations take up less space and are easier to manipulate.

Binary decision diagrams are still limited by the fact that their sizes can grow exponentially [33] in the number of system variables. Instead of attempting to exhaustively prove a temporal property over an entire finite state machine, Bounded Model Checking (BMC) [26] searches for a counterexample trace of k steps. This is accomplished by representing the sequence of visited states using a Boolean formula by unrolling the transition relation k times. The verification of reachability properties is reduced to a Boolean satisfiability (SAT) problem that can be solved by one of the many optimised solvers for that domain. Although BMC can find states violating a property, in general it cannot show that no such state exists due to only searching up to a certain bound k. Note that there are some variants of BMC, notably k-induction, that are sound when a special type of property (inductive invariant) can be found and is shown to hold on the transition system.

Counterexample Guided Abstraction Refinement (CEGAR) [50] is another approach proposed to combat the state space explosion problem. Its strategy is to start with an abstraction of the system to reduce the amount of memory used. The abstraction is model

checked and due to its coarseness, will most likely return a counterexample. The validity of the counterexample is then checked by converting it back to the concrete domain. If it is a real counterexample then the process can stop as the property has been invalidated. Otherwise, the state that caused the spurious counterexample is refined. The process then repeats by model checking the newly created abstract system and analysing counterexamples in the same way. CEGAR is a useful technique because it is automatic and it keeps the transition systems small and manageable.

5.1.4.2 Simulation Relation

The idea of over-approximating the behaviour of one system by a simpler one is a strategy that can help reduce the complexity of a verification task. We have seen it applied so far to the analysis of dynamical systems: a continuous system can be discretised into a qualitative model that is guaranteed to contain all the behaviours of the original system. A similar concept can be applied to the analysis of finite state machines. In this case a complex state machine with a large state space and transition relation is replaced with a simpler one that exhibits identical observable behaviour. This can make the application of methods such as model checking more tractable if the number of states has been reduced by the process. This type of abstraction is formalised by the concept of a *simulation* between two transition systems.

Definition 5.4 (Discrete State Transition System)**.** An input-free discrete transition system is a tuple $(S, Init, t)$.

- S is a finite set of states, each state s labelled by an interpretation of variables $\{v_0, v_1, \ldots, v_k\}$ into some finite domain.
- $Init \in S$ is a set of initial states.
- $t : S \times S$ is the transition relation.

Definition 5.5 (Simulation, [93])**.** Discrete state transition system DS_1 simulates another DS_2 if for all $s \in Init_{DS_2}$

- DS_1 can start in a similarly labelled state as DS_2.
- For any transition DS_2 can make from state s_i to s_{i+1}, DS_1 can match it by transitioning to a state with the same label as s_{i+1}.

A simulation relation $SR \subseteq S_{DS_2} \times S_{DS_1}$ are pairs of corresponding states between the two transition systems DS_1 and DS_2. The existence of a simulation relation implies that for every sequence of state transitions $s_0, s_1, \ldots, s_k$ of DS_2 there exists a sequence of state transitions $s'_0, s'_1, \ldots, s'_k$ of DS_1 such that for every $i \leq k \in \mathbb{N}$, $SR(s_i, s'_i)$.

Theorem 5.1 (Preservation of Properties, [93]). If DS_1 simulates DS_2, then any ACTL* property satisfied by DS_1 is satisfied by DS_2.

Theorem 5.1 guarantees the soundness of the Tiwari algorithm for the abstraction of hybrid systems. By constructing a discrete state abstraction that simulates all externally observable behaviour of the original hybrid system, any safety property proved on the discrete state system will, by the preservation theorem, hold on the hybrid system.

The theorem leaves certain parts undefined. For instance it is ambiguous what is meant by "externally observable behaviour". For the Tiwari algorithm, this means all qualitatively different behaviour based on the sign of a chosen set of predicates. This behaviour is obtained by applying a qualitative analysis on the trajectories of the system over the entire set of predicates. The rest of this chapter presents a method to construct the discrete state abstraction.

5.1.5 Choice of Landmarks

The size of a qualitative abstraction depends crucially on the number of landmark values used to discretise the state space. The choice of landmarks also plays an important role in how well the behaviour of the concrete system is represented by the abstraction. The qualitative simulation methods described in Section 5.1.2 construct an abstraction made up of intervals that can be coarse, making it difficult to separate real behaviours from spurious ones. HybridSAL and QUANTUM on the other hand use continuous functions as the "landmarks": this allows for the construction of much finer abstractions that more closely follow the trajectories of the underlying system. Choosing the discretising functions however should not be done arbitrarily. Each additional function increases the number of states and transitions that must be checked by the abstraction algorithm, potentially increasing the total verification time.

In Section 2.2.4, the concept of a *Lyapunov function* was introduced. For a given dynamical system and a specific equilibrium point, the existence of a Lyapunov function implies that the equilibrium point is stable.[4] In Section 2.2.5, *barrier certificates* were described as being a generalisation of Lyapunov functions. The existence of a barrier certificate implies that for all trajectories starting within a region on one side of the barrier, it is impossible for them to reach another region on the opposite side. Lyapunov functions and barrier certificates are defined in such a way that trajectories pass through their level sets in one direction. We can take advantage of this property by using them as discretising functions to limit both the size of the state space and the number of transitions encountered during the abstraction process.

[4] A Lyapunov function is an energy-like function which shows that if a system is losing energy it will eventually come to rest.

The primary issue is that for general nonlinear systems, finding Lyapunov and barrier certificates is a difficult process. It usually requires constructing a candidate manually and then testing via simulation whether it satisfies the required conditions. This search has been made much easier by techniques introduced by Parrilo [163], where positivity and negativity conditions can be defined in terms of a series of formulas that are constrained to be a sum of squares. The search for parameters that satisfy these constraints can be reduced to a semidefinite programming (convex optimisation) problem. These sum of squares methods were implemented by Papachristodoulou et al. [162] in the SOSTOOLS Matlab toolbox, which provides a tractable method to generate candidate Lyapunov functions and Barrier Certificates.

Definition 5.6. A multivariate polynomial $p(x_1, \ldots, x_n) = p(x)$ is a *sum of squares* if there are polynomials $f_1(x), \ldots, f_m(x)$ such that,

$$p(x) = \sum_{i=1}^{m} f_i^2(x)$$

A polynomial $p(x)$ being a sum of squares (SOS) implies that $p(x) \geq 0$. This sufficient condition is useful for simplifying a wide range of complex problems that reduce to checking the non-negativity of polynomials.[5] The non-negativity conditions can be relaxed to a search for a sum of squares decomposition, which can be solved in (worst case) polynomial time by solving a semidefinite program [163].

The main restriction on the SOSTOOLS method of generating Lyapunov and Barrier Certificates for dynamical systems is that they must have polynomial vector fields. To be able to analyse nonpolynomial systems, the transcendental terms must be recast into a polynomial form (similar to the differential axiomatisation in Section 3.4.2). The recasting process introduces further inequalities that must be converted into sum of squares constraints. Once the system is in a purely polynomial form, SOSTOOLS can be used to search for a candidate function in terms of the recasted variables. A simple back-substitution returns the result to the original domain. This method allows the generation of nonpolynomial functions that can be used by the qualitative abstraction algorithm implemented in QUANTUM.

[5] This problem is NP-hard [152].

5.1.5.1 Generating Nonpolynomial Lyapunov Functions

Recall the conditions for the existence of a Lyapunov Function. For a dynamical system $\dot{x} = f(x)$ with an equilibrium point located at the origin ($x = 0$), if there exists a $V(x)$ that is a continuously differentiable function such that,

$$V(x) > 0 \qquad \text{for } x \neq 0 \tag{5.2a}$$

$$V(0) = 0 \tag{5.2b}$$

$$\frac{\partial V(x)}{\partial x} f(x) \leq 0 \qquad \text{for all } x \tag{5.2c}$$

then $x = 0$ is a stable equilibrium.

The conditions for the existence of a Lyapunov function can be transformed into the following series of SOS conditions [161] where $\phi(x) > 0$ for $x \neq 0$:

$$V(x) - \phi(x) \text{ is SOS} \tag{5.3a}$$

$$V(0) = 0 \tag{5.3b}$$

$$-\frac{\partial V(x)}{\partial x} f(x) \text{ is SOS} \tag{5.3c}$$

The positivity of $\phi(x)$ ensures that $V(x)$ is positive definite as required by Equation (5.2a). The negative sign in Equation (5.3c) ensures that $\dot{V}(x)$ is negative semidefinite as required by Equation (5.2c).

The previous description assumed that the vector field $f(x)$ was polynomial. To be able to apply the sum of squares procedure to a vector field that is nonpolynomial requires recasting all transcendental terms using the process developed by Savageau and Voit [191] and modified for use in a SOS framework by Papachristodoulou and Prajna [161]. For example, if sine and cosine terms appear, they can be replaced by the variables S and C, which induces the equality constraint $S^2 + C^2 = 1$. If the exponential function appears, it can be replaced by the variable E implying the constraint $E > 0$.

Assume that the original state variables are given by $o = (x_1, \ldots, x_n)$ and $r = (x_{n+1}, \ldots, x_{n+m})$ are new variables used to replace nonpolynomial terms. A nonlinear system $\dot{x} = f(x)$ can be represented in terms of these new variables as

$$\dot{o} = f_1(o, r) \tag{5.4a}$$

$$\dot{r} = f_2(o, r) \tag{5.4b}$$

The recasting process generates a series of explicit constraints of the type $r = F(o)$. There are also indirect constraints of the type $E(o, r) = 0$ and $GE(o, r) \geq 0$. E and GE are vectors, with the sign condition being satisfied entry wise. The domain of the

variables is given by a semi-algebraic set $D_1 \times D_2 = \{(o,r) \in \mathbb{R}^n \times \mathbb{R}^m : G_D(o,r) \geq 0\}$ where $F(D_1) \in D_2$. The Lyapunov existence conditions can be given in terms of the recasted system.

Definition 5.7 (Nonpolynomial Lyapunov Function, [161])**.** Assuming the explicit constraint $r_0 = F(0)$ and that there exists a scalar function $\phi(o,r)$ with $\phi(o,F(o)) > 0$ for all $o \in D_1, o \neq 0$ as above. If there exists a function $\tilde{V}(o,r)$, a vector of polynomial functions $\lambda_1(o,r)$, $\lambda_2(o,r)$, and a vector of sum of squares polynomials $\sigma_1(o,r)$, $\sigma_2(o,r)$, $\sigma_3(o,r)$, $\sigma_4(o,r)$ such that

$$\tilde{V}(0,r_0) = 0 \tag{5.5a}$$

$$\tilde{V}(o,r) - \lambda_1^T E - \sigma_1^T GE - \sigma_3^T G_D - \phi \text{ is SOS} \tag{5.5b}$$

$$-(\frac{\partial \tilde{V}}{\partial o} f_1 + \frac{\partial \tilde{V}}{\partial r} f_2) - \lambda_2^T E - \sigma_2^T GE - \sigma_4^T G_D \text{ is SOS} \tag{5.5c}$$

then $x = 0$ is a stable equilibrium of the original dynamical system $\dot{x} = f(x)$.

Proof. The equality constraints ensure that the $\lambda_i^T E$ terms are equal to zero. The $\sigma_i^T GE$ and $\sigma_i^T G_D$ terms are guaranteed to be greater than or equal to zero because each component of σ_k is a sum of squares polynomial and the sets G_D and GE are, by definition, positive semidefinite. Equation (5.5b) therefore reduces to

$$\begin{aligned}
&\tilde{V}(o,r) - \lambda_1^T E - \sigma_1^T GE - \sigma_3^T G_D - \phi \geq 0 \\
&\tilde{V}(o,r) \geq \sigma_1^T GE + \sigma_3^T G_D + \phi \geq 0 \\
&\tilde{V}(o,r) > \phi > 0 \\
&\tilde{V}(o,r) > 0
\end{aligned}$$

which satisfies Equation (5.2a), ensuring the Lyapunov function is positive definite. Similarly, Equation (5.5c) reduces to

$$\begin{aligned}
&-(\frac{\partial \tilde{V}}{\partial o} f_1 + \frac{\partial \tilde{V}}{\partial r} f_2) - \lambda_2^T E - \sigma_2^T GE - \sigma_4^T G_D \geq 0 \\
&-(\frac{\partial \tilde{V}}{\partial o} f_1 + \frac{\partial \tilde{V}}{\partial r} f_2) \geq \sigma_2^T GE + \sigma_4^T G_D \\
&(\frac{\partial \tilde{V}}{\partial o} f_1 + \frac{\partial \tilde{V}}{\partial r} f_2) \leq -\sigma_2^T GE - \sigma_4^T G_D \leq 0 \\
&(\frac{\partial \tilde{V}}{\partial o} f_1 + \frac{\partial \tilde{V}}{\partial r} f_2) \leq 0
\end{aligned}$$

and taking $V(x) = \tilde{V}(x, F(x))$, by the chain rule,

$$\frac{dV(x)}{dt} = \frac{\partial \tilde{V}}{\partial o} f_1 + \frac{\partial \tilde{V}}{\partial r} f_2$$

from which it follows that Equation (5.5c) satisfies Equation (5.2c), where the derivative of the Lyapunov function is guaranteed to be non-positive along the trajectories of the system. Equation (5.2b) is trivially satisfied by Equation (5.5a). □

Example 5.4. Consider the system of equations of the simple pendulum with friction (see Section 2.2.3), with $L = g$ and $b = 1$.

$$\dot{\theta} = \omega \tag{5.6a}$$
$$\dot{\omega} = -\sin\theta - \omega \tag{5.6b}$$

Two new variables are introduced $u_1 = \sin\theta$ and $u_2 = \cos\theta$, which add the constraint $u_1^2 + u_2^2 = 1$ to the system. $u_1' = \theta' \cos\theta = u_2\omega$ and $u_2' = -\theta' \sin\theta = -u_1\omega$. The resulting recasted polynomial system for the pendulum is

$$\dot{\theta} = \omega \tag{5.7a}$$
$$\dot{\omega} = -u_1 - \omega \tag{5.7b}$$
$$\dot{u}_1 = u_2\omega \tag{5.7c}$$
$$\dot{u}_2 = -u_1\omega \tag{5.7d}$$

The nonpolynomial Lyapunov function is given a template of

$$V = a_1\omega^2 + a_2u_1^2 + a_3u_2^2 + a_4u_2 + a_5 \tag{5.8}$$

which implies the constraint $a_3 + a_4 + a_5 = 0$ to make V(0,0) = 0. It is given by the following

$$V(\theta,\omega) = a_1\omega^2 + a_2\sin^2\theta + a_3\cos^2\theta + a_4\cos\theta + a_5$$
$$V(\theta,\omega) = a_1\omega^2 + a_2\sin^2\theta + a_3(1-\sin^2\theta) + a_4(1-2\sin^2\theta/2) + a_5$$
$$V(0,0) = a_3 + a_4 + a_5$$

The function ϕ is defined to be $0.1\theta^2$ to guarantee that V is positive definite. E is the equality constraint $u_1^2 + u_2^2 = 1$. The domain function G_D of x_1, x_2, u_1 and u_2 can each be defined by their upper and lower bounds with the following type of inequality $f_k = [f_l, f_u]$ is equivalent to $(f_k - f_l)(f_u - f_k) \geq 0$. Combining the constraints as described by Definition 5.7 and putting it in the correct syntax, SOSTOOLS returns the following candidate Lyapunov function,

$$V(\theta,\omega) = 11.79\sin^2\theta + 11.80\cos^2\theta - 9.98\cos\theta + 4.63\omega^2 + 5.25$$

A series of level sets, constructed using the candidate Lyapunov function, are depicted in Figure 5.9 as black curves. Each box contains a number that represents the specific energy value (the $V(\theta, \omega)$) along the corresponding curve.

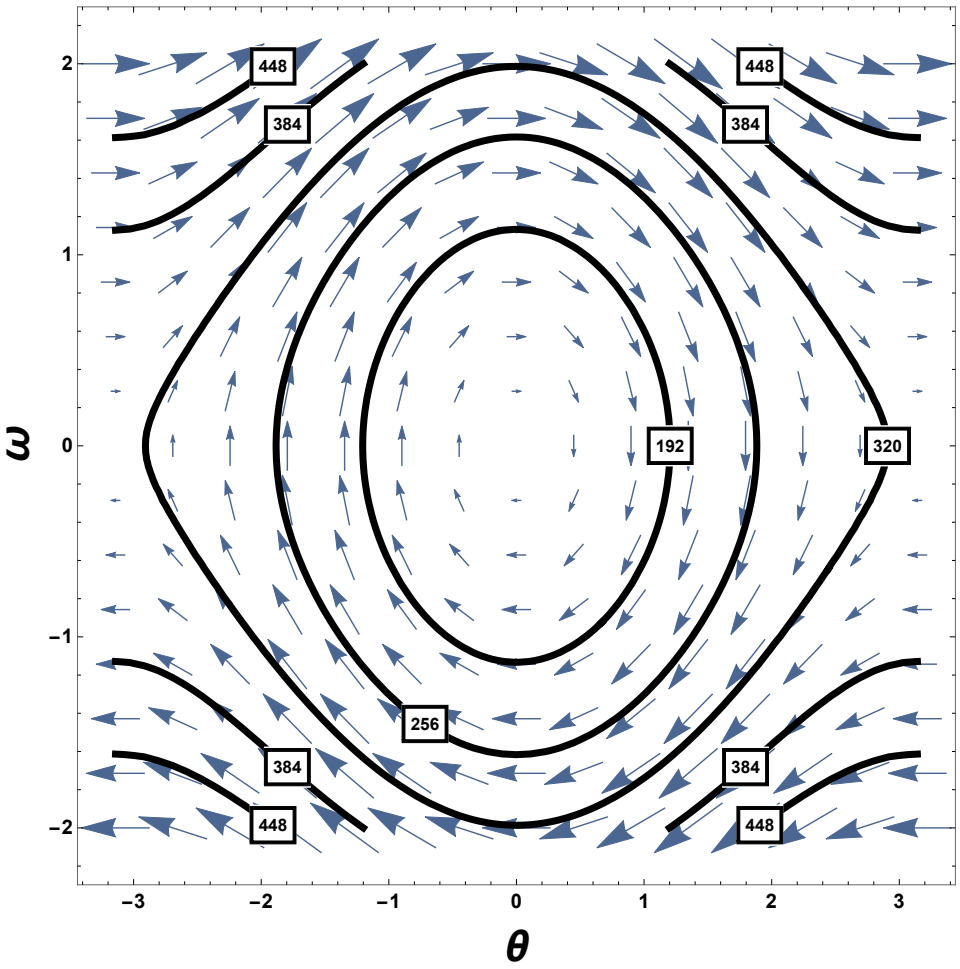


Figure 5.9: Level sets of the nonpolynomial Lyapunov function for the ideal pendulum

5.1.5.2 Generating Nonpolynomial Barrier Certificates

Recall the conditions for the existence of a barrier certificate. A dynamical system is defined over a state space χ. There is a region χ_u of unsafe states and a region χ_o of initial states. Consider a continuous function of state $B(x) : \chi \to \mathbb{R}$ that is differentiable with respect to its argument. If,

$$B(x) > 0 \qquad \text{for all } x \in \chi_u \tag{5.9a}$$

$$B(x) \leq 0 \qquad \text{for all } x \in \chi_o \tag{5.9b}$$

$$\frac{dB(x)}{dt} \leq 0 \qquad \text{for all } x \in \chi \tag{5.9c}$$

then all trajectories of the system starting in χ_o will never reach the states in χ_u and therefore the system is safe. As with nonpolynomial Lyapunov functions, the conditions can be reformulated in terms of a sum of squares decomposition problem.

We consider the regions χ, defined by the inequalities over the intervals $\chi = [\chi_l, \chi_u]$, to be represented by $G_k = (\chi - \chi_l)(\chi_u - \chi) \geq 0$. We reuse the same recasting variable names from the Lyapunov function discussion above, namely o, r, E and GE.

Definition 5.8 (Nonpolynomial Barrier Certificate)**.** Suppose there exists a positive number ϵ, vectors of sums of squares polynomials $\sigma_{\chi_u}(o,r)$, $\sigma_{\chi_o}(o,r)$, $\sigma_{\chi}(o,r)$, $\sigma_1(o,r)$, $\sigma_2(o,r)$, $\sigma_3(o,r)$, vectors of polynomials $\lambda_1(o,r)$, $\lambda_2(o,r)$, $\lambda_3(o,r)$, such that

$$\tilde{B}(o,r) - \epsilon - \sigma_{\chi_u}^T G_{\chi_u} - \sigma_1^T GE - \lambda_1^T E \text{ is a SOS} \tag{5.10a}$$

$$-\tilde{B}(o,r) - \sigma_{\chi_o}^T G_{\chi_o} - \sigma_2^T GE - \lambda_2^T E \text{ is a SOS} \tag{5.10b}$$

$$-(\frac{\partial \tilde{B}}{\partial o} f_1 + \frac{\partial \tilde{B}}{\partial r} f_2) - \sigma_{\chi}^T G_{\chi} - \sigma_3^T GE - \lambda_3^T E \text{ is a SOS} \tag{5.10c}$$

then $B(x) = \tilde{B}(x, F(x))$ is a barrier certificate, proving that all states starting in χ_o cannot reach χ_u.

The correctness of Definition 5.8 follows closely that of the correctness of Definition 5.7 on page 95. Instead of forcing the barrier certificate to be positive or negative definite in the entire state space, it needs to only satisfy this constraint over the sets χ_u and χ_o. Positive definiteness of $\tilde{B}$ over χ_u is given by,

$$\begin{aligned}
&\tilde{B}(o,r) - \epsilon - \sigma_{\chi_u}^T G_{\chi_u} - \sigma_1^T GE - \lambda_1^T E \geq 0 \\
&\tilde{B}(o,r) - \epsilon \geq \sigma_{\chi_u}^T G_{\chi_u} + \sigma_1^T GE - \lambda_1^T E \geq 0 \\
&\tilde{B}(o,r) - \epsilon \geq 0 \\
&\tilde{B}(o,r) \geq \epsilon \\
&\tilde{B}(o,r) > 0
\end{aligned}$$

negative semidefiniteness of $\tilde{B}$ over χ_o is given by,

$$\begin{aligned}
&-\tilde{B}(o,r) - \sigma_{\chi_o}^T G_{\chi_o} - \sigma_2^T GE - \lambda_2^T E \geq 0 \\
&-\tilde{B}(o,r) \geq \sigma_{\chi_o}^T G_{\chi_o} + \sigma_2^T GE + \lambda_2^T E \geq 0 \\
&-\tilde{B}(o,r) \geq 0 \\
&\tilde{B}(o,r) \leq 0
\end{aligned}$$

negative semidefiniteness of $\frac{d\tilde{B}}{dt}$ over χ is given by,

$$\begin{aligned}
&-(\frac{\partial \tilde{B}}{\partial o} f_1 + \frac{\partial \tilde{B}}{\partial r} f_2) - \sigma_\chi^T G_\chi - \sigma_3^T GE - \lambda_3^T E \geq 0 \\
&-(\frac{\partial \tilde{B}}{\partial o} f_1 + \frac{\partial \tilde{B}}{\partial r} f_2) \geq \sigma_\chi^T G_\chi + \sigma_3^T GE + \lambda_3^T E \geq 0 \\
&-(\frac{\partial \tilde{B}}{\partial o} f_1 + \frac{\partial \tilde{B}}{\partial r} f_2) \geq 0 \\
&(\frac{\partial \tilde{B}}{\partial o} f_1 + \frac{\partial \tilde{B}}{\partial r} f_2) \leq 0
\end{aligned}$$

Example 5.5. Consider the same pendulum with friction as Example 5.4. We now ask whether any trajectory starting in a region defined by $\chi_o = 0.5 - ((\theta - 1)^2 + (\omega)^2) \geq 0$, will reach a region defined by $(\chi_u = 0.25 - (\theta)^2 + (\omega - 2.5)^2) \geq 0$. The constraints on the domain of the original variables o, the recasted variables r and the equality condition E are the same as before. The barrier certificate is given a template of the form,

$$B = a_1 u_1^2 + a_2 u_1 u_2 + a_3 u_1 u_2 + a_4 u_1 + a_5 u_2^2 + a_6 u_2 x_1 + a_7 u_2 + a_8 x_1 + a_9 x_2^2 + a_{10}$$

Combining the constraints with the template given above, writing it in terms of Definition 5.8 and putting in the correct syntax, SOSTOOLS returns with the following barrier certificate.

$$\begin{aligned}
B(\theta, \omega) = &- 225.80 \sin^2\theta + 64.05 \sin\theta \cos\theta + 1106.57\omega \sin\theta \\
&- 19.66 \sin\theta + 588.99 \cos^2\theta - 19.66\theta \cos\theta \\
&- 1355.87 \cos\theta - 64.05\theta + 1254.37\omega^2 - 1266.76
\end{aligned}$$

The zero level set defining the barrier between the two regions is shown as a black line in Figure 5.10, and several other level sets are shown as red lines. Notice how the barrier certificate conditions ensure that all trajectories pass through the level sets in one direction.

5.2 Qualitative Abstraction of Hybrid Systems

In this section I will describe how the HybridSAL [211] abstraction algorithm has been implemented in the QUANTUM system. We begin by showing how the two types of dynamics of a hybrid system, discrete and continuous, are each abstracted to a discrete state model. It will be convenient to recall the definition of a hybrid automaton used to model hybrid systems (see Section 2.3 for more details).

Definition 5.9 (Hybrid Automata, Lygeros [135])**.** A hybrid automaton H is a collection $H = (Q, X, f, Init, Inv, E, G, R)$ where

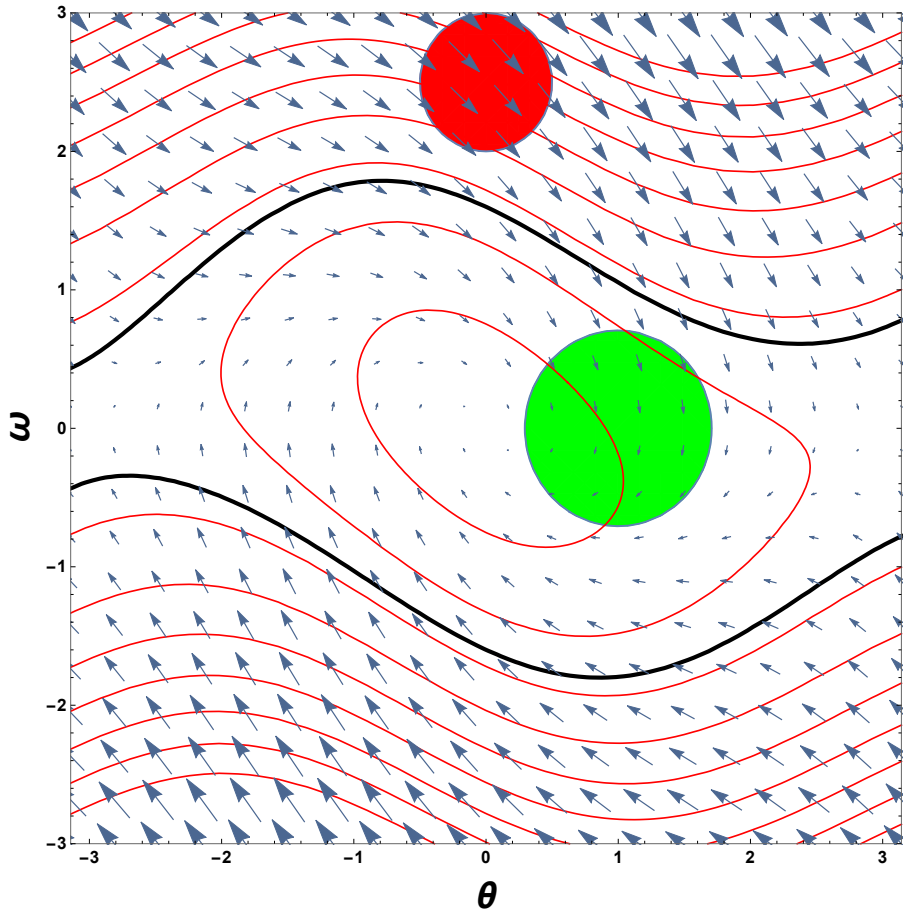


Figure 5.10: Level sets of the nonpolynomial barrier certificate for the pendulum with friction

- $Q = \{q_1, q_2, \dots\}$ is a set of discrete states
- $X \subseteq \mathbb{R}^n$ is a set of continuous states
- $f : Q \times X \to \mathbb{R}^n$ is a vector field
- $Init \subseteq Q \times X$ is a set of initial states
- $Inv : Q \to 2^X$ is an invariant set for each discrete state
- $E \subseteq Q \times Q$ is a set of edges
- $G : E \to 2^X$ provides guard conditions for each edge
- $R : E \times X \to 2^X$ is a reset map

5.2.1 Abstracting the Continuous State Space

Let us first focus on abstracting the continuous part of the hybrid system. The basic idea is to use a finite set F of k smooth functions,

$$F = \{f_1, f_2, ..., f_k \mid f_i : \mathbb{R}^n \to \mathbb{R}\}$$

to discretise the continuous state space into qualitatively distinct regions. There are several automatic and manual methods that can be used to choose the functions to include in F, as we will investigate several ways below. One strategy is to start with the guards, invariants and vector field as a source of functions.

Taking $Sign = \{zero, pos, neg\}$, the abstract state space is $Sign^k$. The abstraction function is defined as $\alpha : \mathbb{R}^n \to Sign^k$ where

$$\begin{aligned} \alpha(x) &= (s_1, ..., s_k) \text{ for } i = 1..k \\ s_i &= \begin{cases} pos & \text{if } f_i(x) > 0 \\ zero & \text{if } f_i(x) = 0 \\ neg & \text{if } f_i(x) < 0 \end{cases} \end{aligned} \tag{5.11}$$

Each abstract state s can be associated with a set of values from the original domain by the concretisation function $\gamma : Sign^k \to 2^{\mathbb{R}^n}$.

$$\gamma(s) = \bigwedge_{i=1..k} f_i(x) \sim_{s_i} 0 \tag{5.12}$$

where $\sim_{pos}$ is $>$, $\sim_{zero}$ is $=$ and $\sim_{neg}$ is $<$.

The first step of the abstraction algorithm is to remove all infeasible states. That is, we try to prove for a certain abstract state s_k that the first order formula $\nexists x : \gamma(s_k)$ is True, indicating that the abstract state s is indeed infeasible. The second step of the abstraction algorithm is to determine all potential next abstract states. For continuous transitions this is done by analysing the sign of the Lie derivative of the abstraction functions with respect to the vector field of the system.

Definition 5.10. The Lie derivative of a function $f : \mathbb{R}^n \to \mathbb{R}$ along the vector field $v \in \mathbb{R}^n$ is

$$L_v f = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} v_j \tag{5.13}$$

Definition 5.11. The sign of the Lie derivative constrains the possible signs of the discretising functions f_i in the next abstract state s' according to the following rules.
There is a transition from state s to s' if and only if for all $i = 1..k$

1. If $s_i = pos$
   (a) If $\gamma(s) \Rightarrow L_v f_i \geq 0$ then $s'_i = pos$
   (b) Otherwise $s'_i \in \{pos, zero\}$
2. If $s_i = neg$
   (a) If $\gamma(s) \Rightarrow L_v f_i \leq 0$ then $s'_i = neg$
   (b) Otherwise $s'_i \in \{neg, zero\}$

3. If $s_i = zero$

   (a) If $\gamma(s) \Rightarrow L_v f_i > 0$ then $s_i' = pos$

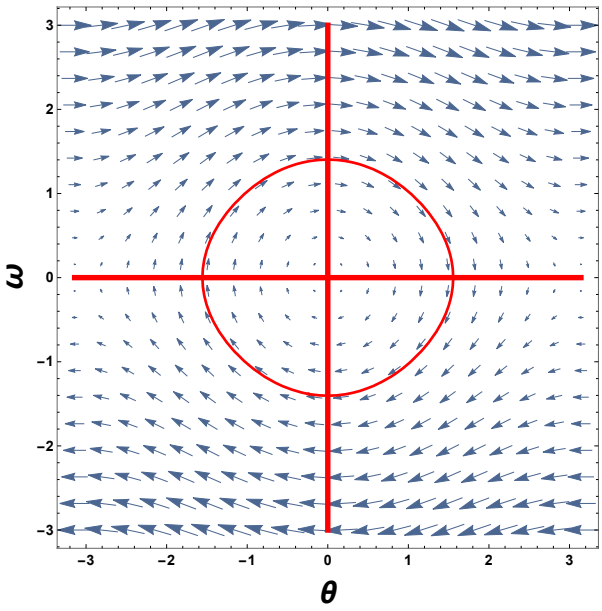
   (b) If $\gamma(s) \Rightarrow L_v f_i < 0$ then $s_i' = neg$

   (c) If $\gamma(s) \Rightarrow L_v f_i = 0 \ \wedge \ \nabla f_i \neq 0$ then $s_i' = zero$

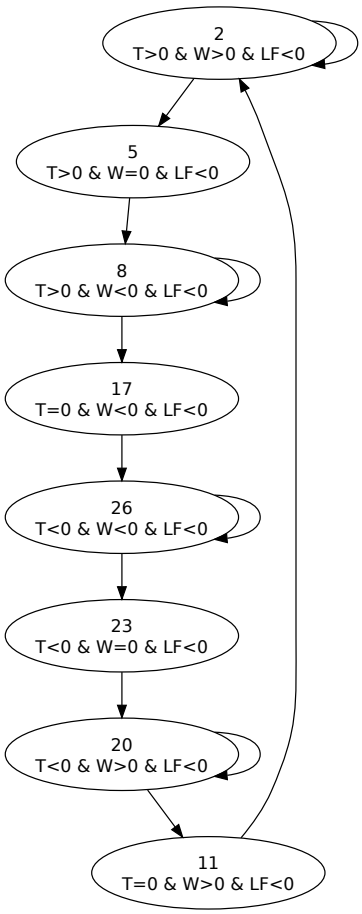
   (d) Otherwise $s_i' \in \{neg, pos, zero\}$

Once the signs of the abstraction functions in the next state are determined, the process repeats. Each newly reachable state is checked to determine if it is feasible and the next states from that new state are also computed. This continues until no new state is found.

Example 5.6 (Ideal Pendulum)**.** Consider again an ideal pendulum as an example. The discretising set of functions are chosen to be $F = \{\theta, \omega, 226.41 + 48.1144\omega^2 - 96.2288\cos\theta\}$. The third term is a level set of a Lyapunov function for the system. Figure 5.11a shows the discretised state space.



(a) Discretised state space



(b) Abstraction

Figure 5.11: Qualitative abstraction of the ideal pendulum

The abstraction algorithm generates the required first order formula to check the feasibility of abstract states. To prove that a particular abstract state is infeasible, that is $\nexists x : \gamma(s_k)$ is True, the equivalent universally quantified logical sentence $\forall x : \neg\gamma(s_k)$ is

sent to MetiTarski. Below in Figure 5.12 is a sample input for feasibility checking of an abstract state of the pendulum that starts hanging straight down ($\theta = 0$) and is given a small push ($\omega > 0$ and $226.41 + 48.1144\omega^2 - 96.2288\cos\theta < 0$). The initial abstract state is $s_0 : \{s_0 = zero, s_1 = pos, s_2 = neg\}$.

```
fof(checkFeasibility, conjecture,
  (![T,W] :
    (~(T=0 & W>0 & 48.1144*W^2 - 96.2288*cos(T) + 1.4101<0)))).
```

Figure 5.12: Feasibility checking

To determine abstract transitions, QUANTUM will symbolically calculate the Lie derivative of each f_i in turn and generate and send the problems to MetiTarski. In Figure 5.13, the equation to the right of the implication is the Lie derivative of $f_i = W$.

```
fof(checkTransition, conjecture,
  (![T,W] :
    (T=0 & W>0 & 48.1144*W^2 - 96.2288*cos(T) + 1.4101<0
     => (-sin(T) >= 0)))).
```

Figure 5.13: MetiTarski input for checking abstract transitions

In this case MetiTarski proves the conjecture in Figure 5.13, indicating that $s_0 = pos$ is the only possible value of the abstract variable in the next state. The full abstraction is shown in Figure 5.11b where $\theta = T$, $\omega = W$ and LF is the chosen Lyapunov function. As expected, the ideal pendulum displays the same qualitative behaviour as the ideal spring (see Section 5.1.2.1).

5.2.2 Abstracting the Discrete State Space

Now let us consider the discrete transitions between two discrete states q and q' of a hybrid system that are enabled by a guard $G(q, q') = G_{qq'}$, that jumps to location $\dot{x} \in \mathbb{R}^n$ determined by the reset map $R((q, q'), G_{qq'}) = R_{qq'}$.

Definition 5.12. We add to the abstraction a transition from s to s' according to the following rules, if and only if for all $i = 1..k$

- If $\gamma(s) \Rightarrow f_i(R_{qq'}) > 0$ then $s'_i = pos$
- Else If $\gamma(s) \Rightarrow f_i(R_{qq'}) \geq 0$ then $s'_i = \{pos, zero\}$
- Else If $\gamma(s) \Rightarrow f_i(R_{qq'}) < 0$ then $s'_i = neg$
- Else If $\gamma(s) \Rightarrow f_i(R_{qq'}) \leq 0$ then $s'_i = \{neg, zero\}$
- Else If $\gamma(s) \Rightarrow f_i(R_{qq'}) \neq 0$ then $s'_i = \{neg, pos\}$

- Else If $\gamma(s) \Rightarrow f_i(R_{qq'}) = 0$ then $s_i' = zero$
- Else $s_i' = \{pos, neg, zero\}$

Example 5.7 (Abstracting the Bouncing Ball)**.** Consider the bouncing ball on a sine curve hybrid system from Section 2.3.1 on page 32. The system of differential equations is

$$\begin{aligned}\dot{p}_x &= v_x \\ \dot{p}_y &= v_y \\ \dot{v}_x &= 0 \\ \dot{v}_y &= -9.8 + 0.01v_y^2\end{aligned}$$

with a guard $\sin p_x - p_y = 0$, that represents the ball hitting the curve, with update (reset) function

$$\begin{aligned}v_x &:= \frac{(1 - 0.8\cos^2 p_x)v_x + 1.8v_y \cos p_x}{1 + \cos^2 p_x} \\ v_y &:= \frac{1.8v_x \cos p_x + (-0.8 + \cos^2 p_x)v_y}{1 + \cos p_x}\end{aligned}$$

The set of abstraction functions F is chosen to include the continuous variables and the guards.

$$F = \{v_x, v_y, p_y - \sin p_x, p_y, p_x\}$$

If we drop the ball from rest within the trough, the first bounce $\sin px = py$ will occur with $v_y < 0$, $v_x = 0$, $p_x > 0$, $p_y < 0$. Figure 5.14 is the MetiTarski code that determines the qualitative value of v_y after the discrete transition. It is proved, indicating that the next qualitative state of v_y is either pos (bouncing back) or $zero$ (at rest). This process continues for each s_i in all abstract states s that satisfy the guard G.

```
fof(checkTransition, conjecture,
  (![VY,VX,PX,PY] :(VY<0 & VX=0 & -PY + sin(PX)=0 & PX>0 & PY<0
  => ((1.8*VX*cos(PX) + VY*(cos(PX)^2 - 0.8))/(cos(PX)^2 + 1) >= 0))
  )).
```

Figure 5.14: Checking discrete transitions for the bouncing ball

5.2.3 Analysis of the HybridSAL algorithm

The exposition in this section follows closely the definitions, theorem and proof of the qualitative abstraction algorithm presented by Tiwari [209, 211]. Certain notational changes have been made to be consistent with the more general hybrid automaton defined in Section 2.3.1.

Definition 5.13. Given a hybrid automaton $H = (Q, X, f, Init, Inv, E, G, R)$ and a mapping $\alpha : Q \times X \mapsto S$, the discrete time transition system representing all observable states of H is $H_\alpha = (Q \cup X, Init, t_\alpha)$.

The two types of transitions are represented in the following ways. Discrete transitions operate between the current discrete state q and the next state q'. Continuous transitions describe the effect of continuous evolution between x and x' in a discrete state q over a finite non-zero time interval. There is a *discrete* transition $((q, x), (q', x')) \in t_\alpha$ if

$$(q, q') \in E \ \wedge \ G((q, q'), x) \ \wedge \ R((q, q'), x) = x' \tag{5.14}$$

There is a *continuous* transition $((q, x), (q, x')) \in t_\alpha$ if there exists a $\delta > 0$ and a continuous function $\xi : [0, \delta] \mapsto X$ such that for all $\tau \in (0, \delta)$

$$\xi(\tau) \in Inv(q), \tag{5.15}$$

$$\dot{\xi}(\tau) = f(q, \xi(\tau)) \tag{5.16}$$

and

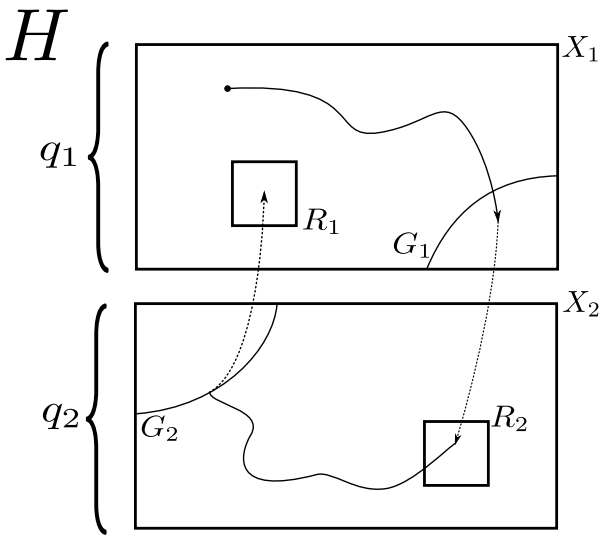
$$\alpha((q, \xi(\tau))) \text{ is a constant function on } [0, \delta) \text{ or } (0, \delta]. \tag{5.17}$$

Equation (5.14) ensures that an edge exists between the two discrete states, that the guard evaluates to True and that the variable x is assigned the right value. Equations (5.15) and (5.16) ensure that the solution or continuous trajectory $\xi(\tau)$ is well defined for the particular mode q. Equation (5.17) ensures that the abstraction function α tracks the observable qualitative behaviour of the trajectories of the system. The qualitative behaviour can change at either the beginning ($\tau = 0$) or end ($\tau = \delta$) of the trajectory. An example hybrid system H and its abstraction H_α are shown in Figure 5.15. The ovals in Figure 5.15b represent some constant valued function $\alpha(x)$.

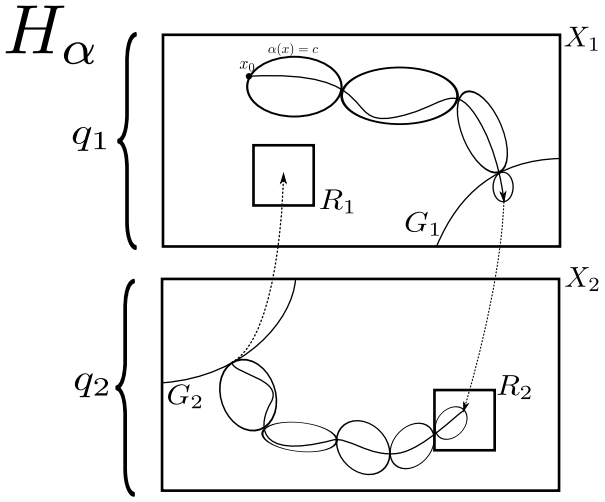
Definition 5.14 (Tiwari [209])**.** Let $H = (Q, X, f, Init, Inv, E, G, R)$ be a hybrid automaton and $DS = (Q', Init', t')$ be a discrete state transition system. DS is an abstraction of H if there exists a mapping $\alpha : Q \times X \mapsto Q'$ such that

1. If $(q, x) \in Init$, then $\alpha(q, x) \in Init'$.
2. If $((q, x), (q', x')) \in t_\alpha$ is a discrete time transition of the system H_α, then there exists a transition $(\alpha(q, x), \alpha(a', x') \in t')$ in DS.

Definition 5.14 makes it clear how the system DS simulates (in the sense defined in Section 5.1.4.2) the discrete state transition system H_α. For every initial state and possible transition of H_α, DS can match it.



(**a**) Hybrid system



(**b**) Abstraction with respect to α

Figure 5.15: Abstraction process

Theorem 5.2 (Tiwari [209])**.** The discrete state abstraction of a hybrid automaton obtained by the qualitative sign based analysis using the Lie derivative is sound.[6]

Theorem 5.2 guarantees that any ACTL* property proved on an abstraction created by the HybridSAL algorithm holds on the original hybrid automaton. The abstraction process provides a guaranteed over-approximation of the reachable states of the system. The proof of this theorem relies on showing that the abstraction DS *simulates* the discrete state system HS_α that captures all the behaviour of the underlying hybrid automaton.

Lemma 5.1 (Tiwari [209])**.** Let $H = (Q, X, f, Init, Inv, E, G, R)$ be a hybrid automaton and F be a finite set of discretising functions over X. If $DS = (Q \cup F_i, Init^A, t^A)$ is the discrete transition system produced by the methods described in Sections 5.2.1 and Sections 5.2.2, then DS is an abstraction for H.

[6]With respect to ACTL* properties.

Proof. If we require that each initial state of the hybrid system be represented by a conjunction of sign conditions on the set of functions $f_i \in F$, then we trivially meet Condition 1 of Definition 5.14. For example, if the initial state of a system with two continuous variables (x, y) has initial conditions that can be described by $\{x > 1, y < -1\}$, then the functions $\{x - 1, y + 1\}$ would be included in F.

Two cases must be considered to show that Condition 2 of Definition 5.14 holds.

1. Continuous transitions ($q' = q$)

   Recall that function ξ represents the continuous trajectory of the system in mode q and that $\alpha(q, (\xi(\tau)))$ is a constant function either on $[0, \delta)$ or $(0, \delta]$. Let $\alpha(x)$, take its meaning from Equation (5.11).

   There are two sub-cases depending on if the qualitative value of a function f_i is constant or changes.

   (a) $s_i \in \alpha(x) = s'_i \in \alpha(x')$.

   This would be the case that the qualitative continuous behaviour of the system does not change (i.e. when the trajectories reach an equilibrium point). We need to show that for all $s_i \in \alpha(x)$, the value of $s'_i \in \alpha(x')$ can remain constant. Consider a $f_i(x) \in \gamma(s_i)$ such that $\forall x \in (q, x) : f_i(x) = 0$, we then have by definition $f_i(\xi(\tau)) = 0$ for all points $\tau \in [0, \delta]$. It then follows that $L_v f_i = 0$, since f_i is constant along all points of the trajectory $\xi(\tau)$. In this case, $s_i = zero$ and therefore we use the rules outlined in Condition 3 of Definition 5.11. We would not be able to prove conjectures 3a or 3b of Definition 5.11. Both other options 3c and 3d of Definition 5.11 can properly choose s'_i to remain $zero$.

   Consider a $f_i(x) \in \gamma(s_i)$ such that $\forall x \in (q, x) : f_i(x) > 0$. This is the case $s_i = pos$. Both options in Condition 1 of Definition 5.11 allow for s'_i to remain pos. The case is similar for $f_i(x) \in \gamma(s_i)$ that $\forall x \in (q, x) : f_i(x) < 0$.

   (b) $s_i \in \alpha(x) \neq s'_i \in \alpha(x')$

   This is the case that the qualitative continuous behaviour of a variable does change. We need to show that for $s_i \in \alpha(x)$ that $s'_i \in \alpha(x')$ can be assigned the correct value.

   Consider a $f_i(x) \in \gamma(s_i)$ such that $\forall x \in (q, x) : f_i(x) > 0$. Since we know that the qualitative value changes, let us assume that $\alpha(q, \xi(\tau))$ is constant on $[0, \delta)$ and that the system has a continuous transition at $\tau = \delta$. The only possible qualitative change is to $f_i(x) = 0$. Therefore at some point along the trajectory of the system $\xi(\tau)$, $L_v f_i(x) < 0$. Since $s_i = pos$, we would consider cases in Condition 1 of Definition 5.11. We would not be able to prove $L_v f_i \geq 0$. As required, option 1b of Definition 5.11 allows $s'_i = zero$. This reasoning is similar for all other cases of $f_i(x) \in \gamma(s_i)$ that change.

It is important to note the following: To ensure the soundness of transitions from a state $s_i = zero$, we must consider the side condition $\nabla f_i \neq 0$ in 3c of Definition 5.11, which represents a vanishing gradient along some f_i. For any $f_i : \nabla f_i = 0$, we have immediately that $L_v f_i = 0$. In any abstraction that uses such an f_i, if we only consider the condition $\gamma(s) \Rightarrow L_v f_i = 0$, then s_i' would be forced to $zero$ even if the underlying trajectories $\xi(\tau)$ moved the system to $f_i > 0$ or $f_i < 0$. We can take care of this degenerate case by symbolically checking the extra condition during the abstraction process or eliminating such functions $f_i \in F$ before the process begins.

Therefore, if $((q, x), (q, x')) \in t_\alpha$ then $(\alpha(q, x), \alpha(q, x')) \in t^A$.

2. Discrete Transition $(q' \neq q)$:

   As in the continuous case, we separate into two sub-cases:

   (a) $s_i \in \alpha(x) = s_i \in \alpha(x')$:

   By Definition 5.15b, if $((q, x), (q', x')) \in t_\alpha$ then $G((q, q'), x)$ for H. If we ensure that all guard conditions are representable in terms of some $f_i \in F$, then any discrete transition enabled by H will have a corresponding correct representation in DS.

   (b) $s_i \in \alpha(x) \neq s_i \in \alpha(x')$:

   In this case, we are dealing with a non-empty $R(x)$, which allows the continuous variables to be updated during a discrete mode change. By the Conditions given in Definition 5.12, we see that for any $f_i(x) \in \gamma(s_i)$, $f_i(x) \in \gamma(s_i')$ is assigned to the proper value depending on the qualitative value of the assignment $f_i(R(x))$.

   Therefore, if $((q, x), (q', x')) \in t_\alpha$ then $(\alpha(q, x), \alpha(q, x')) \in t^A$ and the simulation relation holds.

□

5.3 Initial Evaluation

A sound qualitative abstraction method for hybrid systems has been introduced. During the abstraction process, many first order formulas over the theory of the reals are generated that must be solved to determine whether a particular abstract state is feasible and to determine all possible transitions from it. The QUANTUM system uses MetiTarski to discharge these proof obligations. In this section, I present a series of experiments that were used to evaluate MetiTarski's ability to handle the types of conjectures that are encountered.

The first version of QUANTUM followed directly the implementation of the qualitative algorithm in HybridSAL. Given a system of ODEs, a set of functions F, a safety property p and an initial state i. The following steps are taken:

1. A system S is generated containing all possible abstract states in terms of each $f_i \in F$.
2. For each state $s \in S$, MetiTarski is used to check its feasibility. State s is removed from S if it is not feasible.
3. For each feasible state s of S, MetiTarski is used to determine all feasible next states reachable from s.
4. i and p are converted into a representation in terms of the abstract functions $f_i \in F$.
5. A discrete state system with a temporal logic property is output in the SMV format.
6. The model checker NuSMV is used to analyse the SMV file.

This process generates a discrete transition system that encompasses all possible behaviours of the concrete system. By the soundness of the abstraction process, any safety property proved by the model checker is guaranteed to hold on the concrete system. One primary issue, which is problematic for all qualitative frameworks, is that of the existence of *spurious behaviours*. These are particular behaviours that exist in the abstraction but do not represent any real behaviour in the original system. To reduce the amount of spurious behaviours requires carefully choosing the functions used to discretise the state space. Furthermore, MetiTarski must also be able to prove conjectures involving them for the abstraction to be useful for verification.

5.3.1 Picking the Functions in F

Let us consider proving the following simple property about the pendulum with friction (Example 5.4). If we release the pendulum from rest with $\theta = 1$ and $\omega = 0$, then it should be that θ will never increase above 2. The initial set of functions to include in F are chosen from

- The state variables of the system: θ, ω
- Invariants defined on the state space: $\theta - \pi$, $\theta + \pi$
- The initial state: $\theta - 1$
- Conditions referred to by the safety property: $\theta - 2$

The next set of functions that can be included are generated by repeatedly taking the Lie derivative of the system of differential equations. These functions will generally get more complicated as the number of derivatives taken increases. They represent physical limits on the continuous trajectories such as the acceleration, the jerk (derivative of acceleration), the jounce (derivative of the jerk) etc. Starting with the original system, $\dot{\theta} = \omega$, $\dot{\omega} = -\sin\theta - \omega$, several higher order derivatives that can be added to F are

- $-\sin\theta - \omega$
- $\omega + \omega\cos\theta + \sin\theta$
- $\omega(-1 + 2\cos\theta) + (-1 + \omega^2 + \cos\theta)\sin\theta$
- $\cos\theta(-3\omega + \omega^3 - 2\sin\theta) + \sin\theta + 2\omega(\cos 2\theta - 2\omega\sin\theta)$
- $4\omega + \sin\theta + \cos\theta(4\omega - 7\omega^3 + (3 - 11\omega^2)\sin\theta) - \omega(8\cos 2\theta + \omega(-11 + \omega^2)\sin\theta) - \sin 3\theta$

The number of abstract states and transitions can be potentially limited by adding zero level sets of Lyapunov functions and barrier certificates of the underlying system to F. This will reduce the time required to construct and verify the abstraction and original dynamical system. Since the vector field has nonpolynomial terms, the sum of squares methods (SOSTOOLS) presented in Section 5.1.5 must be used. Since SOSTOOLS use numerical optimisation methods, the generated functions might not truly satisfy the necessary positive or negative definiteness criteria. Since the abstraction methods employed by QUANTUM are fault tolerant,[7] a candidate Lyapunov function or barrier certificate that does not actually meet the requirements will only increase the size of the abstraction rather than cause unsoundness. The level sets that can be added to F have the following form

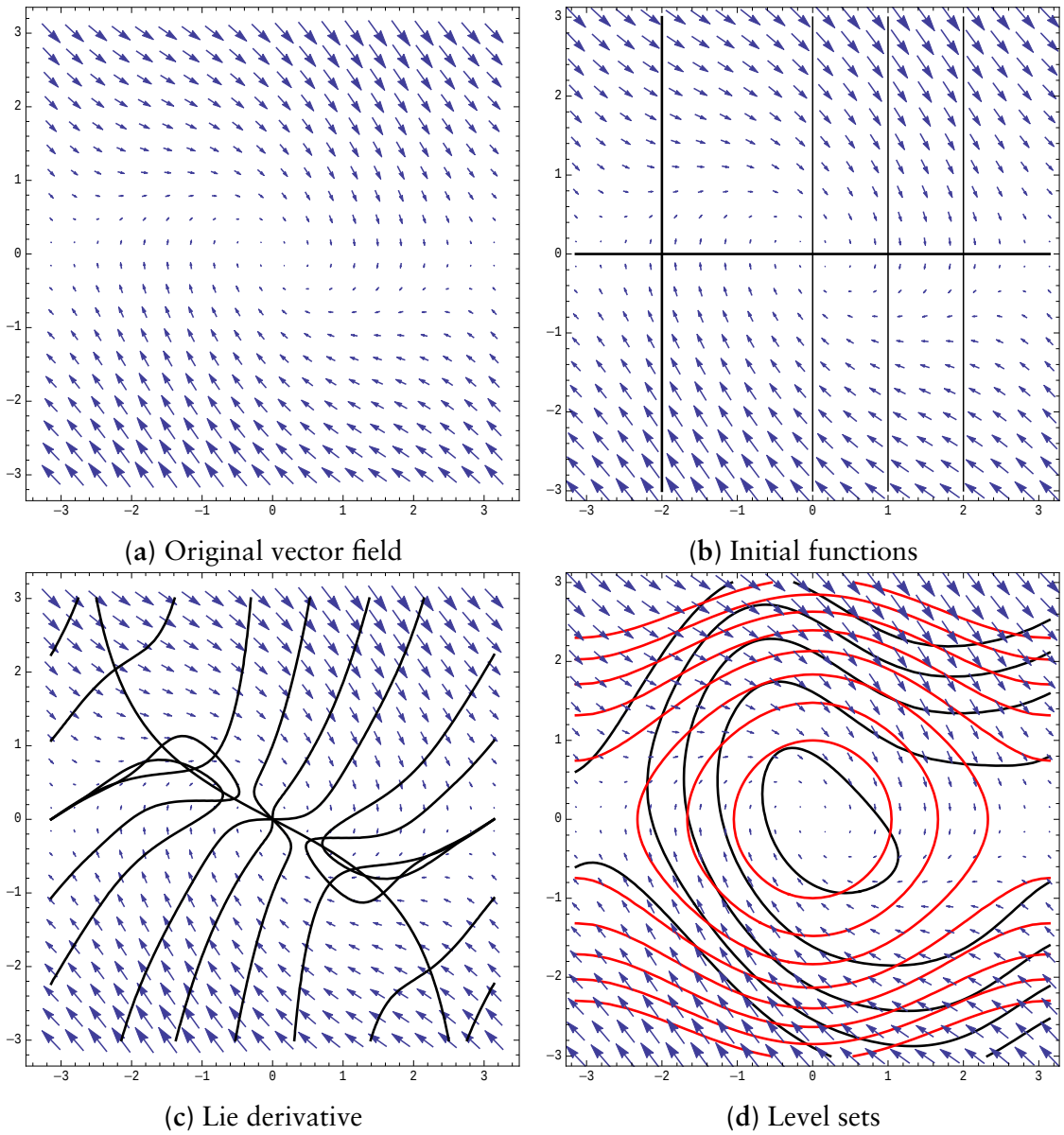
- $-10.09 - 3.83\theta + 4.17\omega^2 - 10.39\cos\theta + 0.17\theta\cos\theta - 7.39\cos^2\theta - 0.17\sin\theta + 4.31\omega\sin\theta + 3.83\cos\theta\sin\theta - 6.50\sin^2\theta$
- $4.2214 + 4.2214\omega^2 - 8.4428\cos\theta$

Figure 5.16 shows the different types of functions that can be used to discretise the state space of the pendulum system.

5.3.2 Experimental Results

Several experiments were conducted using the pendulum example to evaluate how well MetiTarski could handle each type of discretising function. These experiments focus on purely continuous dynamical systems. Fully hybrid systems will be analysed in Chapter 6.

[7] If a conjecture cannot be proved, all possible transitions from an abstract state are considered.



(a) Original vector field

(b) Initial functions

(c) Lie derivative

(d) Level sets

Figure 5.16: Different types of functions in F for the pendulum with friction

The first experiment investigated the use of higher order Lie derivatives. Starting with the base set of functions derived from the system of differential equations and the property to be verified, successively finer abstractions were created by adding in each subsequent trial the next Lie derivative to the set F. The results are shown in Table 5.4. $|F|$ is the number of functions in the set F. *Timeout* is the MetiTarski global timeout. The two numbers in the *Feasibility* column list respectively the number of abstract states that have been proved and not proved infeasible. The two numbers in the *Transitions* column list respectively the number of conjectures regarding transitions that have been proved and unproved. *Abs. Time* is the total amount of time spent constructing the abstraction. All listed times are in seconds.

Trial	$\|F\|$	Timeout	Feasibility	Transitions	Abs. Time
		0.01	15/66	92/208	18
#1	4	0.1	105/24	38/58	17
		1	105/24	38/58	83
		0.01	69/174	260/842	60
#2	5	0.1	102/43	64/176	41
		1	104/36	52/152	187
		0.01	215/514	988/2760	213
#3	6	0.1	325/125	233/587	541
		1	320/81	182/537	633
		0.01	653/1534	3636/9102	785
#4	7	0.1	991/359	837/1926	461
		1	256/238	715/1798	2096
		0.01	1860/4642	13603/31952	3331
#5	8	0.1	2856/1194	3580/6607	1759
		1	3029/929	3280/7114	8460

Table 5.4: Experiment 1 - Lie derivatives

There are several important things to take note of in Table 5.4. If the MetiTarski timeout is set too low, this can cause a much higher number of states to be included in the abstraction (this is equal to the number of unproved feasibility conjectures). Across all 5 trials, the average reduction in the size of the abstract state space by increasing the timeout from 0.01 seconds to 0.1 seconds is a factor of 4. In all trials except #3, this caused a noticeable decrease in the time required to construct the abstraction. Unfortunately, none of the trials resulted in an abstraction that allowed the model checker to prove the safety property. To address this, Experiment 2 investigated how the inclusion of zero level sets of nonpolynomial Lyapunov functions or barrier certificates affected the size of the abstraction and the provability of the safety property.

The results of including one and two zero level sets of a Lyapunov function are shown in Table 5.5. Each trial dropped the use of a timeout of 0.01 seconds due to the results of Experiment 1 that showed the resulting abstractions were too coarse. The entire Trial #5 was also dropped as the abstraction time was over 5 hours. The a sub-trials included 1 zero level set and the b sub-trials included 2 zero level sets.

Each trial of Experiment 2 exhibited the same general behaviour as that of the corresponding trial in Experiment 1, increasing the size of F leads to a significant increase in time required to create the abstraction. The benefit of including level sets for reducing the size of the state space is immediately clear. For instance, in Trial #1a, the size of the abstraction has been reduced by 25% and in Trial #4a it has been reduced by 75%.

Trial	$\|F\|$	Timeout	Feasibility	Transitions	Abs. Time
		0.1	18/18	46/44	14
#1a	5	1	18/18	46/44	52
		10	18/18	46/44	403
		0.1	18/18	64/46	15
#1b	6	1	18/18	64/44	57
		10	18/18	64/44	548
		0.1	76/25	61/99	28
#2a	6	1	63/20	50/74	93
		10	63/20	50/74	1039
		0.1	77/29	102/117	34
#2b	7	1	63/20	70/74	94
		10	63/20	70/74	1089
		0.1	241/83	392/237	103
#3a	7	1	223/42	153/304	366
		10	172/35	129/214	3382
		0.1	245/84	377/441	126
#3b	8	1	219/42	227/291	368
		10	172/35	175/215	3472
		0.1	257/230	1786/2488	75
#4a	8	1	632/94	752/1302	1496
		10	172/32	129/214	13382
		0.1	764/235	1919/2106	664
#4b	9	1	628/104	901/1132	1417
		10	583/79	706/899	13672

Table 5.5: Experiment 2 - Level sets and Lie derivatives

For simpler abstractions with a small set F, such as in trials #1a, #1b, #2a and #2b, increasing the timeout of MetiTarski does not have any effect on the size of the abstraction. It only serves to increase the total time necessary for its construction. However, as the complexity of the Lie derivatives increases, the timeout allowed to MetiTarski starts to have an effect as shown in trials #3a, #3b, #4a and #4b.

In each of the trials (#1a through #4b), the safety property $G\neg(\theta > 2)$ was proved by NuSMV. As the abstract transition systems are small (the largest in Trial #4b has only 235 discrete states), the average time required by the model checker to verify this simple safety property was under 1 second. This demonstrates that the majority of the effort is in the construction of the abstraction and not in verification. Even though the required safety property was proved, the abstract models were still found to contain many spurious behaviours.

Experiment 3 looked at this issue in more depth by allowing MetiTarski to focus only on the transition conjectures of Trial #2a. The results are shown in Table 5.5, where the *Extra Problems Proved* column indicates how many extra conjectures were proved by increasing the MetiTarski timeout to that indicated in the first column.

MetiTarski Timeout	Extra Problems Proved	Total Proof Time
20	9	9m7s
60	6	11m43s
120	0	44m4s
240	0	1h43m16s
600	4	2h55m3s

Table 5.6: Experiment 3 - Extending MetiTarski timeout

Experiment 3 shows that MetiTarski must work significantly longer on only a few specific conjectures to eliminate spurious behaviours from an abstraction. In the initial implementation of QUANTUM, increasing MetiTarski's global timeout would also lengthen the time spent on both provable and unprovable conjectures. As experiments 1 and 2 demonstrate, the increase in total abstraction time renders the process essentially unusable. The next section investigates improvements to QUANTUM that address this specific issue.

5.4 Lazy Improvements to QUANTUM

Experiment 1 clearly shows one of the weaknesses of the original implementation of QUANTUM. Each trial generates an abstraction that violates the safety property of interest, but this negative result is obtained only after the entire abstraction process has finished. Any time spent on a conjecture that cannot be proved is entirely wasted. Figure 5.17 shows the amount of time that QUANTUM wastes with respect to the total amount of time that is used to create the abstraction. The red bars represent the total proof time and the blue bars represent the amount of time used by MetiTarski that resulted in either a timeout or gave up result. I consider this time wasted as those results cannot be used to refine the abstraction in any way.

It could be that the conjecture is false or that MetiTarski simply requires more time to prove the conjecture. In either case, it is clear that constructing the entire abstraction explicitly, followed by verification using model checking, is wholly inefficient.

Let us assume that we are dealing with safety properties that can be defined in terms of the functions that are used to discretise the state space. In this case, it may be possible to determine early on in the abstraction process whether it is possible to reach an unsafe

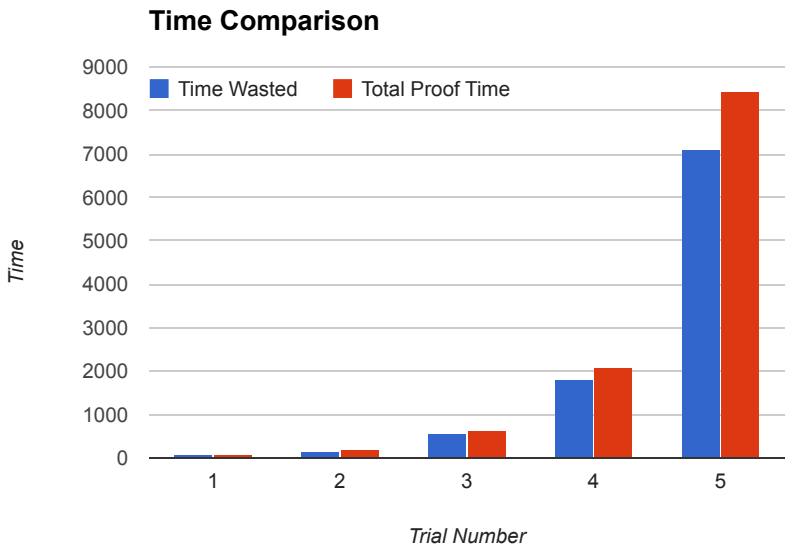


Figure 5.17: Wasted time for experiment 1

state. Once it is detected that there is a transition into a bad state, the process can terminate early, potentially saving on the abstraction time. This *lazy qualitative abstraction* performs the following steps:

1. Check that all initial states are feasible.
2. From each state, generate all possible next states.
3. Check that each reachable state is feasible.
4. For each feasible reachable state check if the safety property is violated, terminate if it is.
5. If there is no feasible next state, terminate, otherwise go to step 3.

Experiment 4 tested this lazy qualitative abstraction methodology on the trials of Experiment 1 (with a timeout of 1 second). The results are shown in Table 5.7. The *Time Saved* column indicates the percentage of the original abstraction time saved.

The results show that constructing an abstraction lazily can greatly reduce the amount of time wasted by QUANTUM. By keeping track of the safety property, the abstraction loop can terminate immediately when it has been invalidated. The verification of simple safety properties of the form $G\neg(state)$ does not require the use of a model checker. The lazy framework is very good at reducing the time wasted for disproving a safety property. However, of much more interest is creating an abstraction that can actually prove properties. In the following sections, we will see how the lazy framework can be used to reduce the number of spurious behaviours in an abstraction.

Trial	Original Abs. Time	Lazy Abs. Time	Time Saved
#1	83	60	23%
#2	187	97	48%
#3	633	215	66%
#4	2096	618	70%
#5	8460	320	96%

Table 5.7: Experiment 4 - Lazy qualitative abstraction

5.4.1 CEGAR Loop Implementation

Chapter 3 discussed the ability of MetiTarski to solve difficult continuous system verification problems. Through several experiments I developed useful techniques for reducing the time required to prove difficult conjectures. The techniques included varying the amount of time allowed to the underlying RCF solver, a series of recasting heuristics and splitting the problem into many individual subproblems. We will now see how we can use these techniques in QUANTUM.

The original implementation of QUANTUM did not work well with the MetiTarski proof techniques as the same parameters had to be used on every single generated conjecture. The previous experiments show that even with a short global timeout of 10 seconds, the total time required to construct an abstraction is much too long to be of any use. With the lazy implementation, the abstraction process can focus on specific difficult abstract states and transitions that would invalidate the safety property. The lazy abstraction loop can be enhanced by adding a simple type of *counter example guided abstraction refinement* (CEGAR) loop (see Section 5.1.4.1), to allow a much finer abstraction to be created.

If in step 4 of the lazy abstraction loop, the system transitions into an unsafe state:

1. Re-check the feasibility of the specific transition to the unsafe state, using one of several strategies,
   (a) Apply a higher global timeout.
   (b) Apply a lower RCF timeout.
   (c) Recast if possible any sine and cosine terms.
2. If the bad transition still exists, analyse the feasibility of the originating state itself, applying the same set of strategies from step 1.
3. Repeat the last two steps a set number of times.
4. If the unsafe transition still exists, abort.

The CEGAR loop allows the initial MetiTarski timeout to be kept low, ensuring a minimum amount of time is wasted on unprovable conjectures. Only when it is absolutely necessary is the timeout allowed to increase. This guarantees that effort is only put on trying to prove conjectures that disprove the candidate safety property. The CEGAR loop could also introduce additional discretising functions into F to further refine the particular invalidating state. Although the appropriate machinery is in place, this addition has been left as future work.

5.4.2 Multiprocessing

Another improvement to QUANTUM was motivated by the realisation that many of its computations could be run in parallel. This is because the determination of all abstract transitions from an abstract state $s \in S$ can be broken down into three distinct calls to MetiTarski. Take for example Algorithm 1, which is the implementation of Definition 5.11. There are three generated calls to MetiTarski in lines 1, 2 and 3 that can be worked on at the same time. Using similar reasoning, the feasibility checking of a series of newly generated next states can also be sent to MetiTarski in parallel as the conjectures are independent of one another. This parallel processing extension was implemented using the Python multiprocessing framework.

5.4.3 Evaluation

Experiment 5 reran several trials from Experiment 2 on the improved version of QUANTUM that implemented lazy abstraction, parallel processing and the CEGAR loop. The results are shown in Table 5.8. There was an average speed up of 2.76 across all trials, even as the timeout to MetiTarski was increased. These results demonstrate that the improved version not only constructs the abstraction faster than the original abstraction algorithm, but also proves the required verification condition.

5.5 Chapter Summary

This chapter has been concerned with a technique called *verification by abstraction*. In this methodology, if a certain model is too complex to be verified, it is abstracted into a simpler form that is more amenable to the methods that are available. The critical requirement is that the abstraction process must preserve all the properties of interest of the original system. The ideal situation is that the model is simplified just enough to allow the verification to succeed.

Algorithm 1: Determining next value of s_i within $q_i \in Q$ of a hybrid system

Input: System of Ordinary Differential Equations, O
Input: Abstract state component, $s_i \in s$
Output: List of possible values of s_i'

$Q_1, Q_2, Q_3 = \emptyset, \emptyset, \emptyset$
$L_o f_i =$ Lie derivative of $\gamma(s_i)$ with respect to O

if *MetiTarski proves* $\gamma(s) \Rightarrow L_0 f_i \geq 0$ **then**
  Q_1 += s
if *MetiTarski proves* $\gamma(s) \Rightarrow L_0 f_i \leq 0$ **then**
  Q_3 += s
if *MetiTarski proves* $\gamma(s) \Rightarrow L_0 f_i < 0 \vee$ *MetiTarski proves* $\gamma(s_i) \Rightarrow L_0 f_i > 0$ **then**
  Q_2 += s

if $s_i = pos$ **then**
  if $s \in Q_1$ **then**
    $s_i' = pos$
  else
    $s_i' = \{pos, zero\}$
else if $s_i = neg$ **then**
  if $s \in Q_3$ **then**
    $s_i' = neg$
  else
    $s_i' = \{neg, zero\}$
else
  if $s \in Q_1 \wedge s \in Q_2$ **then**
    $s_i' = pos$
  else if $s \in Q_3 \wedge s \in Q_2$ **then**
    $s_i' = neg$
  else if $s \in Q_1 \wedge s \in Q_3$ **then**
    $s_i' = zero$
  else
    $s_i' = \{pos, neg, zero\}$

Trial	$\|F\|$	Timeout	Old Abs. Time	New Abs. Time	Speed Up
		0.1	14	5	2.8
#1a	5	1	52	26	2.0
		10	403	250	1.6
		0.1	15	6	2.5
#1b	6	1	57	28	2.0
		10	548	317	1.7
		0.1	28	11	2.5
#2a	6	1	93	34	2.7
		10	1039	437	2.4
		0.1	34	13	2.6
#2b	7	1	94	33	2.8
		10	1089	489	2.2
		0.1	103	46	2.2
#3a	7	1	366	105	3.5
		10	3382	1457	2.3
		0.1	126	55	2.3
#3b	8	1	368	112	3.2
		10	3472	1428	2.4
		0.1	614	178	3.5
#4a	8	1	1496	482	3.1
		10	13382	4834	2.8
		0.1	664	225	3.0
#4b	9	1	1417	643	2.2
		10	13672	5472	2.5

Table 5.8: Experiment 5 - Improvement evaluation

The verification of safety properties of dynamical systems is inherently difficult. In particular, the verification of reachability properties of hybrid systems is well known to be undecidable. Therefore, abstraction methods have been developed to reduce the verification problem to one that can be solved by automated methods such as model checking.

Tiwari introduced a qualitative abstraction method for hybrid systems with HybridSAL. In his method, continuous functions are used to discretise the continuous state space into sign invariant regions. A theorem prover is used to determine how continuous and discrete trajectories move between the abstract states. The major restriction of his method is that the dynamical systems must have polynomial vector fields.

The QUANTUM system implements an improved qualitative abstraction method for nonpolynomial hybrid systems. The system's vector field, invariants and guards can all contain transcendental and other special functions. The discretising functions too, can

contain arbitrary combinations of transcendental functions. In particular, QUANTUM can use level sets of barrier certificates and Lyapunov functions to reduce the size and complexity of the state space. To construct the abstraction, the automated theorem prover MetiTarski is used to discharge the generated conjectures. The abstraction method implemented in QUANTUM is completely automatic. All that must be provided is the initial seed set of discretising functions.

Two versions of QUANTUM were presented in this chapter. The first explicitly constructed the discrete state space and then applied model checking to verify safety properties. This proved to be inefficient, especially in cases when the number of discretising functions was high and the final model checking result was a counterexample. The main reason for this arises from MetiTarski's inability to reject false conjectures. Given a false conjecture MetiTarski will run until it hits a timeout or until there are no more clauses to process. Therefore, whenever MetiTarski returns a *Timeout* or *Gave Up*, nothing can be concluded about the trajectories of the original system. The resulting abstraction in this case would be larger and potentially contain spurious behaviours.

The second version of QUANTUM constructs the abstract system lazily. When it is detected that the system transitions into an unsafe state, the proof parameters to MetiTarski can be changed, allowing application of the techniques developed in Chapter 3 of this dissertation. The lazy process allows QUANTUM to focus on specific conjectures that are difficult for MetiTarski. In addition, certain computations performed during the abstraction process have been coded to run in parallel. This serves to further reduce the amount of time taken during the construction of the abstraction. This enhanced version of QUANTUM can create abstractions that are smaller in size, more accurate and faster to produce.

CHAPTER 6

Case Studies

Chapter 5 described the development and experimental evaluation of QUANTUM, a qualitative abstractor for nonpolynomial hybrid dynamical systems. Its initial implementation was shown to be inefficient. For a relatively small sized dynamical system, the abstraction process took too much time to be of any practical use. This negative result motivated the development of several techniques to speed up the process. The general strategy that led to a noticeable improvement was to keep the timeout to MetiTarski low (0.1 seconds was found to be an appropriate duration) and to only increase it when absolutely necessary (e.g. when a transition into an unsafe qualitative state is detected). The improvements to QUANTUM have made it possible to successfully verify several benchmarks that have been previously used for the comparison of bounded time hybrid system verification frameworks [78, 117]. This chapter presents these experimental case studies to further demonstrate the capabilities of QUANTUM.

Each section in this chapter describes a separate series of experiments where QUANTUM was applied to verify safety properties of a nonpolynomial hybrid automaton. The first example is a modified bouncing ball hybrid system. Instead of bouncing on a flat surface, this ball bounces on a surface defined by a sinusoidal function. The second example is a model that comes from the chemical engineering domain. Two tanks holding a liquid are connected in series with one placed higher than the other. The flow rate between the two tanks is proportional to the square of the geometries of the connecting pipe. The third example is a model of a self-driving car that moves along a road bounded by a canal on each side. If it nears the edge of the road, it can make a curved turn to avoid falling into the canal. Finally, the last example is a modification of the self-driving car case study where both the road the car is travelling on and the edges of the canal are curved.

If QUANTUM, via MetiTarski, returns that a discrete state abstraction is safe (via many individual proofs), then I consider that the case study has been successful in showcasing its abilities. This is the criteria used for determining whether the QUANTUM system can effectively verify real dynamical systems.

6.1 Bouncing Ball on a Sine Curve

6.1.1 Model Description

The model analysed in this section originates from particle physics experiments that are concerned with identifying and describing the complex behaviour of granular media (such as sand) on a vibrating plate [110, 213]. The simplified hybrid system model, which represents the vibrating plate as a sine wave shaped surface, has been used by Ishii et al. [117] and by Eggers et al. [78] respectively for the simulation and verification of nonlinear hybrid automata. It is an interesting case study primarily because there is a nonpolynomial function in the transition guard of the hybrid automaton, which can cause issues for simulation methods.

The hybrid automaton representing the bouncing ball system is shown in Figure 6.1 below. p_x and p_y are the positions of the ball in the x and y direction. v_x and v_y are the velocities in the x and y direction. Figure 6.2a shows a proper simulation trace for initial conditions $v_x = 1$, $v_y = 0$, $p_x = 0$, $p_y = 2$. Figure 6.2b shows an example of how simulation can produce incorrect behaviour when the transition guard contains a transcendental function. In this case, the simulation outputs the impossible behaviour of the ball passing through the curve. When the ball is in the air, it follows the standard equations for projectile motion under the effect of air resistance. When the ball hits the curve, it undergoes an inelastic collision and new values are assigned to v_x and v_y.

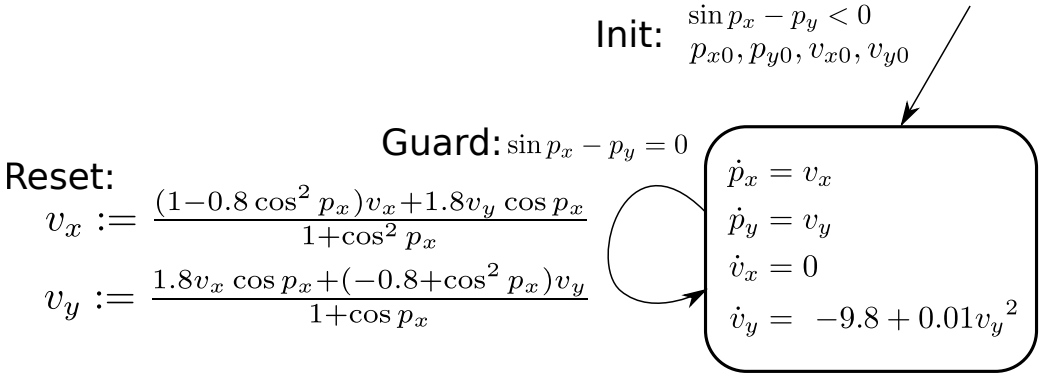
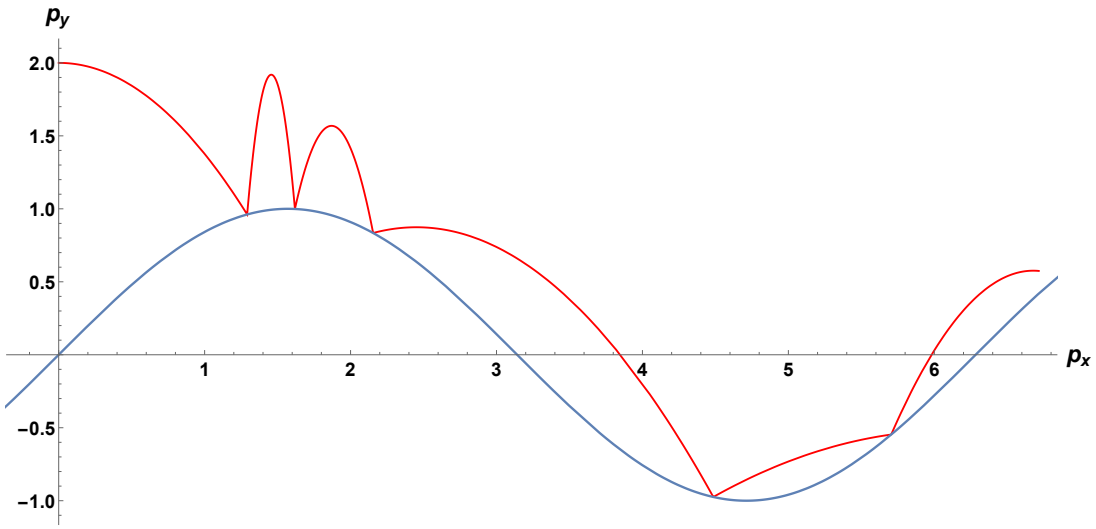
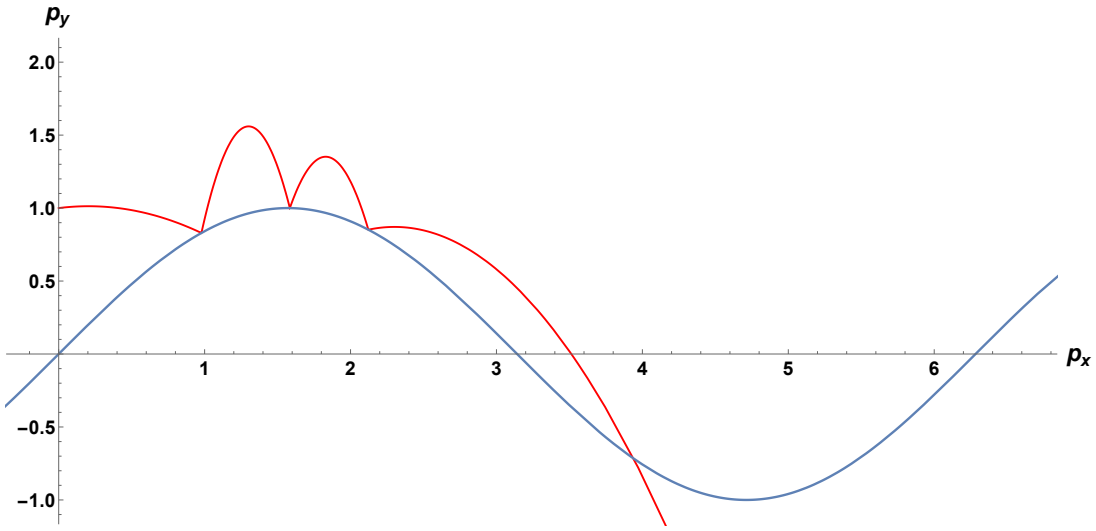


Figure 6.1: Bouncing ball hybrid automaton



(a) Good simulation trace



(b) Bad simulation trace

Figure 6.2: Simulation of the bouncing ball made using Mathematica

6.1.2 Experimental Results

The property of interest to be verified is whether dropping a ball from rest ($v_x = 0$, $v_y = 0$) inside a trough of the sine curve ($p_x > \pi/2$, $p_x < 3\pi/2$, $p_y < 1$) will remain inside the trough for all time. Another way to represent this verification condition is by the LTL safety property $G\neg(p_y > 1)$. That is, p_y should always be less than or equal to 1. The first experiment used the system definition (vector field, invariant and guard) to define the set of discretising functions F.

$$F = \{p_x, p_y, p_y - 1, v_y, v_x, -9.8 + 0.01v_y, \sin p_x - p_y\}$$

For this set F and initial state $p_y - 1 < 0$, $v_y = 0$, $v_x = 0$, QUANTUM constructed an abstract system containing 77 abstract states and proved 296 transitions in 107 seconds before the safety property was violated. The CEGAR loop attempted several higher MetiTarski timeouts on the violating transition, but was unable to prove it to be spurious. This result indicated that the discretising functions in F were inadequate for abstracting the behaviour of the bouncing ball. Based on the results of Section 5.1.5, another attempt to verify the safety property was made by adding a constraint on the energy of the system to F.

The total energy of the system, with a mass of 1 kilogram, is

$$E = 9.8(p_y - \sin p_x) + \frac{1}{2}\sqrt{v_x^2 + v_y^2}$$

and if the ball is released inside the trough, the initial energy of the system is

$$E_0 = 9.8(1 - \sin p_x)$$

The following function was added to F to capture the constraint that the total amount of energy in the system will never be greater than the initial energy put into the system.

$$\frac{1}{2}\sqrt{v_x^2 + v_y^2} + 9.8(p_y - \sin p_x) - 9.8(1 - \sin p_x)$$

Since the property to be verified was concerned only with the height of the ball at $p_y = 1$, the functions p_x and p_y were removed from F as they provided no extra qualitative information. The second experiment used the following functions to discretise the state space.

$$F = \{p_y - 1, v_x, v_y, \sin p_x - p_y, \frac{1}{2}\sqrt{v_x^2 + v_y^2} - 9.8(1 - \sin p_x)\}$$

With this new set F, QUANTUM was able to construct an abstraction that proved the safety property $G\neg(p_y > 1)$. There were 22 infeasible state conjectures proved and 70 transition conjectures proved, resulting in an abstract system of 30 states. The abstraction was constructed in 27 seconds.

Although the safety property was proved, it was also of some interest to investigate if QUANTUM could further reduce the size of the abstract model and remove spurious behaviours by increasing the timeout to MetiTarski. Table 6.1 shows the results of analysing only those conjectures that could not be proved within the default time limit of 0.1 seconds. QUANTUM was able to decide the feasibility of the states quite quickly; increasing the timeout did not greatly affect the number of states in the abstraction. On the other hand, MetiTarski was able to prune away a noticeable number of spurious continuous transitions when given more time to work on the conjectures. There is no change to the number of discrete transition conjectures proved. This result further supported the choice of maintaining a low timeout and only increasing it when required.

Conjecture Type	Timeout	Proved	Increase	Proof Time
Feasibility	0.1	20/57	n/a	3
	1	22/57	10%	19
	10	22/57	0%	220
	100	22/57	0%	572
Continuous Transitions	0.1	132/292	n/a	5
	1	143/292	8%	16
	10	151/292	6%	659
	100	151/292	0%	3343
Discrete Transitions	0.1	104/285	n/a	4
	1	104/285	0%	96
	10	104/285	0%	707
	100	104/285	0%	4732

Table 6.1: Bouncing ball abstraction reduction

The final experiment used the recasting technique described in Section 3.3.5 (see page 53) on the bouncing ball model. Instances of $\sin p_x$ and $\cos p_x$ were replaced with the variables S and C and the constraint $S^2 + C^2 = 1$ was added to all conjectures. QUANTUM could not prove the safety property of the new system with a timeout of 0.1 seconds. It took 28.5 seconds to create the abstraction of 29 states (20 proved infeasible) with 46 transitions conjectures proved feasible.

The failure was determined to be caused by the increase in the number of variables induced by the recasting process. The generated conjectures were made more difficult for the RCF decision procedure, causing more to be left unproved and consequently the abstraction retained more safety violating spurious behaviours. This clearly showcases one weakness of the recasting technique when applied to the generation of a qualitative abstraction.

The CEGAR strategy was used successfully to eliminate the infeasible states that led to the bad transition in the recasted system. The final abstract system had 34 states (22 proved infeasible) with 59 transition conjectures proved. It took 31.7 seconds to create the abstraction, with the CEGAR loop being called 4 times to increase the timeout from 0.1 seconds to 1 seconds. The strategy taken by the CEGAR loop was to increase the timeout by a factor of 10 on each iteration.

Table 6.2 shows the results of increasing the timeout to MetiTarski on only those conjectures of the recasted system that could not be proved within the default timeout of 0.1 seconds. Again, MetiTarski decided the feasibility of the abstract states quickly. Increasing the timeout had little effect on the number of final abstract states. The most important difference between the original and recasted system was seen in the number of discrete transition conjectures proved. Although the higher number of variables made

the problems more difficult for the RCF decision procedure Z3, a timeout of 100 seconds allowed several spurious discrete transitions to be removed from the abstraction. The results suggest the global timeout to MetiTarski might have to be increased when using the recasting technique during construction of a qualitative abstraction.

Conjecture Type	Timeout	Proved	Increase	Proof Time
Feasibility	0.1	28/78	n/a	2
	1	43/78	20%	14
	10	43/78	0%	153
	100	43/78	0%	1022
Continuous Transitions	0.1	162/365	n/a	6
	1	198/365	10%	62
	10	207/365	12%	668
	100	213/365	14%	4439
Discrete Transitions	0.1	125/315	n/a	5
	1	152/315	9%	61
	10	152/315	9%	509
	100	158/315	10%	3747

Table 6.2: Recasted bouncing ball abstraction reduction

6.2 Two Tank System

6.2.1 Model Description

For this section, we consider a model of two liquid holding tanks that are connected in series by a pipe, with the first placed higher than the second. In a hybrid system context, it was first investigated by Stursberg et al. [201] who used it to demonstrate methods for abstracting hybrid dynamical systems to timed automata. It has since become a standard benchmark problem for hybrid system verification frameworks [78, 107, 117, 177]. It is an interesting case study for QUANTUM because of the square root terms that appear in the definition of the dynamics. Figure 6.3 shows the physical setup of the system.

The first tank is filled at a rate proportional to k_1. The diameter of the connecting pipe, which affects the flow rate between the two tanks, is proportional to k_2. k_3 is the height of Tank 1 above Tank 2. The flow out of Tank 2 is dependent on the diameter of the outlet pipe that is proportional to k_4. x_1 is the height of the water in the first tank, x_2 is the height of the water in the second tank. The hybrid automaton modelling this system is shown in Figure 6.4. The dynamics of the system switch when the height of the water x_2 is above or below k_3.

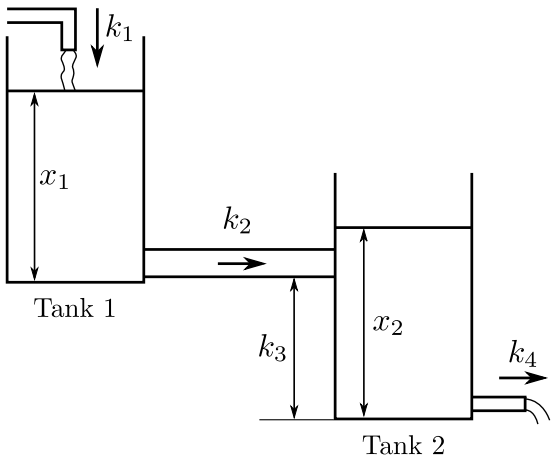


Figure 6.3: Two-tank dynamical system

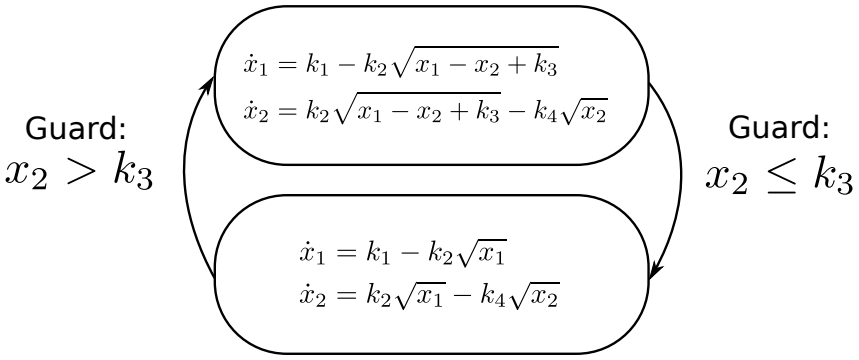


Figure 6.4: Hybrid automaton of the two-tank system

We investigate an instance of the model with all k_i parameters set to 1. For the state of the hybrid automaton with the height of the water in the second tank is below $k_3 = 1$, the invariant $(x_1, x_2) \in [4, 6] \times [0, 1]$ is imposed. For the other state, when the water in the second tank is above $k_3 = 1$, the invariant $(x_1, x_2) \in [4, 6] \times [1, 2]$ is imposed. The verification condition to be checked is whether all trajectories of the system starting in the region $init : (x_1, x_2) \in [5.25, 5.75] \times [0, 0.5]$ avoid the region $unsafe : \{(x_1 - 4.25)^2 + (x_2 - 0.25)^2 \leq 0.0625\}$. The vector field, the regions (*init* in green and *unsafe* in red) and several trajectories are shown in Figure 6.5.

6.2.2 Experimental Results

The first experiment used the definition of the guards and invariants of the system to discretise the state space. The set F was chosen to be

$$F = \{x_1 - 4, x_1 - 6, x_2 - 2, x_2, x_2 - 0.5, x_2 - 1, x_1 - 5.25, x_1 - 5.75$$
$$(x_1 - 4.25)^2 + (x_2 - 0.25)^2 - 0.0625\}$$

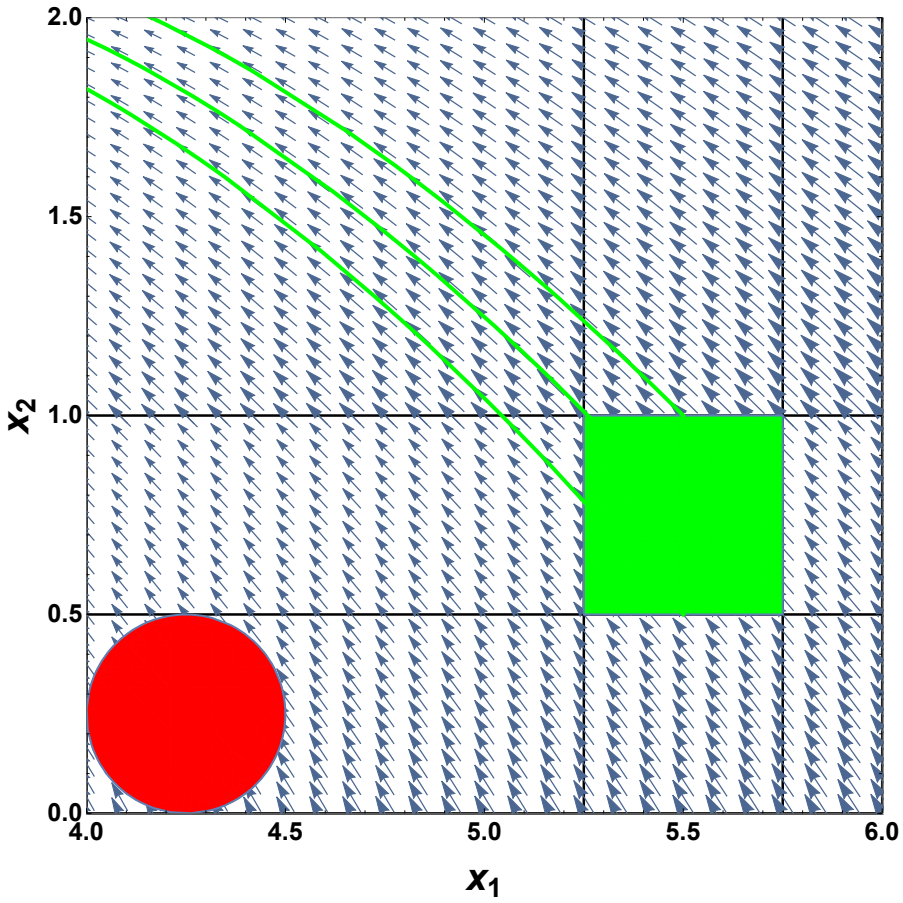


Figure 6.5: Trajectories of the two-tank system

With the default timeout of 0.1 seconds, QUANTUM created an abstraction that violated the safety property. The abstraction had 9 states (34 proved infeasible) with 22 proved transition conjectures. It took 16 seconds to create the abstraction.

The CEGAR loop was able to make some progress on proving the safety property. It was called four times, increasing the MetiTarski timeout to 1 second to successfully remove infeasible states. Unfortunately on the fifth call it got stuck in a state where the violating behaviour could not be eliminated by MetiTarski within 1000 seconds. Investigating this result further, it was discovered that neither the originating state nor the violating transition were incorrect according to the abstraction construction rules outlined in Sections 5.2.1 and 5.2.2. Looking at the abstraction in more detail, it was found that a spurious state several steps before caused a series of valid jumps to occur. This result demonstrates one weakness of the CEGAR loop as implemented in QUANTUM, that it only reexamines potentially spurious transitions at a distance of most 1 transition away.

The next experiment increased the global timeout of MetiTarski to 1 second. QUANTUM was able to construct an abstraction that proved that the state *unsafe* could not be reached from trajectories beginning in *init*. The abstraction contained 14 states (20 proved infeasible) and 103 proved transition conjectures. It took 41 seconds to create the

abstraction. As before, the unproved conjectures were analysed to see if MetiTarski could remove any more spurious behaviour by increasing its timeout. The results are shown in Table 6.3. For this particular example, increasing the timeout did not change the number of abstract states and only proved 3 more transition conjectures. This suggests that in some cases the default timeout of 0.1 second might be too restrictive. The reason that QUANTUM was able to create a highly refined abstraction with only a timeout of 1 second was because the conjectures were relatively easy to prove (only two continuous variables) and the only special function in this particular example was the square root.

Conjecture Type	Timeout	Proved	Increase	Proof Time
Feasibility	1	56/74	n/a	2
	10	56/74	0%	16
	100	56/74	0%	20
Continuous Transitions	1	142/224	n/a	8
	10	145/224	2%	338
	100	145/224	0%	3686

Table 6.3: Two tanks abstraction reduction

6.3 Self-Driving Car

The next example is concerned with verifying the model of a simple self-driving car. It was originally investigated by Clarke et al. [51] for the application of counter example guided abstraction refinement methods to the verification of hybrid systems. A modified version, which eliminates a potential race condition depending on the semantics of the hybrid automaton used, was proposed by Eggers et al. [78].

Consider the environment shown in Figure 6.6. A car drives along a road that is bounded on each side by a shoulder and a canal. When it is detected that the car has reached either side of the road, it switches to a correction mode that initiates a curved turn to avoid the canal. When it is detected that the car is back on the road, another correction mode is entered that re-orients the car to move straight ahead.

6.3.1 Model Description

The car is assumed to be moving at a constant linear velocity v. Its position is p, measured from the center of the road ($p = 0$). Its heading is γ measured from the horizontal axis ($\gamma > 0$ points to the right and $\gamma < 0$ points to the left). The angular velocity during a turn is ω. The time spent in the correction modes is measured by a clock c. It records how

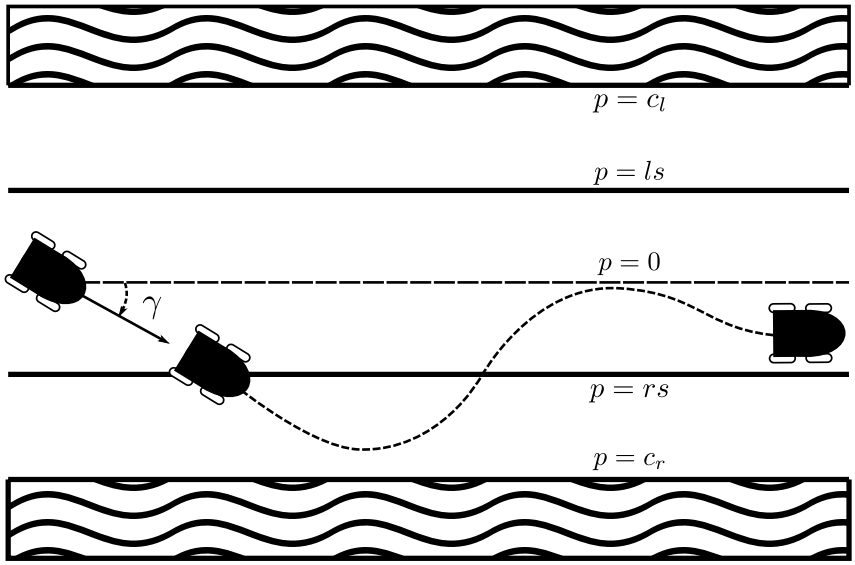


Figure 6.6: Self-driving car

much time is spent turning away from the canal, to properly re-position the car when it is back on the road. The car is modelled by the hybrid automaton in Figure 6.7, where ls and rs are respectively the location of the left and right sides of the road.

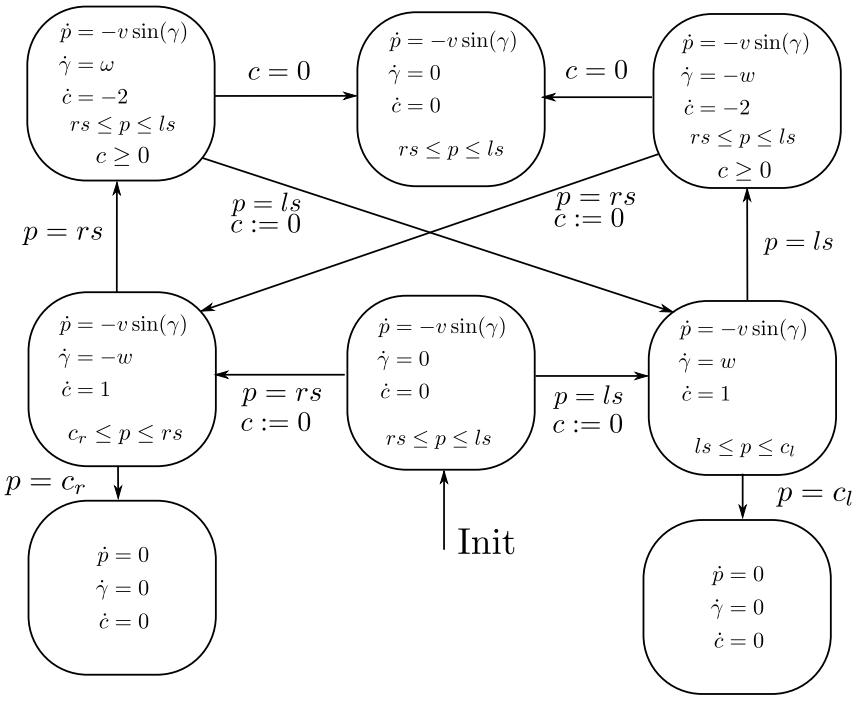
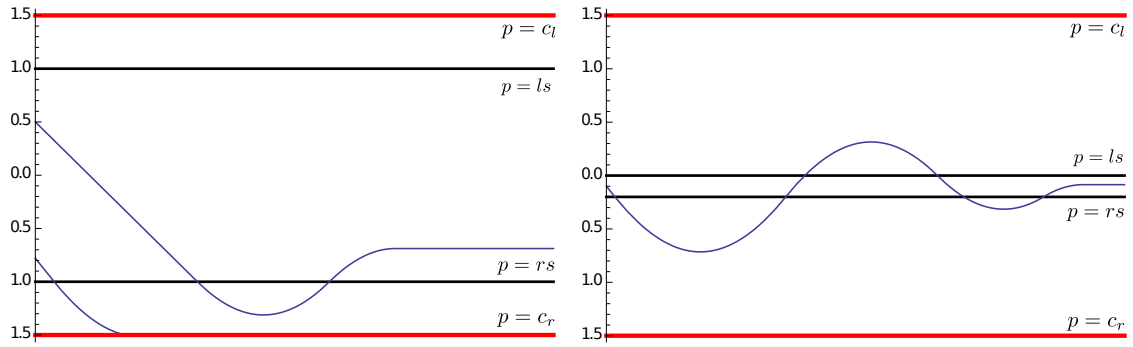


Figure 6.7: Hybrid automaton model of the self-driving car

The car starts initially on the road with an appropriate heading for moving forward (i.e. $\gamma \in (\pi/-2, \pi/2)$). If it hits either the left or right hand side of the road, the car will initiate a turn either at an angular velocity of ω or $-\omega$. The clock will start recording at this point. If the car enters the canal, this is represented in the hybrid automaton by the speed and acceleration of the car being set to zero. If the car has come back on the



(**a**) Two trajectories, wide road (**b**) Multiple corrections, narrow road

Figure 6.8: Steering car trajectories

road, then a turn in the opposite direction is commenced at a rate of $-\omega$ or ω. The clock now decreases at twice the rate to enforce a half turn. If the clock reaches zero, then the turn is deemed to have finished and the car continues straight on. During this second turn, it can happen that the opposite side of the road is reached. In this case, another corrective manoeuvre is made. Several of these different scenarios are shown in Figure 6.8. In Figure 6.8a, two trajectories of a car are shown. One shows the car falling into the canal and the other shows the car making two corrective turns to return back onto the road. Figure 6.8b shows how multiple corrections might have to be made in the case of a narrow road where, during the second corrective turn, the other side of the road is reached.

6.3.2 Experimental Results

The instance of the self-driving car model investigated with QUANTUM used the following parameters: the road was modelled by $ls = 1$, $rs = -1$, the canal by $c_l = 2$, $c_r = -2$. The turn speed and linear velocity were respectively set to $w = \frac{\pi}{4}$ and $v = 2$. The verification goal was to guarantee the safety of the system. That is, for all starting positions $p_0 \in [-1, 1]$ and heading angles $\gamma_0 \in [-\frac{\pi}{4}, \frac{\pi}{4}]$, the canal was unreachable.

The first experiment used the system definition, guards, invariants and state variables to populate the set of discretising functions F.

$$F = \{c, x, x-1, x+1, x-2, x+2, \gamma - \frac{\pi}{4}, \gamma + \frac{\pi}{4}\}$$

QUANTUM created an abstraction of 142 states (289 proved infeasible) with 310 proved transition conjectures in 84 seconds. Unfortunately, the safety condition $G\neg(p \geq 2 \vee p \leq -2)$ was violated by the abstraction. The CEGAR loop was unable to show that any of the abstract states or transitions were spurious. The second experiment added several Lie derivatives of each mode's vector field to the set F in an attempt to refine the

abstraction.

$$F = \{c, x, x-1, x+1, x-2, x+2, \gamma - \frac{\pi}{4}, \gamma + \frac{\pi}{4} \\ -2\sin\gamma, \frac{\pi}{2}\cos\gamma, 8\pi^2\sin\gamma\}$$

With this set F, QUANTUM created an abstraction of 52 states (274 proved infeasible) and 817 proved transition conjectures in 94 seconds. Although the abstraction in this experiment was smaller and many spurious behaviours were removed, the safety property was still violated. The CEGAR loop was again unable to refute any states or transitions.

Analysing the abstractions from the two experiments in more detail revealed several interesting facts. In the first experiment, there were ten abstract states that violated the safety property. Adding the Lie derivatives to the set of discretising functions F dropped that number to three. As the previous case studies have shown, including Lie derivatives can reduce the size of abstractions produced by QUANTUM. Each of the resulting error states was truly feasible[1] with respect to the functions in F and the transitions were correct according to the abstraction rules outlined in Section 5.2.

The negative result highlights one weakness of the qualitative abstraction process with regards to dynamical systems that make decisions based on a clock, such as in the self-driving car example. As none of the functions in F contains the variable t, the resulting discrete abstraction is completely independent of the flow of time. This loss of time information can lead to spurious behaviour being included in the abstraction regardless of what functions are in F. For example, consider a trajectory of the car that enters the right shoulder region of the road ($p = -1$). Since the abstraction does not take time into account, no matter what heading angle γ the car takes to enter the shoulder, all possible values are considered for transitions out of the region $-2 < p < -1$. Because the vector field in that mode is only dependent on γ, the next abstract state will always be allowed to hit the boundary of the canal, violating the safety property.

One way to address this problem is to pick more discretising functions to include in F. For instance, a variable t representing time could be added and each of the other functions in F could be re-written in terms of this new variable t. However, this might not be ideal since there is no guarantee that the resulting abstraction would be able to prove the safety property of interest and more violating states might be introduced. Furthermore, all the effort spent building and analysing the original abstraction would be wasted. Another option to refute spurious qualitative behaviour would be to use any available quantitative information about the system's real trajectories.

[1] Confirmed using the computer algebra system Mathematica.

For the abstraction created in experiment 2, symbolic solutions were computed for each of the three states violating the safety property $G\neg(p \geq 2 \vee p \leq -2)$. MetiTarski was then used to prove that for the variable ranges defined by the state's qualitative value, the trajectories could not occur in the original system. For instance, QUANTUM indicated that there was a transition to the abstract state

$$\begin{aligned} S = \{c > 0, x + 2 = 0, x + 1 < 0, g > -\frac{\pi}{4}, g < \frac{\pi}{4}, \\ -2\sin\gamma > 0, \frac{\pi}{2}\cos\gamma > 0, 8\pi^2 \sin\gamma < 0\} \end{aligned} \tag{6.1}$$

where the function $x + 2 = 0$ indicated that the car reached the canal (a violation of the safety property). In that particular state, the symbolic solution to the system was calculated to be

$$p(t) = \frac{\pi p_0 + 8\cos\gamma - 8\cos(\gamma_0 - \frac{\pi t}{4})}{\pi} \tag{6.2}$$

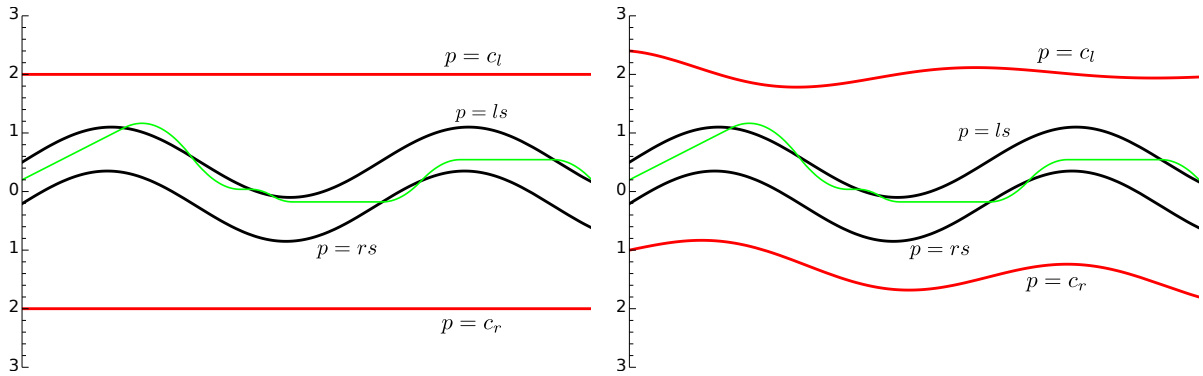
The variable ranges defined by the abstract state (6.1) were combined with the symbolic solution (6.2) and put into the appropriate MetiTarski format, resulting in the conjecture $\forall t : p(t) > -2$. MetiTarski was able to prove it in 2.3 seconds. For each of the other two violating states, MetiTarski was able to use similar trajectory information to prove that any transitions into the safety violating abstract states were physically impossible. The time for QUANTUM to construct the abstraction, compute the symbolic solutions and then use MetiTarski to automatically eliminate the spurious trajectories was 105 seconds in total.

6.4 Modified Self-Driving Car

6.4.1 Model Description

This final example considers the environment shown in Figure 6.9. The road the car is now travelling on is curved, where the edges of the road are modelled using sine and cosine functions. We consider two cases: The first shown in Figure 6.9a keeps the sides of the canal modelled by straight lines, and the second shown in Figure 6.9b models the sides of the canal by a combination of sine, cosine and exponential functions.

The hybrid automaton shown in Figure 6.7 was modified to properly handle the realigning of the car on the curved road. This new example is shown in Figure 6.10. The timer c has been removed. Now, after the second corrective turn takes place (either on the left or right shoulder), the car detects when it is pointing straight forward ($\gamma = 0$) and switches back to the default movement forward mode. If another side of the road is hit before γ reaches zero, then another corrective turn is taken.



(**a**) Straight canal edges

(**b**) Curved canal edges

Figure 6.9: Car travelling on a curved road

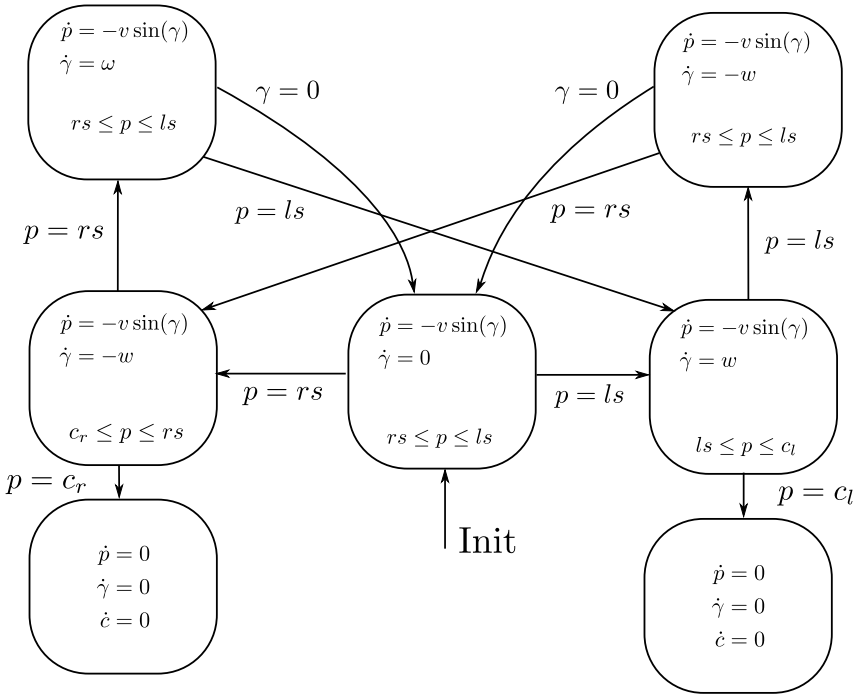


Figure 6.10: Modified hybrid automaton of the self-driving car

6.4.2 Experimental Results

The first experiment, with constant value on the bounds of the canal ($lc = 2, lc = -2$), used the following two functions to represent the curved road: the left side by $0.2\sin t + 0.5$ and the right side by $0.2\cos(t - 1.5) - 0.25$. These are the bounds shown in Figure 6.9a. The following set was chosen to discretise the state space.

$$F = \{x + 2, x - 2, \gamma - \pi/4, \gamma + \pi/4,$$
$$x - (0.2\cos(t - 1.5) - 0.25), x - (0.2\sin t + 0.5)\}$$

With a default timeout of 0.1 seconds, QUANTUM constructed an abstraction of 649 states in 6 minutes and 34 seconds. 1190 states were proved infeasible and there were 1190 transition conjectures proved. Out of the remaining feasible states, there was a total of 50 (4.2%) that violated the safety property by reaching the canal. Taking the same strategy as before, the symbolic trajectories of the system were used to refute each transition into a bad state. However, in this case, instead of a constant initial position p_0, a symbolic position had to be used instead. For instance, the conjectures for showing that the left side of the canal was not reachable had the form of

$$\forall t : ((\pi(-0.2\cos(t-1.5)+0.25)-8\cos\gamma)+8\cos(\gamma-\pi t/4))/\pi < 2$$

MetiTarski was able to prove the family of conjectures refuting the violating qualitative behaviour in an average of 3.7 seconds.[2]

For the final experiment, transcendental functions were used to represent the canal boundary. The left canal side was defined by $0.4\cos t\exp(-0.2t)+2$ and the right by $0.4\cos t\exp(-0.2t)-2$. These are the boundaries shown in Figure 6.9b. The set of discretising functions was chosen to be

$$\begin{aligned} F = \{&\gamma-\pi/4, \gamma+\pi/4,\\ &x-(0.4\cos t\exp(-0.2t)+2), x+0.4\cos t\exp(-0.2t)+2,\\ &x-(0.2\cos(t-1.5)-0.25), x-(0.2\sin t+0.5)\} \end{aligned}$$

With a default timeout of 0.1 seconds, QUANTUM created an abstraction of 1630 states in 14 minutes and 7 seconds. 579 states were proved infeasible and 1842 transition conjectures were proved. On further analysis of the abstraction, out of the 579 feasible states 60 (10%) violated the safety property. In this case, the conjectures used to refute spurious transitions not only contained symbolic initial states, but the condition for reaching the boundary to the canal was symbolic as well. For instance, the conjectures proving that transitions through the left hand side of the canal were impossible had the form

$$\begin{aligned} \forall t : &(\pi(-0.2\cos(t-1.5)+0.25)-8\cos\gamma+8\cos(\gamma-\pi/4))/\pi\\ &< -(0.4\exp(-0.2t)\cos t+2) \end{aligned}$$

MetiTarski was able to prove the family of conjectures in an average of 18.3 seconds.

[2]With an RCF timeout of 1000 seconds.

These last two experiments demonstrate that a combination of qualitative and quantitative methods for constructing and then refining abstractions is much more powerful than one that is purely qualitative. Not only does it allow for a much finer abstraction to be created, but it eliminates the need for adding more functions to the set F that can cause the abstraction time to increase to an unusable level.

6.5 Chapter Summary

This chapter has presented four separate case studies where QUANTUM was used to construct an abstraction of a nonlinear hybrid system. In each case, it was shown that using only the definition of the hybrid automaton itself resulted in an abstraction that failed to prove the required safety property. A series of refinement techniques were introduced to remove infeasible behaviour resulting in an abstraction that was good enough to prove the properties of interest.

The case studies presented in this chapter have been used previously as benchmarks for the evaluation of two verification methods for nonlinear hybrid systems (iSat/ODE [78] and hydlogic [117]). The most important difference between QUANTUM and these other tools is the validity of the verification result. The abstractions created by QUANTUM are guaranteed to be over approximations of all possible behaviours of the underlying hybrid automaton. iSat/ODE and hydlogic determine the reachable states of the hybrid automaton by unwinding the transition relation a fixed number of times. Therefore, if they return with the claim that a safety property is *verified*, it is actually only valid up to a bounded time limit. QUANTUM, on the other hand, guarantees for all time that the safety specification holds. However, with QUANTUM, dealing with spurious behaviour and a coarse abstraction can be difficult in its own right.

It was shown that the inclusion of energy constraints in the set of discretising functions F can be required for the removal of spurious behaviour that invalidates a safety property. Recasting the transcendental terms in the generated conjectures to a polynomial form can make the proofs easier for MetiTarski, resulting in a further reduction in the size of the abstraction. The CEGAR loop was shown to work for simple violating transitions of lengths of one step. However, a series of valid transitions originating from an infeasible state cannot be refuted. In such cases, the global timeout of QUANTUM must be raised, resulting in an increase in the overall abstraction time.

The abstraction process based on the analysis of the Lie derivative of continuous functions can fail when the passage of time is critical for the switching behaviour of the hybrid system. In this case, quantitative knowledge of the underlying continuous trajectories can be used to refute infeasible qualitative behaviour. The experimental results suggest that a semi-quantitative/semi-qualitative methodology could potentially reduce the amount of

time required to construct an abstraction, by not requiring the modification of the set F. Combining qualitative and quantitative methods would be the best avenue to follow for future improvements to the capabilities of QUANTUM.

CHAPTER 7

Related Work

This chapter covers alternative approaches to those given in this dissertation for modelling and verifying hybrid dynamical systems. Several are natural extensions of the reachability verification methods for purely continuous systems that were introduced in Section 3.1. Of note is the recent development of interactive theorem provers targeted directly at hybrid systems, which was touched on briefly in Section 4.1. Besides HybridSAL, there have been few notable applications of qualitative analysis to the verification of hybrid automata. For the related work concerning qualitative reasoning in general, see Section 5.1.

Section 7.1 reports on alternative ways to model hybrid systems. Different frameworks have been proposed, tailored to different applications. For instance, a modelling framework that is easy to simulate, might not be amenable to reachability verification methods and vice-versa. A framework might make it easy to model a system at a high level, but does not provide a way to split the model into parts that are simpler to analyse. Section 7.2 reports on the state of the art for the verification of hybrid systems, with a focus on abstraction based techniques that allow for the application of formal methods for discrete state systems.

7.1 Modelling and Simulation of Hybrid Systems

The most commonly used formalism for modelling hybrid systems in computer science is the hybrid automaton. Several variants exist, the choice of which one to use depends on the types of behaviour (delay, nondeterminism, uncertainty, stuttering, Zeno etc.) that are allowed and how they are modelled [9, 34, 135]. The general structure of a hybrid automaton, which was presented in Section 2.3.1, can be simply described as a finite state machine where each individual state defines a separate set of differential or difference equations governing some continuous behaviour.

The hybrid automaton originates from the verification of reactive systems [101]. These are systems that interact with their environments under some timing constraints (e.g. a secure communication protocol). The undecidability of checking the reachable states of a hybrid automaton [106] was an early theoretical result that limited verification techniques to very simple classes of hybrid systems. One breakthrough was the development of a reachability algorithm for timed automata [11], which are hybrid automata that are governed only by simple clocks, represented by the differential equation $\dot{t} = 1$. Although timed automata have been used to model and verify real dynamical systems such as a rail road crossing [7], the limited dynamics is one obvious weakness. Much effort has been expended on developing ways to conservatively abstract hybrid systems to timed automata. Several such abstraction techniques will be presented in Section 7.2.

The Hybrid I/O Automaton (HIOA) modelling framework [136] is a generalisation of hybrid automata that adds to the hybrid state, $Q \times X$, well defined input and output variables that track its external behaviour. These extra state variables explicitly model interactions with the environment and other hybrid I/O automata. One of the main strengths of the framework is that it allows a hybrid system to be decomposed into many simpler HIOA, making both modelling and analysis easier to conduct. The notion of abstraction, where one HIOA implements another, is explicitly defined by the framework, as is the concept of *receptiveness* where unwanted blocking and Zeno behaviour are disallowed. Safety properties of the HIOA can be verified using inductive methods over its executions. A non-trivial helicopter control system modelled as an HIOA was verified in this way [144].

In real world hybrid dynamical systems, the trajectories out of one state might be dependent on the value of a noisy signal. Take for example a wireless controller: instructions might be incorrectly received due to dropped packets or because of a sudden loss of power. This uncertain behaviour cannot be properly modelled by the frameworks described above, but stochastic hybrid systems [112] that incorporate probabilities into the model can. Discrete transitions are assigned a probability of being taken and continuous flows evolve according to stochastic differential equations (i.e. standard ODEs augmented with a random perturbation term). The reachability problem is now concerned with computing bounds on the probability that the system will reach an unsafe state. Probabilistic hybrid automata [200] restrict the assignment of probabilities to discrete jumps only. This simpler form has been shown to be amenable to abstraction methods that allow for the automatic verification of probability bounded safety properties [219].

Similarly to Hybrid I/O Automata, the hybrid program notation introduced by Platzer [169] addresses some of the weaknesses of hybrid automata. It is a textual, logical formula based, representation of the behaviour of a hybrid system that can be easily decomposed into several sub-formulas following sound transformation rules. It was developed so a *compositional* proof calculus could be used to reduce the verification of complex properties and systems into easier to prove parts. A proof of each of the individual parts

implies the correctness of the original property on the original system. Another motivation for this type of compositional verification is for scalability, as large complex proofs can be easily broken down into easier to manage parts. A hybrid automaton can be mapped to a hybrid program that is identical with respect to the reachable states of the corresponding hybrid system [168, p. 371]. The hybrid program representation is at the core of the deductive method implemented in the KeYmaera theorem prover for hybrid systems [171].

There are several language based modelling frameworks for hybrid systems: SHIFT [73], Massaccio [104], Charon [10], ExCharon [98], Zélus [30], to name a few. They each define formal hierarchical models for components that can interact with each other through channels or shared variables. The goal of these frameworks is ease of modelling and simulation. Of particular note to this dissertation is QCharon [199]. It is a modification of standard Charon that replaces all shared variables with qualitative variables, allowing for the application of qualitative simulation to analyse the behaviour of a hybrid model (see Section 5.1.2 for an overview of qualitative simulation).

The automata and hybrid program based modelling frameworks described so far come from the domain of computer science. Consequently, they are appropriate for applying automated and interactive deductive methods for proving safety properties about their behaviour. On the other hand, the study of stability and simulation of hybrid systems (specifically with control inputs, impacts, discontinuities, jumps, sliding modes etc.) has been investigated by the control engineering community for quite some time. There is frequently a disconnect between the two domains as their priorities often differ. A computer scientist may be content with the exhaustive safety verification of a hybrid model that is relatively small in size (that contains only simple jumps, no inputs etc.). The control engineer may consider such a model trivial for their standard simulation methods, control synthesis or stability analysis. There are quite a few control oriented modelling frameworks for hybrid systems, several of note will now be discussed.

Switching systems [132] are a type of model that describe the behaviour of continuous-time and discrete-time systems that are governed by a set of ordinary differential equations. A switching signal indicates which ODE is active according to some predicate. Switching systems can be similarly viewed as a restricted form of hybrid automaton with a reset mapping that is empty. During a mode change, the continuous variables are not allowed to jump to a new value. Validated solutions to switching systems have been used to verify parallel landing protocols for multiple aircraft on closely separated runways [76].

Another framework, which focuses on placing guarantees on the robustness of the asymptotic stability of hybrid systems, was proposed by Goebel et al. [89]. It separates hybrid system behaviour into four components: a flow set, a flow map, a jump set and a jump map. The maps define the allowable values for the continuous and discrete variables (they can be seen as an invariant definition). The flows govern how the continuous and

discrete variables change with respect to time (the differential equations and the reset map). The simulation toolbox HyEq [186] implementing this framework is available for MATLAB. Related frameworks from the same control-oriented viewpoint include the equation and event-flow formulae representation of Antsaklis et al. [15], Branicky [34], Tavernini [206] and the behvioural representation of van der Schaft and Schumacher [214].

Navarro-López and Carter [156] have investigated ways to augment the standard hybrid automaton model to handle more control oriented properties such as nonlinear discontinuities and sliding motion. Their discontinuous dynamical system hybrid automaton (DDS-HA) framework allows for the simulation of hybrid systems containing non-trivial dynamics such as friction. Additionally, the framework allows for multiple DDS automata to be composed together, permitting the analysis of control systems with multiple discontinuity surfaces. This methodology was successfully applied to the simulation and analysis of a complex model of an oil-well drill-string [39].

There are several ways to simulate hybrid systems using a variety of commonly encountered engineering tools. LabVIEW, MATLAB (Simulink), System Builder (Modelica) and Mathematica can all handle various kinds of hybrid automata. In particular, there are several MATLAB toolboxes that combine simulation and formal methods for the verification of hybrid systems. Breach [75] can estimate the reachable sets of systems with uncertain parameters via a finite number of simulations. It does this by performing a sensitivity analysis on the trajectories of the system, under influence of the unknown variables.[1] There is also some support for synthesizing the required parameters of a system based on parameters written in a temporal logic. S-TaLiRo [14] uses randomised testing and Monte-Carlo techniques to search for trajectories of Simulink models that violate temporal logic properties. PyDSTool [54] is a simulation environment written in Python that can handle hybrid models, especially those arising from biological domains. Another relevant analysis environment is Ptolemy II [175], which focuses on using an underlying object oriented modelling language to connect components from different physical domains.

7.2 Formal Verification of Hybrid Systems

The concept of abstraction has been the key idea behind several techniques in the hybrid system verification domain. This is primarily due to the undecidability barrier of the safety verification problem of general hybrid automata [106]. The idea of looking for

[1] The solutions of the system are approximated by a linearisation, via a Taylor expansion where the higher order terms are dropped to obtain an estimate for the reachable set.

ways to convert the reachability problem into one that could be solved by model checking led to several early breakthroughs. The methods described in this dissertation are an example of this general methodology.

7.2.1 Linear Dynamics

HyTech [9] is a model checker for linear hybrid automata, where the guards, reset maps and invariants are all defined by linear inequalities. The definitions of the vector fields are only allowed to contain constant bounds over the first derivatives of the continuous variables. The reachable set of a linear hybrid automata is represented by a finite union of polyhedra, which are defined as a conjunction of linear inequalities. Although these conditions are very restrictive, they allow for the efficient computation of the reachable states. The techniques were further extended to general nonlinear hybrid system, which could not be analysed by HyTech, by translating them into linear hybrid automata [105]. The most successful approach was one that over approximates the nonlinear flow using manually chosen linear constraints. This method, called linear phase-portrait approximation, is in essence a technique to discretise the hybrid state space into an abstraction that simulates the original system. This discretisation process can be seen as a direct precursor to the techniques described in this dissertation. Several improvements to HyTech were implemented in PHAVer [82], most importantly the on-the-fly over-approximation of continuous dynamics. Other related tools of the same vintage for verifying restricted dynamics or approximated hybrid dynamical systems include Cospan [8], Kronos [32] and UPAAL [20].

The next generation of verification methods focused on approximating the continuous dynamics of the hybrid system by a union of polyhedra called a flow pipe. Chutinan and Krogh [45] describe the construction of a flow pipe by numerically solving an initial value problem to approximate the end points of a polyhedron for some time point t. The determination of the polyhedra that approximates the flow of the system, based on the calculation of the end points, is then reduced to an optimisation problem. This method was implemented in the MATLAB toolbox Checkmate [46]. It is important to note that the input to Checkmate is a Polyhedral-Invariant Hybrid Automata (PIHA). Reset mappings are disallowed, invariants must be defined by linear inequalities and the guards must be on the face of an invariant (i.e. transitions must immediately occur when a guard is satisfied). Similar methods were implemented by Asarin et al. in the d/dt tool [16], which computes a flow pipe using non-convex orthogonal polyhedra that are a union of hyper-rectangles. There are various space saving properties of orthogonal polyhedra versus convex polyhedra. For instance, it is possible to check if one orthogonal polyhedron is contained in another, giving a unique representation for the set of reachable states. d/dt is restricted to analysing hybrid systems with affine continuous dynamics.

Predicate abstraction was touched on briefly in Section 5.1.1. It is a technique that constructs a finite-state model from a possibly infinite-state system, using a finite set of predicates evaluated over a Boolean domain. One application of predicate abstraction to the verification of hybrid systems was proposed by Alur et al. [13]. The analysable hybrid systems are completely linear (all invariants, guards, resets and differential equations contain linear terms). The predicates must also be defined by linear equations. These restrictions arise from using the flow pipe approximation techniques described above, which are limited to linear systems. The reachability computation over abstract hybrid states was enhanced by a CEGAR loop [12, 51]. When an abstract counterexample is returned, an approximated flow pipe is constructed in an attempt to show that it is spurious in the concrete model. If this process is successful, a process based on the computation of a Lyapunov function, given by a union of polyhedra, is used to separate the abstract states. This function is then added to the set of predicates and the reachability process restarts with the new set of predicates.

The process of converting general nonlinear hybrid systems into linear automata can be computationally expensive, leading to an abstraction that can be quite coarse and therefore of little use. Also, the time complexity of manipulating polyhedra is exponential in the number of continuous variables [6]. This has prompted much research into finding more efficient representations of reachable sets. Several were mentioned briefly in the discussion on reachability methods in Section 3.1.3. The main difficulty in applying these reachability methods to hybrid systems is computing the intersection of the flow pipes with the transition guard. This is especially difficult when a guard is defined by a nonlinear function.

Tiwari and Sankaranarayanan [187] have proposed a new type of hybrid system abstraction technique that summarises the behaviour of the dynamics in each mode. This is accomplished by finding a *relational abstraction* of the type $R(x, y)$ that represents the sets of states that flow continuously from x to y (they are equivalent to a positive invariant set). They make use of techniques to generate template invariants that are then used to abstract the continuous dynamics of the hybrid system into an infinite discrete state system. These abstractions are then verified using standard techniques such as k-induction and bounded model checking. Their techniques for generating the abstractions are not directly applicable to nonpolynomial systems. However, for nonpolynomial hybrid systems that contain some modes that are purely polynomial, relational abstractions could be combined with qualitative abstractions as both are just discrete state systems. These methods are implemented in a tool called HybridSAL[2] [210]. Relational abstractions completely remove any dependence on time, which can cause the abstractions to be quite coarse and include many spurious behaviours. Time-Aware relation abstractions [146]

[2]The relational abstracter mentioned here shares only its name with the previously discussed qualitative abstracter HybridSAL [208]. Throughout this dissertation any use of name HybridSAL refers specifically to the qualitative version.

provide a way to retain timing information in the abstraction, by using piecewise linear approximations of the trajectories of the system. These methods, however, are all limited to linear hybrid systems that have closed form solutions.

7.2.2 Nonlinear Dynamics

The hybridization technique [18, 60] specifically targets the analysis of systems with continuous nonlinear dynamics. The continuous system is converted into a linear hybrid automaton, which can then be analysed using any of the tools described previously. There are two types of hybridization that have been investigated: in the original *static* method, the state space is first discretised into distinct cells where the system is linearised. Within each cell, the system is approximated by a linear function and an error term. Combining the cells together results in a piecewise-affine hybrid system. In the improved *dynamic* method [59], new cells are generated on-the-fly only when it is determined that the continuous trajectories reach a switching surface. As with most cell based abstraction methods,[3] the number of continuous variables must be kept low. In spite of this limitation, hybridization has been successfully used to automatically verify the safety of complex manoeuvres of orbiting satellites [120].

The Bernstein polynomial representation[4] [24] has recently been employed for the verification of continuous dynamical systems. The maximum and minimum coefficients of a polynomial written in the Bernstein form give conservative bounds on the range of a multivariate polynomial over the unit cube. The difficulty lies in extending this to arbitrary intervals $[a, b]$. Dang and Testylier [58] introduced a method for over approximating the reachable set of polynomial discrete-time dynamical systems using template polyhedra. These are a special type of convex polyhedra that can be efficiently manipulated in higher dimensions [188]. The Bernstein polynomial representation allows the computation of the transition from one state (template) to the next, which is a convex optimisation problem, to be reduced to one that can be solved with linear programming. This methodology was implemented in the NLTOOLBOX [207], which also contains procedures for applying hybridization based techniques. Muñoz and Narkawicz [150] formalised a Bernstein representation in PVS and then used it to verify a global optimisation algorithm. These methods were improved upon by Cheng et al. [44] with the JBernstein tool, giving an order of magnitude speed up over the PVS implementation. The dynamical systems that can be analysed by these methods are however all restricted to polynomial vector fields.

There have been several developments involving the construction of flow pipes of nonlinear polynomial hybrid systems based on Taylor Models and template polyhedra [189]. Taylor Models approximate a function over a bounded range by a truncated higher-order Taylor polynomial combined with an interval to represent the approximation error. They

[3] Shared by the QUANTUM software described in this dissertation.
[4] They are also known as Bézier curves.

allow a much tighter approximation of nonlinear dynamics than general polyhedra. The trajectories of the dynamical system are approximated using a Taylor Model and then converted into template polyhedra. In this form, the intersection with the guard conditions can be computed efficiently. These methods were implemented in the Flow* tool [43]. The templates that are used to represent the reachable states of the system are user provided, which as with choosing the functions to include in the set F for QUANTUM, can be the deciding factor as to whether a positive verification result can be achieved. A related framework is Ariadne [21], which uses a similar Taylor model based representation for computing reachable sets, but instead of template polyhedra it uses a variable grid-based discretisation of the state space. The algorithms used by Ariadne have been developed and proved formally correct using the Coq theorem prover [55].

7.2.2.1 Discrete Abstraction Based Methods

Seldom cited in hybrid system literature is the work of Sacks [185], who developed qualitative analysis methods for systems defined by nonlinear differential equations. In his methodology, nonlinear vector fields are repeatedly discretised by piecewise linear approximations until no new qualitative behaviour is detected. A rudimentary theorem prover for inequalities is used to determine transitions between the abstract states. There is no formal verification methodology applied to the generated transition systems, primarily because the work pre-dates the establishment of model checking as a viable method to do so. This work can be seen as the direct ancestor to the family of verification methods for continuous and hybrid systems that discretise the state space, including HybridSAL and QUANTUM.

This dissertation has described several extensions of the original version of HybridSAL [211]. There are several notable theoretical results obtained by Tiwari [209] that are applicable only to qualitative abstractions of the restricted class of polynomial systems. For instance, he has shown that under certain conditions (there is no external input signal and there is no shared variable in the guards) the abstractions created by HybridSAL can be composed together. For purely continuous systems, if the process of taking repeated Lie derivatives terminates,[5] then the sign of each $f_i \in F$ is unique for each abstract state. This gives a bisimulation relation between the abstract and concrete systems, guaranteeing the abstraction is complete (see Section 5.1.3). This completeness result was independently proved by Tabuada using a similar sign-based abstraction process [204].

One early application of qualitative abstraction to analyse hybrid systems was published by Stursberg et al. [201]. In their work, a hybrid dynamical system is discretised by hyperboxes. The intervals are based on the structure of the system (these are the landmarks that are critical to its operation). The amount of time spent in the boxes is approximated by calculating the Lie derivative at sample points within the boxes. This analysis

[5] As is the case with linear polynomial systems.

results in a semi-quantitative abstraction that can be represented by either a timed automaton or a linear hybrid automaton. They compared the qualitative behaviours of the abstraction to those of the real system and found that in general there are many spurious behaviours in the discrete state model.

An abstraction that preserves timing information, such as the process of converting a continuous dynamical system to a timed automaton described above, is critical for proving liveness properties of the form: "Does the dynamical system eventually reach some required location of the state space?" The difficulty, as is shared by the qualitative abstraction methods described in this dissertation, is choosing how to discretise the state space. Carter and Navarro-López [40] proposed refinements to method originally suggest by Bhatt and Maler [139], to discretise the state space based on properties of the underlying vector field. By carefully choosing the location to split the state space, they are able to show that the resulting timed automata abstraction of a restricted form of linear system is able to prove a certain class of liveness properties. The work on verifying the liveness of hybrid automata and proving other related properties was greatly expanded in Carter's PhD thesis [41].

Sloth and Wisniewski [196, 198] have developed a sound and complete method for constructing abstractions of continuous systems using sub-level sets of Lyapunov functions. Each abstract region created by this process is positively invariant and can be used as a discrete state of a timed automaton. The invariants and guards of the time automaton are then generated by solving an optimisation problem. This allows the use of tools, such as UPPAAL, that can automatically check properties of timed automata. Their techniques are geared to *Hirsch-Smale* systems, which are guaranteed to have polynomial Lyapunov functions. It is important to note that many examples from their work can be solved analytically and MetiTarski/QUANTUM has been successful at verifying them without the need for any form of abstraction [67].

7.2.2.2 Interval Based Methods

The methods described so far have been limited to linear and nonlinear polynomial dynamical systems. Early in the development of hybrid system verification methods, the domain of intervals was found to be expressive enough to over approximate the reachable states of systems defined by nonpolynomial vector fields. HyperTech [107] uses an interval ordinary differential equation solver to enclose the trajectories of systems that are defined by trigonometric functions. Ishii et al. [116, 117] have developed similar methods employing a hybrid constraint system to represent the continuous trajectories of the system. They combine an interval ODE solver within an SMT framework to verify time bounded executions. The work of Eggers et al. [78] uses the interval-based solver VNODE-LP combined with iSAT, which integrates SAT solving and interval constraint propagation, to form a bounded model checking framework for nonlinear hybrid sys-

tems. A related SMT encoding for linear hybrid automata was proposed by Cimatti et al. [48] to allow the translation of mode invariants into a quantifier free form. This is accomplished by repeatedly taking its derivative and checking whether over a continuous time interval $[t, t']$ its value is constant.

All these methods are concerned with finding efficient ways to over-approximate the behaviour of a hybrid system. This essentially reduces to finding efficient ways to represent the continuous flow of ordinary differential equations as a set of constraints that must be solved. Gao et al. [85, 86] have developed δ-complete decision procedures for SMT formulas containing hundreds of nonpolynomial constraints over real variables and nonlinear differential equations. δ-completeness allows a formula to be labelled satisfiable under small perturbations using guaranteed interval constraint propagation methods. This approximation has been shown to be suitable for developing practical algorithms for solving an SMT encoded reachability problems for nonlinear hybrid systems (implemented in the tool dReach [87]). In general, because these methods rely on unwinding a transition relation a fixed number of times (a bounded model checking framework), the verification results are only valid for limited time horizons. This means that safety results given by QUANTUM are much strong than those given by dReach, because they are valid for all time.

Extending the interval enclosures for nonpolynomial systems, the tool HySon [28] represents reachable sets using zonotopes, which are a compact and efficient representation of a restricted subset of polyhedra. Guaranteed numerical simulation algorithms are used to compute bounds on the trajectories of the system that can then be used to accurately determine when a nonlinear guard evaluates to True. Although HySon is mostly useful in a simulation context, its flow pipe approximation methods could be used by the hybrid system verification methods outlined above to extend their application to fully nonpolynomial hybrid systems. Unfortunately, HySon is not publicly available. Again, as this is simulation algorithm, the safety results are only valid for finite time horizons. QUANTUM has no such restriction.

HSOLVER [178] uses a similar framework as described above, which discretises the state space into a grid of boxes and then uses interval arithmetic to compute flows between them. The reachability problem is recast as an abstraction/refinement problem where the interval boxes are considered as abstract states. To determine the transitions between them, a constraint satisfaction problem must be solved. Qualitative knowledge about the flows of the system inside the boxes are used to refute infeasible abstract transitions. This heuristic limits the amount of times a box must be split when the refinement loop is called. HSOLVER is unable to handle the bouncing ball on sine curve example given in the case studies chapter, it simply times out with “Safety Unknown”.

7.2.2.3 Non-Reachability Based Methods

The concept of using a barrier certificate (described in Section 2.2.5) to certify the safety of a continuous dynamical system has been extended to hybrid systems by Prajna and Jadbabaie [174]. Instead of defining a single barrier certificate that must hold globally, one can be defined for each discrete state. Then, imposing strict conditions on the reset mapping, the existence of the set of barrier certificates implies that any unsafe state can never be reached. These constraints are easily written as a sum of squares program and can be solved using the SOSTOOLS method described in Section 5.1.5. A related concept from dynamical system theory is guaranteeing the stability of a hybrid system via multiple Lyapunov functions Branicky [35], Navarro-Lopez and Laila [157]. Sloth et al. [197] propose a method to verify the safety of interconnected systems by generating barrier certificates for each subsystem that are then coupled via extra constraints. Their method, however, is limited to dynamical systems with polynomial vector fields. Their techniques can not analyse many of the examples given in this dissertation.

An interesting alternative to reachability analysis is the logic-based deductive method for analyzing hybrid systems by Platzer [168], who has developed a sound and relatively complete proof calculus for hybrid systems. Central to this work is a technique called *differential induction*, which allows reasoning about differential equations without having to solve them directly. It is essentially a way to formally reason about the qualitative behaviour of the system's vector field. The method developed by Platzer is very powerful and the applications are quite impressive. MetiTarski/QUANTUM has been integrated with KeYmaera to discharge proofs that contain the nonpolynomial solutions of differential equations [118].

CHAPTER 8

Conclusion

This dissertation has been primarily concerned with using the theorem prover MetiTarski to automatically verify the behaviour of continuous and hybrid dynamical systems. The main advantage of this approach is that it guarantees a system is completely safe under all possible operating conditions. Such a result cannot be obtained with only a finite number of simulations, which is the limitation imposed in the standard methods for certifying the behaviour of dynamical system models.

8.1 Contribution Summary

The main difficulty in my approach was finding an appropriate way to represent a dynamical system's behaviour as a conjecture to be proved by MetiTarski. The key contributions described in this dissertation are summarised as follows:

- Chapter 3 covered the verification of purely continuous dynamical systems that admit closed form solutions. The verification of such systems can be represented as a symbolic reachability problem. I showed that the reachability problem, expressed as a first-order formula over time, could be solved using MetiTarski. This general methodology was developed initially by myself and then improved through several collaborations. Various collaborators and myself have applied the methodology to various domains, including: the verification of analogue circuits, controller stability and flight collision avoidance problems [66, 70–72], several of which were presented as examples in this dissertation.
- Chapter 4 covered the integration of MetiTarski and the interactive theorem prover PVS. I showed how the logical sequents generated by PVS could be handled by MetiTarski, outperforming the methods for nonlinear reasoning available using PVS alone. This work was conducted in collaboration with César Muñoz of the NASA Langley Research Center [69].

- Chapter 5 covered the verification of hybrid dynamical systems. Qualitative reasoning was used to construct a discrete state abstraction, which could then be verified using model checking. Even for very simple dynamical systems, this process required the quick processing of many conjectures containing transcendental functions. I was able to show that MetiTarski was well suited for this purpose [67, 68].

Another challenge was dealing with the doubly exponential complexity of the back-end decision procedures employed by MetiTarski. Chapter 3 investigated several high-dimension problems which could not initially be proved. This led to the development of techniques to make the verification problems more tractable. The main finding was that enforcing a time limit on the decision procedure nlsat (provided by the Z3 SMT solver) made it possible, for the first time, to prove in a reasonable amount of time conjectures of up to 9 continuous variables. Other techniques developed included recasting away transcendental functions and adding extra constraints to the problem. Finally, in some cases, it was possible to break one difficult conjecture into several easier sub-problems.

The positive results described in Chapter 4 were the main motivation to select MetiTarski, rather than PVS, as the proof machinery used by the hybrid system verification framework QUANTUM. An additional benefit was that the implemented proof strategy made it easy to generate interesting problems that allowed further assessment of MetiTarski's capabilities.

Using the experience gained from the experiments on purely continuous systems, the verification of hybrid dynamical systems was targeted next. Their models contained transcendental functions in their guards, invariants and vector fields. The verification of this class of system is difficult, even for state-of-the-art verification tools. A qualitative reasoning framework was chosen primarily because the verification problem could be split into a series of inequalities, which is the form of conjecture that MetiTarski can solve. The qualitative abstracter QUANTUM was built and shown to work on a series of non-polynomial hybrid system benchmarks.

I showed in Chapter 5 that if nonpolynomial Lyapunov functions and nonpolynomial Barrier certificates are used to discretise the continuous state space, MetiTarski is able to prove the type of conjectures generated by the abstraction process. Although convex optimisation can be used to facilitate the generation of such functions, a candidate template must still be manually provided. Consequently, this process is not always guaranteed to succeed.

To enable QUANTUM to abstract systems in a reasonable amount of time, MetiTarski's time limit was required to be set quite low. In this first iteration, the resulting abstractions contained many extra behaviours not present in the original system. The qualitative abstraction process was improved by implementing a simple CEGAR loop. When an abstract counterexample was found, the time limit could be increased on the

fly. Another improvement, which gave a noticeable speed up to the abstraction time, was processing independent conjectures in parallel. All these improvements were able to greatly reduce the number of spurious behaviours in the generated abstractions.

8.2 Future Work

The results from the experiments of this dissertation demonstrate that MetiTarski is capable of automatically verifying the behaviour of dynamical systems. As well, I have shown that combining qualitative reasoning with a theorem prover is a viable approach for the verification of nonpolynomial hybrid systems.

There are several avenues that could be followed to further extend the work initiated in this dissertation:

- The abstraction process uses a set F of functions to discretise the state space. Lie derivatives, Lyapunov functions and barrier certificates have all been used for this purpose. I have shown that if the appropriate functions can be found, MetiTarski will be able to handle the complexity of conjectures that are generated. However, it would be of further interest to investigate how these functions implicitly define an invariant of the state space. Instead of checking the safety of a system, QUANTUM could be used to search for possible invariants, represented as a union of sign conditions over the discretising functions.

- QUANTUM implements a fully qualitative abstraction process. It was shown however that combining both qualitative and quantitative information can be useful when the qualitative analysis fails to prove the required safety property. QUANTUM could potentially benefit from being able to use information provided by interval differential equation solvers to determine potential bounds to be included in the set F of discretising functions.

- The rudimentary CEGAR loop has been shown to only work for counterexamples of one step. A more thorough implementation would include a better search strategy for finding the violating concrete state of the abstract counterexample.

8.3 Final Comments

Hybrid systems have been of great interest to the formal verification community because they are the appropriate model for many real world safety critical applications. Recently, a more general class of models called cyber-physical systems has been proposed. These models represent not only the controller but also the behaviour of multiple agents that must coordinate together to accomplish a task. The model may also define the properties

of the underlying network fabric that the agents use for communication. These systems are highly nonlinear due to the various connections with the external environment. Undoubtedly, verification frameworks for cyber-physical systems will benefit from access to automated theorem provers that can handle nonlinear, transcendental and other special functions. It is my hope that the lessons learned from designing and building QUANTUM will make it easier for MetiTarski to be used for this purpose.

APPENDIX A

QUANTUM - Qualitative Abstractions of Nonpolynomial Models

The QUANTUM tool is available for download at:

`http://www-dyn.cl.cam.ac.uk/~wd239/quantum/`

The following appendix will demonstrate how to use it to produce a discrete state abstraction of a hybrid system modelled as a hybrid automaton. The example is a simple model of a thermostat and is located in `examples/heater-thesis.py` of the QUANTUM distribution.

A.1 Model Input

Take for example a simple model of a thermostat controlling the heater of a room, shown in Figure A.1. There are two modes: ON when the room is heating; OFF when the room is cooling. Transitions between the two states are governed by the guards indicated on the corresponding edges. In each mode, an invariant designates when a mode change must immediately take place.

The input to QUANTUM is a Python dictionary that defines each part of the hybrid automaton. First we define several functions that will represent the guards and invariants. `MetitPredicate` is a class that represents a mathematical function that includes a predicate symbol. Each discrete state is given a string name and put in the list `q`. The safety condition, which is a predicate that should not evaluate to true, is assigned to the `bad_state` variable.

```
pre = x-27
```

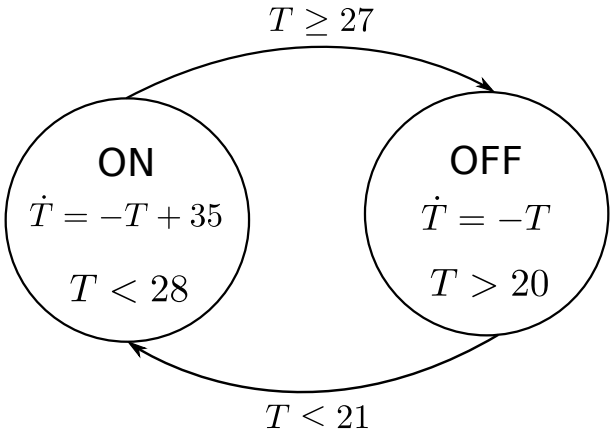


Figure A.1: Hybrid automaton of a thermostat controller

```
g_pred_27gt = MetitPredicate(pre,'>')
g_pred_27eq = MetitPredicate(pre,'=')

pre = x-21
g_pred_21lt = MetitPredicate(pre,'<')
g_pred_21eq = MetitPredicate(pre,'=')

pre = x-28
g_pred_28gt = MetitPredicate(pre,'>')
g_pred_28eq = MetitPredicate(pre,'=')

pre = x-20
g_pred_20lt = MetitPredicate(pre,'<')
g_pred_20eq = MetitPredicate(pre,'=')

bad_state = g_pred_20lt

q = [('on','off')]
```

The variable `system_def` holds the Python dictionary describing the hybrid automaton. Each discrete state is represented by a tuple key. Within each discrete state there is another dictionary that contains four fields `flow`, `t`, `inv` and `colour`. The `flow` field contains a dictionary assigning to each continuous variable's derivative, the flow equation. The dictionary defines a system of ordinary differential equations. The `t` field is the transition relation, which contains a list of dictionaries that represent each possible guarded transition. The `guard` field lists predicates that when evaluated to True, allow a transition to occur. The `next_state` key designates precisely which state the system will transition to. Finally, the `updates` key is an assignment function for the continuous

variables in the case of a discrete jump. The `inv` field lists all predicates that are allowed in the discrete state. The `colour` field is for colouring the nodes of the abstraction using the graphiz toolset.

```
system_def = {('on',):
              {'flow': {x.diff(t): -x+35},
               't': [{'guard': [[g_pred_27gt],[g_pred_27eq]],
                      'next_state': ('off',),
                      'updates': ()}],
               'inv': (g_pred_28eq, g_pred_28gt),
               'colour': 'green'},

              ('off',):
              {'flow': {x.diff(t): -x},
               't': [{'guard': [[g_pred_21lt],[g_pred_21eq]],
                      'next_state': ('on',),
                      'updates': ()}],
               'inv': (g_pred_20eq, g_pred_20lt),
               'colour': 'red'}}
```

The set F of discretising functions are defined in the `equations` variable. Here we use the class `MetitEquation` to represent the continuous equations. Its definition can be simplified using a standard Python list comprehension. The `var_id` label ensures that QUANTUM does not allow more than one transition between variables of the same name.

```
equations = [MetitEquation(*_) for _ in
             [(x-20,'var_id=1'),
              (x-21,'var_id=1'),
              (x-27,'var_id=1'),
              (x-28,'var_id=1')]]
```

To specify the initial abstract state of the hybrid automaton, the `initial_state` variable is assigned to a dictionary containing a key `d` mapped to the initial discrete state and a key `c` mapped to a series of predicates from the set F.

```
initial_state = {'d':('on',),
                 'c': [str(MetitPredicate(*_)) for _ in
                       [(x-21,'>'),
                        (x-27,'<')]]}
```

A.2 Using QUANTUM

To startup QUANTUM issue the following command at the terminal shell,

```
$python top_level.py
```

You should then be presented with the following prompt,

```
Quantum V0.1
(QUANTUM) >
```

To run QUANTUM on the example from Section A.1, issue the following command,

```
Quantum V0.1
(QUANTUM) > abstract heater-thesis
```

If succesful there should be no error messages. If the safety property is violated, the abstraction procedure will stop with the message `Transition to unsafe state detected`. Typing the command `g` will produce a file called `test.png`, which is a visual representation of the abstraction. The example from Section A.1 produces the abstraction shown in Figure A.2. A summary of the proof statistics will be found in the generated `log.txt` file. A SMV file, which can be used as input to the model checker NuSMV, will be generated and placed in the `smv` folder.

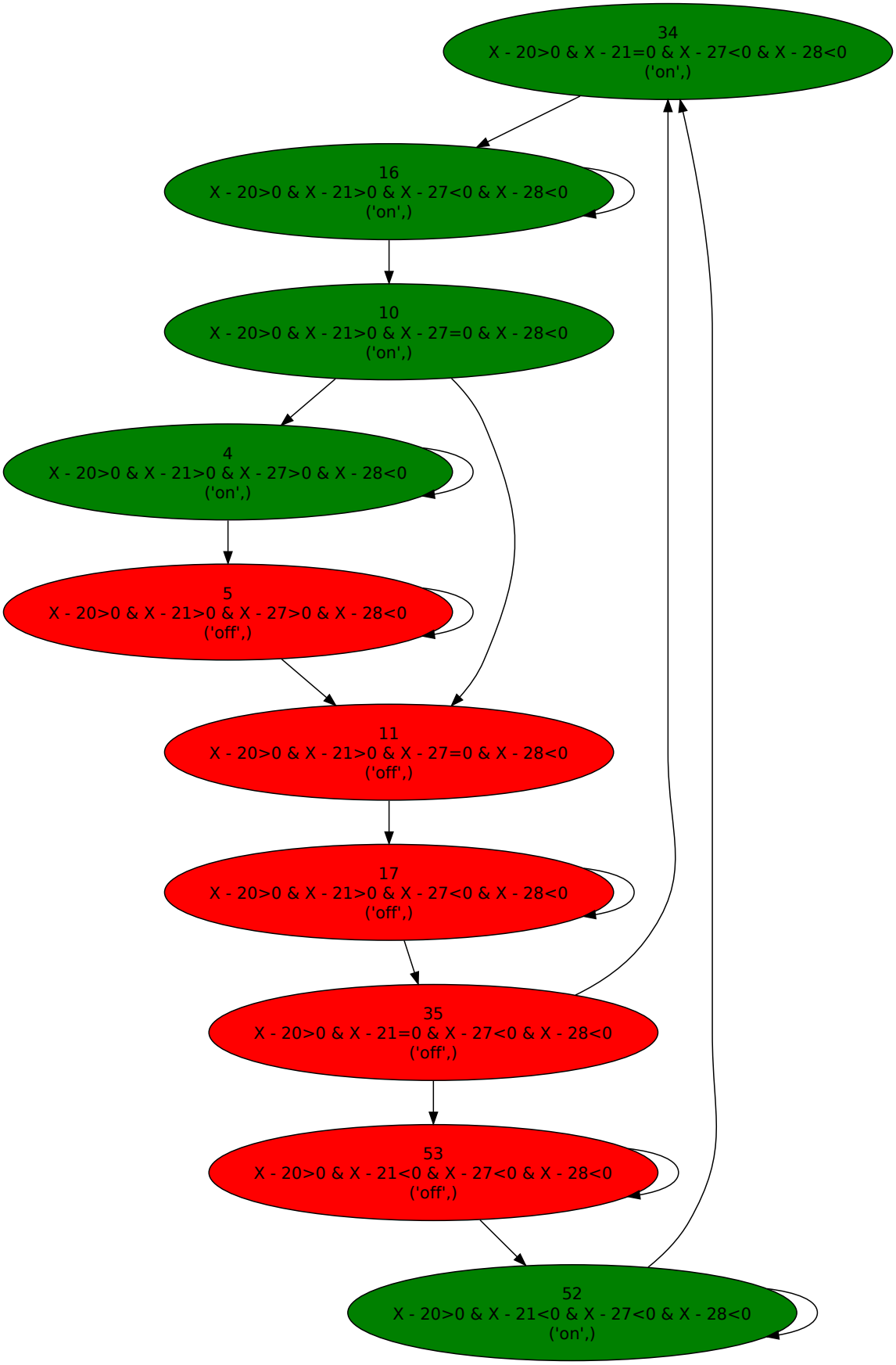


Figure A.2: Thermostat abstraction

Bibliography

[1] NASA PVS Library, 2014. `http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html`.

[2] Terence S. Abbott. Flight test evaluation of the airborne information for lateral spacing (AILS) concept. Technical Report TM-2002-211639, NASA Langley Research Center, 2002.

[3] Behzad Akbarpour and Lawrence C. Paulson. Applications of MetiTarski in the verification of control and hybrid systems. In *Hybrid Systems: Computation and Control*, LNCS 3414. Springer, 2009.

[4] Behzad Akbarpour and Lawrence C. Paulson. MetiTarski: An automatic theorem prover for real-valued special functions. *Journal of Automated Reasoning*, 44: 175–205, 2010.

[5] Rajeev Alur. Timed automata. *Theoretical Computer Science*, 126:183–235, 1999.

[6] Rajeev Alur. Formal verification of hybrid systems. In *Proceedings of the International Conference on Embedded Software*, pages 273–278. IEEE, October 2011.

[7] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[8] Rajeev Alur and Robert P. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III*, pages 220–231. Springer, 1996.

[9] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[10] Rajeev Alur, Radu Grosu, Yerang Hur, Vijay Kumar, and Insup Lee. Modular specification of hybrid systems in CHARON. In *Hybrid Systems: Computation and Control*, pages 6–19. Springer, 2000.

[11] Rajeev Alur, Thomas A. Henzinger, Gerardo Lafferriere, and George J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[12] Rajeev Alur, Thao Dang, and Franjo Ivancic. Counterexample-guided predicate abstraction of hybrid systems. *Theoretical Computer Science*, 354(2):250–271, 2006.

[13] Rajeev Alur, Thao Dang, and Franjo Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM transactions on embedded computing systems*, 5 (1):152–199, 2006.

[14] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257, 2011.

[15] Panos J Antsaklis, James A Stiver, and Michael Lemmon. Hybrid system modeling and autonomous control systems. In *Hybrid Systems*, pages 366–392. Springer, 1993.

[16] Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control*, pages 20–31. Springer, 2000.

[17] Eugene Asarin, Thao Dang, and Antoine Girard. Reachability analysis of nonlinear systems using conservative approximation. In *Hybrid Systems: Computation and Control*, LNCS 2623, pages 20–35. Springer, 2003.

[18] Eugene Asarin, Thao Dang, and Antoine Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43(7):451–476, 2007.

[19] 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*, 2010.

[20] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. *UPPAAL: A tool suite for automatic verification of real-time systems*. Springer, 1996.

[21] Luca Benvenuti, Davide Bresolin, Pieter Collins, Alberto Ferrari, Luca Geretti, and Tiziano Villa. Assume-guarantee verification of nonlinear hybrid systems with Ariadne. *International Journal of Robust and Nonlinear Control*, 24(4):699–724, 2014.

[22] Christoph Benzmüller, Lawrence C. Paulson, Frank Theiss, and Arnaud Fietzke. LEO-II: A cooperative automatic theorem prover for classical higher-order logic. In *Automated Reasoning*, pages 162–170. Springer, 2008.

[23] Daniel Berleant and Benjamin Kuipers. Qualitative-numeric simulation with Q3. *Recent advances in qualitative physics*, 98:285–313, 1992.

[24] Sergei N. Bernstein. Démonstration du théoreme de weierstrass fondée sur le calcul des probabilités. *Comm. Soc. Math. Kharkov*, 13(1), 1912.

[25] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Springer, 2004.

[26] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1579, pages 193–207. Springer, 1999.

[27] Daniel G. Bobrow, Matthew Klenk, Johan de Kleer, Bill Janssen, and John Hanley. Challenges for qualitative reasoning for engineering design. In *Proceedings of the 26th International Workshop on Qualitative Reasoning*, 2012.

[28] Olivier Bouissou, Samuel Mimram, and Alexandre Chapoutot. HySon: Set-based simulation of hybrid systems. In *2012 23rd IEEE International Symposium on Rapid System Prototyping*, pages 79–85. IEEE, 2012.

[29] Richard J. Boulton, Hanne Gottliebsen, Ruth Hardy, Tom Kelsey, and Ursula Martin. Design verification for control engineering. In *Integrated Formal Methods*, pages 21–35. Springer, 2004.

[30] Timothy Bourke and Marc Pouzet. Zélus: A synchronous language with ODEs. In *Hybrid Systems: Computation and Control*, pages 113–118. ACM, 2013.

[31] William E. Boyce and Richard C. DiPrima. *Elementary Differential Equations and Boundary Value Problems*. John Wiley and Sons, seventh edition, 2003.

[32] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 298–302. Springer, 1998.

[33] Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient implementation of a BDD package. In *Proceedings of the 27th ACM/IEEE design automation conference*, pages 40–45. ACM, 1991.

[34] Michael S. Branicky. *Studies in hybrid systems: Modeling, analysis, and control*. PhD thesis, Massachusetts Institute of Technology, 1995.

[35] Michael S. Branicky. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control*, 43(4): 475–482, 1998.

[36] Chad E. Brown. Satallax: An automatic higher-order prover. In *Automated Reasoning*, pages 111–117. Springer, 2012.

[37] Christopher W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *SIGSAM Bulletin*, 37(4):97–108, 2003.

[38] Victor Carreño and César Muñoz. Aircraft trajectory modeling and alerting algorithm verification. In *Theorem Proving in Higher Order Logics*, LNCS 1869. Springer, 2000.

[39] Rebekah Carter. Computation model of a rotary system with discontinuous elements. Master's thesis, University of Manchester, 2009.

[40] Rebekah Carter and Eva Navarro-López. Dynamically-driven timed automaton abstractions for proving liveness of continuous systems. In *Formal Modeling and Analysis of Timed Systems*, LNCS 7595, pages 59–74. Springer, 2012.

[41] Rebekah A. Carter. *Verification of Liveness Properties on Hybrid Dynamical Systems*. PhD thesis, University of Manchester, 2013.

[42] Ahmet C. Cem Say and Levent Akın. Sound and complete qualitative simulation is impossible. *Artificial Intelligence*, 149(2):251–266, 2003.

[43] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification*, pages 258–263. Springer, 2013.

[44] Chih-Hong Cheng, Harald Ruess, and Natarajan Shankar. JBernstein: A validity checker for generalized polynomial constraints. In *Computer Aided Verification*, pages 656–661. Springer, 2013.

[45] Alongkrit Chutinan and Bruce H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *Hybrid Systems: Computation and Control*, pages 76–90. Springer, 1999.

[46] Alongkrit Chutinan and Bruce H. Krogh. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control*, 48(1):64–75, 2003.

[47] Alessandro Cimatti, Edmund M. Clarke, et al. NuSMV 2: An OpenSource tool for symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification*, pages 359–364, 2002.

[48] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. A quantifier-free SMT encoding of non-linear hybrid automata. In *Formal Methods in Computer-Aided Design*, pages 187–195. IEEE, 2012.

[49] Edmund Clarke, Kenneth L. McMillan, Sérgio Campos, and Vasiliki Hartonas-Garmhausen. Symbolic model checking. In *Computer Aided Verification*, LNCS 1102, pages 419–422. Springer, 1996.

[50] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169. Springer, 2000.

[51] Edmund Clarke, Ansgar Fehnker, Zhi Han, Bruce Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *International Journal of Foundations of Computer Science*, 14(04):583–604, 2003.

[52] Edmund M. Clarke and E. Allen Emerson. *Design and synthesis of synchronization skeletons using branching time temporal logic*. Springer, 1982.

[53] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[54] Robert Clewley. Hybrid models and biological model reduction with PyDSTool. *PLoS computational biology*, 8(8):e1002628, 2012.

[55] Pieter Collins, Milad Niqui, Nathalie Revol, et al. A Taylor function calculus for hybrid system analysis: Validation in Coq. In *NSV-3: Third International Workshop on Numerical Software Verification.*, 2010.

[56] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[57] Judy Crow, Sam Owre, John Rushby, Natarajan Shankar, and Dave Stringer-Calvert. Evaluating, testing, and animating PVS specifications. Technical report, SRI International, Menlo Park, CA, 2001.

[58] Thao Dang and Romain Testylier. Reachability analysis for polynomial dynamical systems using the Bernstein expansion. *Reliable Computing*, 17(2):128–152, 2012.

[59] Thao Dang, Colas Le Guernic, and Oded Maler. Computing reachable states for nonlinear biological models. In *Computational Methods in Systems Biology*, pages 126–141. Springer, 2009.

[60] Thao Dang, Oded Maler, and Romain Testylier. Accurate hybridization of nonlinear systems. In *Hybrid Systems: Computation and Control*, pages 11–20. ACM, 2010.

[61] Marc Daumas, David Lester, and César Muñoz. Verified real number calculations: A library for interval arithmetic. *IEEE Transactions on Computers*, 58(2):226–237, February 2009.

[62] James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1–2):29–35, 1988.

[63] Hidde De Jong. Qualitative simulation and related approaches for the analysis of dynamic systems. *The Knowledge Engineering Review*, 19(2):93–132, 2004.

[64] Johan De Kleer and John Seely Brown. A qualitative physics based on confluences. *Artificial intelligence*, 24(1):7–83, 1984.

[65] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[66] William Denman. Verification of nonpolynomial systems using MetiTarski. In *Proceedings of the 19th Automated Reasoning Workshop*, pages 31–32. School of Computer Science, The University of Manchester, April 2012.

[67] William Denman. QUANTUM: Qualitative abstractions of non-polynomial models. In *Proceedings of the 27th International Workshop on Qualitative Reasoning*, pages 9–15, August 2013.

[68] William Denman. Verifying nonpolynomial hybrid systems by qualitative abstraction and automated theorem proving. In *NASA Formal Methods*, LNCS 8430, pages 203–208. Springer, April 2014.

[69] William Denman and César Muñoz. Automated real proving in PVS via MetiTarski. In *FM 2014: Formal Methods*, LNCS 8442, pages 194–199. Springer, May 2014.

[70] William Denman, Behzad Akbarpour, Sofiène Tahar, Mohamed H. Zaki, and Lawrence C. Paulson. Formal verification of analog designs using MetiTarski. In *Formal Methods in Computer-Aided Design*, pages 93–100. IEEE, November 2009.

[71] William Denman, Mohamed H. Zaki, and Sofiène Tahar. Formal verification of bond graph modelled analogue circuits. *Circuits, Devices and Systems, IET*, 5(3): 243–255, 2011.

[72] William Denman, Mohamed H. Zaki, Sofiène Tahar, and Luis Rodrigues. Towards flight control verification using automated theorem proving. In *NASA Formal Methods*, LNCS 6617, pages 89–100. Springer, April 2011.

[73] Akash Deshpande, Aleks Göllü, and Pravin Varaiya. SHIFT: A formalism and a programming language for dynamic networks of hybrid automata. In *Hybrid Systems IV*, LNCS 1273, pages 113–133. Springer, 1997.

[74] Ben Di Vito. A PVS prover strategy package for common manipulations. Technical Report TM-2002-211647, NASA Langley Research Center, 2002.

[75] Alexandre Donzé. Breach: A toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*, pages 167–170. Springer, 2010.

[76] Parasara Sridhar Duggirala, Le Wang, Sayan Mitra, Mahesh Viswanathan, and César Muñoz. Temporal precedence checking for switched models and its application to a parallel landing protocol. In *FM: Formal Methods*, pages 215–229, 2014.

[77] Bruno Dutertre and Leonardo De Moura. The YICES SMT solver. Technical report, SRI International, 2006.

[78] Andreas Eggers, Nacim Ramdani, Nedialko Nedialkov, and Martin Fränzle. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In *Software engineering and formal methods*, pages 172–187. Springer, 2011.

[79] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[80] Federal Aviation Administration. A system under stress: Aviation congestion. `http://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=8807`, 2007. Accessed: 25-08-2014.

[81] Michael Fisher, Louise Dennis, and Matt Webster. Verifying autonomous systems. *Communications of the ACM*, 56(9):84–93, September 2013.

[82] Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past hytech. In *Hybrid Systems: Computation and Control*, pages 258–273. Springer, 2005.

[83] 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 *Proceedings of the 23rd International Conference on Computer Aided Verification*, LNCS. Springer, 2011.

[84] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica modeling, simulation and software development environment. *Simulation News Europe*, 44(45):8–16, 2005.

[85] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *Automated Deduction–CADE-24*, pages 208–214. Springer, 2013.

[86] Sicun Gao, Soonho Kong, and Edmund M. Clarke. Satisfiability modulo odes. In *Formal Methods in Computer-Aided Design*, pages 105–112. IEEE, 2013.

[87] Sicun Gao, Soonho Kong, Wei Chen, and Edmund M. Clarke. δ-complete analysis for bounded reachability of hybrid systems. Technical report, Carnegie Mellon University, 2014.

[88] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.

[89] Rafal Goebel, Joao Hespanha, Andrew R. Teel, Chaohong Cai, and Ricardo Sanfelice. Hybrid systems generalized solutions and robust stability. In *Proceedings of the 6th IFAC symposium in nonlinear control systems*, 2004.

[90] Michael J. C. Gordon. HOL: A proof generating system for Higher-Order logic. *VLSI Specification, Verification and Synthesis*, 35:73–128, 1988.

[91] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with pvs. In *Computer Aided Verification*, pages 72–83. Springer, 1997.

[92] David F. Griffiths. *Numerical Methods for Ordinary Differential Equations*. Springer, 2010.

[93] Orna Grumberg and David E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994.

[94] Thomas C. Hales. A proof of the Kepler conjecture. *Annals of Mathematics*, pages 1065–1185, 2005.

[95] Thomas C. Hales. Introduction to the Flyspeck project. In *Mathematics, Algorithms, Proofs*, Dagstuhl Seminar Proceedings 5021. Internationales Begegnungs und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. `http://drops.dagstuhl.de/opus/volltexte/2006/432`.

[96] Thomas C. Hales, John Harrison, Sean McLaughlin, Tobias Nipkow, Steven Obua, and Roland Zumkeller. A revision of the proof of the Kepler conjecture. In *The Kepler Conjecture*, pages 341–376. Springer, 2011.

[97] John M. Hammersley. Monte carlo methods for solving multivariable problems. *Annals of the New York Academy of Sciences*, 86(3):844–874, 1960.

[98] Younghun Han, Sungwon Kang, and Jinhyun Kim. ExCHARON: Improved modeling language for cyber-physical systems based on CHARON. In *IEEE Conference on Computational Science and Engineering*, pages 734–741. IEEE, 2013.

[99] Ruth Hardy. *Formal methods for control engineering: A validated decision procedure for Nichols Plot analysis*. PhD thesis, School of Computer Science, University of St. Andrews, February 2006.

[100] Terry Hardy. Case studies in process safety: Lessons learned from software-related accidents. In *Proceedings of the 9th Global Congress on Process Safety*. American Institute of Chemical Engineers, May 2013.

[101] David Harel and Amir Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI Series*, pages 477–498. Springer, 1985.

[102] William H. Harman. TCAS: A system for preventing midair collisions. *The Lincoln Laboratory Journal*, 2:437–458, 1989.

[103] John Harrison. Formal verification at Intel. In *IEEE Symposium on Logic in Computer Science*, pages 45–54. IEEE, 2003.

[104] Thomas A. Henzinger. Masaccio: A formal model for embedded components. In *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics*, pages 549–563. Springer, 2000.

[105] Thomas A. Henzinger and Pei-Hsin Ho. Algorithmic analysis of nonlinear hybrid systems. In *Computer Aided Verification*, pages 225–238. Springer, 1995.

[106] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th ACM Symposium on Theory of Computing*, pages 373–382. ACM, 1995.

[107] Thomas A. Henzinger, Benjamin Horowitz, Rupak Majumdar, and Howard Wong-Toi. Beyond HYTECH: Hybrid systems analysis using interval numerical methods. In *Hybrid Systems: Computation and Control*, pages 130–144. Springer, 2000.

[108] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Software verification with BLAST. In *Model Checking Software*, pages 235–239. Springer, 2003.

[109] Thomas R. Hinrichs, Kenneth D. Forbus, Johan de Kleer, and Sungwook Yoon. Hybrid qualitative simulation of military operations. In *AAAI Conference on Artificial Intelligence*, pages 1655–1661, 2011.

[110] Allen T. Hjelmfelt Jr. Chaotic behavior of particle on vibrating plate. *Journal of Engineering Mechanics*, 115(7):1458–1471, 1989.

[111] Gerard J. Holzmann. *The SPIN model checker: Primer and reference manual*. Addison-wesley, 2003.

[112] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *Hybrid Systems: Computation and Control*, pages 160–173. Springer, 2000.

[113] Zongyan Huang, Matthew England, David Wilson, James H. Davenport, Lawrence C. Paulson, and James Bridge. Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In *Intelligent Computer Mathematics*, LNCS 8543, pages 92–107, 2014.

[114] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. In *Design and Application of Strategies/Tactics in Higher Order Logics*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, 2003.

[115] Fabian Immler. Formally verified computation of enclosures of solutions of ordinary differential equations. In *NASA Formal Methods*, LNCS 8430, pages 113–127. Springer, 2014.

[116] Daisuke Ishii, Kazunori Ueda, Hiroshi Hosobe, and Alexandre Goldsztejn. Interval-based solving of hybrid constraint systems. In *Analysis and Design of Hybrid Systems*, pages 144–149, 2009.

[117] Daisuke Ishii, Kazunori Ueda, and Hiroshi Hosobe. An interval-based SAT modulo ODE solver for model checking nonlinear hybrid systems. *International Journal on Software Tools for Technology Transfer*, 13(5):449–461, October 2011.

[118] Paul Jackson, Andrew Sogokon, James Bridge, and Lawrence Paulson. Verifying hybrid systems involving transcendental functions. In *NASA Formal Methods*, pages 188–202. Springer, 2014.

[119] Karl Henrik Johansson, Magnus Egerstedt, John Lygeros, and Shankar Sastry. On the regularization of zeno hybrid automata. *Systems and Control Letters*, 38(3): 141–150, 1999.

[120] Taylor T. Johnson, Jeremy Green, Sayan Mitra, Rachel Dudley, and Richard Scott Erwin. Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems. In *FM 2012: Formal Methods*, pages 252–266. Springer, 2012.

[121] Dejan Jovanovic and Leonardo De Moura. Solving non-linear arithmetic. In *Automated Reasoning - 6th International Joint Conference*, LNCS 7364, pages 339–354. Springer, 2012.

[122] Herbert Kay and Benjamin Kuipers. Numerical behavior envelopes for qualitative models. In *Proceedings of the eleventh national conference on Artificial intelligence*, pages 606–613. AAAI Press, 1993.

[123] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, third edition, 2002.

[124] Matthew Klenk, Daniel G. Bobrow, Johan de Kleer, John Hanley, and Bill Janssen. Placing qualitative reasoning in the design process. In *Proceedings of the 26th International Workshop on Qualitative Reasoning*, pages 16–18, 2012.

[125] Matthew Klenk, Johan de Kleer, Daniel G. Bobrow, and Bill Janssen. Using Modelica models for qualitative reasoning. In *Proceedings of the 27th International Workshop on Qualitative Reasoning*, page 1, 2013.

[126] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[127] Benjamin Kuipers. Qualitative simulation. *Artificial intelligence*, 29(3):289–338, 1986.

[128] Benjamin Kuipers. *Qualitative reasoning: Modeling and simulation with incomplete knowledge*. MIT press, 1994.

[129] Benjamin Kuipers and Daniel Berleant. Using incomplete quantitative knowledge in qualitative reasoning. In *AAAI Conference on Artificial Intelligence*, volume 88, pages 324–329, 1988.

[130] Colas Le Guernic. *Reachability analysis of hybrid systems with linear continuous dynamics*. PhD thesis, Universite Grenoble 1, 2009.

[131] Colas Le Guernic and Antoine Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, 2010.

[132] Daniel Liberzon. *Switching in systems and control*. Springer, 2003.

[133] Youdong Lin and Mark A. Stadtherr. Validated solutions of initial value problems for parametric ODEs. *Applied Numerical Mathematics*, 57(10):1145–1162, 2007.

[134] Thorsten Luettel, Michael Himmelsbach, and Hans-Joachim Wuensche. Autonomous ground vehicles – Concepts and a path to the future. *Proceedings of the IEEE*, 100:1831–1839, May 2012.

[135] John Lygeros, Karl H. Johansson, Slobodan N. Simic, Jun Zhang, and Shankar Sastry. Dynamical properties of hybrid automata. *IEEE Transactions on Automatic Control*, 48(1):2–17, Jan 2003.

[136] Nancy Lynch, Roberto Segala, Frits Vaandrager, and Henri B Weinberg. Hybrid I/O automata. In *Hybrid Systems III*, LNCS 1066, pages 496–510. Springer, 1996.

[137] Kyoko Makino and Martin Berz. COSY INFINITY version 9. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 558(1):346–350, 2006.

[138] Oded Maler. Algorithmic verification of continuous and hybrid systems. In *Proceedings of the 15th Workshop on the Verification of Infinite-State Systems*, EPTCS 140, pages 48–69, 2013.

[139] Oded Maler and Grégory Batt. Approximating continuous systems by timed automata. In *Formal methods in systems biology*, LNBI 5054, pages 77–89. Springer, 2008.

[140] Paolo Masci, Yi Zhang, Paul Jones, Patric Oladimeji, Enrico D'Urso, Cinzia Bernardeschi, Paul Curzon, and Harold Thimbleby. Combining PVSio with Stateflow. In *NASA Formal Methods*, volume 8430 of *Lecture Notes in Computer Science*, pages 209–214. Springer, 2014.

[141] Tom Melham. *Higher Order Logic and Hardware Verification*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1993.

[142] Jia Meng, Lawrence C. Paulson, and Gerwin Klein. A termination checker for Isabelle Hoare logic. In *International Verification Workshop - VERIFY*, pages 104–118, 2007.

[143] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[144] Sayan Mitra, Yong Wang, Nancy Lynch, and Eric Feron. Safety verification of model helicopter controller using hybrid Input/Output automata. In *Hybrid Systems: Computation and Control*, pages 343–358. Springer, 2003.

[145] Ramon E. Moore. *Interval analysis*. Prentice-Hall, 1966.

[146] Sergio Mover, Alessandro Cimatti, Ashish Tiwari, and Stefano Tonetta. Time-aware relational abstractions for hybrid systems. In *International Conference on Embedded Software*, page 14. IEEE Press, 2013.

[147] César Munoz. Rapid prototyping in PVS. Technical Report NASA/CR-2003-212418, National Institue of Aerospace, 2003. NIA Report No. 2003-03.

[148] César Muñoz and Micaela Mayero. Real automation in the field. Technical Report CR-2001-211271, NASA Langley Research Center/ICASE, December 2001.

[149] César Muñoz and Anthony Narkawicz. Formalization of a representation of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 51(2):151–196, 2013.

[150] César Muñoz and Anthony Narkawicz. Formalization of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 51(2): 151–196, 2013.

[151] César Muñoz, Víctor Carreño, Gilles Dowek, and Ricky Butler. Formal verification of conflict detection algorithms. *International Journal on Software Tools for Technology Transfer*, 4(3):371–380, 2003.

[152] Katta G. Murty and Santosh N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical programming*, 39(2):117–129, 1987.

[153] Anthony Narkawicz and César Muñoz. A formally verified generic branching algorithm for global optimization. In *Fifth Working Conference on Verified Software: Theories, Tools and Experiments*, LNCS 8164, pages 326–343. Springer, 2014.

[154] Anthony Narkawicz and César Muñoz. A formally-verified decision procedure for univariate polynomial computation based on Sturm's theorem. Unpublished Manuscript, 2014.

[155] Anthony Narkawicz, César Muñoz, Heber Herencia-Zapana, and George Hagen. Formal verification of lateral and temporal safety buffers for state-based conflict detection. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of aerospace engineering*, 227(9):1412–1424, 2013.

[156] Eva M. Navarro-López and Rebekah Carter. Hybrid automata: An insight into the discrete abstraction of discontinuous systems. *International Journal of Systems Science*, 42(11):1883–1898, 2011.

[157] Eva M Navarro-Lopez and Dina Shona Laila. Group and total dissipativity and stability of multi-equilibria hybrid automata. *Automatic Control, IEEE Transactions on*, 58(12):3196–3202, 2013.

[158] Nedialko S. Nedialkov. VNODE-LP: A validated solver for initial value problems in ordinary differential equations. Technical Report CAS-06-06-NN, Department of Computing and Software, McMaster University, 2006. `http://www.cas.mcmaster.ca/~nedialk/vnodelp/`.

[159] Sam Owre. Random testing in PVS. In *Workshop on Automated Formal Methods*, 2006.

[160] Sam Owre, John M. Rushby, and Natarajan Shankar. PVS: A prototype verification system. In *Automated Deduction: CADE-11*, pages 748–752. Springer, 1992.

[161] Antonis Papachristodoulou and Stephen Prajna. Analysis of non-polynomial systems using the sum of squares decomposition. In *Positive Polynomials in Control*, LNCIS 312, pages 23–43. Springer, 2005.

[162] Antonis Papachristodoulou, Stephen Prajna, Peter Seiler, and Pablo A. Parrilo. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*. `http://arxiv.org/abs/1310.4716`, 2013.

[163] Pablo A. Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, 2000. `http://thesis.library.caltech.edu/1647/`.

[164] Grant Olney Passmore and Paul B. Jackson. Combined decision techniques for the existential theory of the reals. In *Intelligent Computer Mathematics*, pages 122–137. Springer, 2009.

[165] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828. Springer, 1994.

[166] Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In *IWIL-2010*, 2010.

[167] Raleigh B. Perry, Michael M. Madden, Wilfredo Torres-Pomales, and Ricky W. Butler. The simplified aircraft-based paired approach with the ALAS alerting algorithm. Technical report, NASA, 2013. TM-2013-217804.

[168] Andre Platzer. *Logical Analysis of Hybrid Systems*. Springer, 2010.

[169] André Platzer and Edmund M Clarke. Computing differential invariants of hybrid systems as fixedpoints. In *Computer Aided Verification*, pages 176–189. Springer, 2008.

[170] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In *FM 2009: Formal Methods*, LNCS 5850, pages 547–562. Springer, 2009.

[171] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In *Automated Reasoning*, LNCS 5195, pages 171–178. Springer, 2008.

[172] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[173] Stephen Prajna. Barrier certificates for nonlinear model validation. *Automatica*, 42(1):117–126, 2006.

[174] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control*, LNCS 2993, pages 477–492. Springer, 2004.

[175] Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. `www.ptolemy.org`, 2014. URL `http://ptolemy.org/books/Systems`.

[176] Sreeranga Rajan, Natarajan Shankar, and Mandayam K. Srivas. An integration of model checking with automated proof checking. In *Computer Aided Verification*, pages 84–97. Springer, 1995.

[177] Stefan Ratschan and Zhikun She. Constraints for continuous reachability in the verification of hybrid systems. In *Artificial Intelligence and Symbolic Computation*, pages 196–210. Springer, 2006.

[178] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Transactions on Embedded Computing Systems*, 6(1):8, 2007.

[179] Nathalie Revol, Kyoko Makino, and Martin Berz. Taylor models and floating-point arithmetic: Proof that arithmetic operations are validated in COSY. *The Journal of Logic and Algebraic Programming*, 64(1):135–154, 2005.

[180] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI communications*, 15(2):91–110, 2002.

[181] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[182] Paul Roosens. Congestion and air transport: A challenging phenomenon. *European Journal of Transport and Infrastructure Research*, 8(2):137–146, 2008.

[183] Kristin Y. Rozier. Linear temporal logic symbolic model checking. *Computer Science Review*, 5(2):163–203, 2011.

[184] John M. Rushby. An evidential tool bus. In *Proceedings of the International Conference on Formal Engineering Methods*, LNCS 3785, page 36, 2005.

[185] Elisha Sacks. *Automatic Qualitative Analysis of Ordinary Differential Equations Using Piecewise Linear Approximations*. PhD thesis, Massachusetts Institute of Technology, 1988.

[186] Ricardo Sanfelice, David Copp, and Pablo Nanez. A toolbox for simulation of hybrid systems in MATLAB/Simulink: Hybrid equations (HyEQ) toolbox. In *Hybrid Systems: Computation and Control*, pages 101–106. ACM, 2013.

[187] Sriram Sankaranarayanan and Ashish Tiwari. Relational abstractions for continuous and hybrid systems. In *Computer Aided Verification*, pages 686–702. Springer, 2011.

[188] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In *Verification, Model Checking, and Abstract Interpretation*, pages 25–41. Springer, 2005.

[189] Sriram Sankaranarayanan, Thao Dang, and Franjo Ivancic. Symbolic model checking of hybrid systems using template polyhedra. In *Tools and algorithms for the construction and analysis of systems*, pages 188–202. Springer, 2008.

[190] Shankar Sastry. *Nonlinear Systems*. Springer, first edition, 1999.

[191] Michael A. Savageau and Eberhard O. Voit. Recasting nonlinear differential equations as S-systems: A canonical nonlinear form. *Mathematical biosciences*, 87(1): 83–115, 1987.

[192] Stephan Schulz. System Description: E 1.8. In *Proceedings of the 19th LPAR, Stellenbosch*, LNCS 8312, pages 735–743. Springer, 2013.

[193] Peter Sheppard. Autonomous vehicles: A railway perspective. The Institution of Engineering and Technology (IET) Sector Insights. `http://www.theiet.org/sectors/transport/documents/av-railway.cfm?type=pdf`, December 2013. Accessed: 25-08-2014.

[194] Benjamin Shults and Benjamin Kuipers. Proving properties of continuous systems: Qualitative simulation and temporal logic. *Artificial Intelligence*, 92(1):91–129, 1997.

[195] Reid Simmons, Charles Pecheur, and Grama Srinivasan. Towards automatic verification of autonomous systems. In *IEEE/RSJ International conference on Intelligent Robots and Systems*, pages 1410–1415, 2000.

[196] Christoffer Sloth and Rafael Wisniewski. Verification of continuous dynamical systems by timed automata. *Formal Methods in System Design*, 39(1):47–82, August 2011.

[197] Christoffer Sloth, George J. Pappas, and Rafael Wisniewski. Compositional safety analysis using barrier certificates. In *Hybrid Systems: Computation and Control*, pages 15–24. ACM, 2012.

[198] Christopher Sloth and Rafael Wisniewski. Abstraction of continuous dynamical systems utilizing Lyapunov functions. In *Proceedings of the 49th IEEE Conference on Decision and Control*, pages 3760–3765, December 2010.

[199] Oleg Sokolsky and Hyoung Seok Hong. Qualitative modeling of hybrid systems. In *Proceedings of the Monterey Workshop on Engineering Automation for Computer Based Systems*, 2001. `http://repository.upenn.edu/cis_papers/87`.

[200] Jeremy Sproston. Decidable model checking of probabilistic hybrid automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 31–45. Springer, 2000.

[201] Olaf Stursberg, Stefan Kowalewski, Ingo Hoffmann, and Jörg Preußig. Comparing timed and hybrid automata as approximations of continuous systems. In *Hybrid Systems IV*, pages 361–377. Springer, 1997.

[202] Nik Sultana, Jasmin Christian Blanchette, and Lawrence C. Paulson. LEO-II and Satallax on the Sledgehammer test bench. *Journal of Applied Logic*, 11(1):91–102, 2013.

[203] Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. The tptp problem library. In *Automated Deduction: CADE-12*, pages 252–266. Springer, 1994.

[204] Paulo Tabuada. *Verification and control of hybrid systems: A symbolic approach*. Springer, 2009.

[205] Alfred Tarski. A decision method for elementary algebra and geometry. Technical report, RAND Corp., 1948.

[206] Lucio Tavernini. Differential automata and their discrete simulators. *Nonlinear Analysis: Theory, Methods & Applications*, 11(6):665–683, 1987.

[207] Romain Testylier and Thao Dang. NLTOOLBOX: A library for reachability computation of nonlinear dynamical systems. In *Automated Technology for Verification and Analysis*, pages 469–473. Springer, 2013.

[208] Ashish Tiwari. HybridSAL: Modeling and abstracting hybrid systems. Technical report, Computer Science Laboratory, SRI International, 2003.

[209] Ashish Tiwari. Abstractions for hybrid systems. *Formal Methods in System Design*, 32(1):57–83, February 2008.

[210] Ashish Tiwari. HybridSAL relational abstracter. In *Computer Aided Verification*, pages 725–731. Springer, 2012.

[211] Ashish Tiwari and Gaurav Khanna. Series of abstractions for hybrid automata. In *Hybrid Systems: Computation and Control*, LNCS 2289, pages 465–478. Springer, March 2002.

[212] Claire Tomlin, George J. Pappas, and Shankar Sastry. Conflict resolution for air traffic management: A study in multiagent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, 1998.

[213] Jeffrey S. Urbach and Jeffrey S. Olafsen. Spheres on a vibrating plate: Clustering and collapse. In *Proceedings of the 5th Experimental Chaos Conference*, pages 365–376. World Scientific Publications, 2001.

[214] A. J. van der Schaft and J. M. Schumacher. *Introduction to Hybrid Dynamical Systems*. Springer-Verlag, London, UK, 1999. ISBN 1852332336.

[215] Pravin Varaiya. Reach set computation using optimal control. In *Verification of Digital and Hybrid Systems*, pages 323–331. Springer, 2000.

[216] Christoph Weidenbach. SPASS version 3.5. In *Automated Deduction: CADE-22*, pages 140–145. Springer, 2009.

[217] Lee C. Yang and James K. Kuchar. Prototype conflict alerting system for free flight. *Journal of Guidance, Control, and Dynamics*, 20(4):768–773, 1997.

[218] Natalia Zelazna, Andrzej Krajniak, Hubert Wagner, and Piotr Brendel. Computer assisted proofs in dyanmics, 2014. `http://capd.ii.uj.edu.pl`.

[219] Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Moritz Hahn. Safety verification for probabilistic hybrid systems. *European Journal of Control*, 18(6):572–587, 2012.