Projectagon-Based Reachability Analysis for Circuit-Level Formal Verification

by

Chao Yan

B.Sc., Peking University, 2003

M.Sc., The University of British Columbia, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University Of British Columbia

(Vancouver)

September 2011

Abstract

This dissertation presents a novel verification technique for analog and mixed signal circuits. Analog circuits are widely used in many applications include consumer electronics, telecommunications, medical electronics. Furthermore, in deep sub-micron design, physical effects might undermine common digital abstractions of circuit behavior. Therefore, it is necessary to develop systematic methodologies to formally verify hardware design using circuit-level models.

We present a formal method for circuit-level verification. Our approach is based on translating verification problems to reachability analysis problems. It applies nonlinear ODEs to model circuit dynamics using modified nodal analysis. Forward reachable regions are computed from given initial states to explore all possible circuit behaviors. Analog properties are checked on all circuit states to ensure full correctness or find a design flaw. Our specification language extends LTL logic with continuous time and values and applies Brockett's annuli to specify analog signals. We also introduced probability into the specification to support practical analog properties such as metastability behavior.

We developed and implemented a reachability analysis tool COHO for a simple class of moderate-dimensional hybrid systems with nonlinear ODE dynamics. COHO employs *projectagons* to represent and manipulate moderate-dimensional, non-convex reachable regions. COHO solves nonlinear ODEs by conservatively approximating ODEs as linear differential inclusions. COHO is robust and efficient. It uses arbitrary precision rational numbers to implement exact computation and trims projectagons to remove infeasible regions. To improve performance and reduce error, several techniques are developed, including a guess-verify strategy, hybrid computation, approximate algorithms, and so on.

The correctness and efficiency of our methods have been demonstrated by the success of verifying several circuits, including a toggle circuit, a flip-flop circuit, an arbiter circuit, and a ring-oscillator circuit proposed by researchers from Rambus Inc. Several important properties of these circuits have been verified and a design flaw was spotted during the toggle verification. During the reachability computation, we recognized new problems (*e.g.*, stiffness) and proposed our solutions to these problems. We also developed new methods to analyze complex properties such as metastable behaviors. The combination of these methods and reachability analysis is capable of verifying practical circuits.

Table of Contents

List of Tables

List of Figures

Abbreviations

Abbreviations	Full Names	Definitions
ACTL	A Universal Fragment of CTL	page 21
AnaCTL	Analog CTL	page 21
AMS	Analog and Mixed Signal	page 1
APR	Arbitrary Precision Rational	page 93
ASL	Analog Specification Language	page 22
BDD	Binary Decision Diagram	page 27
BLF	Biquad Lowpass Filter	page 43
CDD	Clock Difference Diagram	page 27
CTL	Computation Tree Logic	page 20
CTL-AT	Analog and Timed CTL	page 21
DBM	Difference Bound Matrix	page 27
$\Delta\Sigma$M	$\Delta - \Sigma$ Modulator	page 37
DTTS	Discrete-Trace Transition System	page 20
FIFO	First In First Out	page 3
HA	Hybrid Automaton	page 18
HIOA	Hybrid Input Output Automaton	page 19
ICTL	Integrator CTL	page 21
LDHA	Linear Dynamical Hybrid Automaton	page 19
LDI	Linear Differential Inclusion	page 78
LHA	Linear Hybrid Automaton	page 19
LHPN	Labeled Hybrid Petri Net	page 19
LP	Linear Programming or Linear Program	page 100

LTL	Linear Time Temporal Logic	page 20
MITL	Metric Interval Temporal Logic	page 21
NHA	Nonlinear Hybrid Automaton	page 19
ODE	Ordinary Differential Equation	page 28
ODI	Ordinary Differential Inclusion	page 28
ORH	Oriented Rectangular Hull	page 26
PDE	Partial Differential Equation	page 32
PIHA	Polyhedral Invariant Hybrid Automaton	page 20
PLL	Phase-Locked Loop	page 3
PSL	Accellera Property Specification Language	page 20
PVT	Process Voltage Temperature	page 60
RRO	Rambus Ring Oscillator	page 54
RTCTL	Real Time CTL	page 21
SAV	Simulation Aid Verification	page 20
SMT	Satisfiability Modulo Theory	page 36
SRAM	Static Random Access Memeory	page 6
SRE	System of Recurrence Equations	page 16
STL	Signal Temporal Logic	page 21
TA	Timed Automaton	page 18
TCTL	Timed CTL	page 21
TDO	Tunnel Diode Oscillator	page 39
THPN	Timed Hybrid Petri Net	page 19
VCO	Voltage Controlled Oscillator	page 41

Acknowledgments

First and foremost, I must acknowledge Dr. Mark Greenstreet for his guidance, kindness, and patience during this work. I feel very fortunate to have been mentored by a supervisor with such keen insight. I would like to thank the members of my supervisory committee: Alan Hu and Ian Mitchell. I would also like to thank the members of my examining committee: William Evans, Eldad Haber, and Warren Hunt. Without their contribution and direction, this thesis would not have been what it is now.

I would not have developed a solution for a new research topic without many helpful discussions with experts from different areas and experiences from both academic and industry. Many thanks to Kevin Jones, Kathryn Mossawir, Tom Sheffler and the verification group in Rambus Inc for the industry experience and providing practical verification problems. This thesis also profits from the collaboration with members of the TIMA lab, France: Laurent Fesquet, Florent Ouchet, and Katell Morin-Allory. Finally, I am very grateful to Will Evans and Chen Grief at UBC, Oded Maler, Thao Dang as well as Goran Frehse at IMAG, Chris Myers at University of Utah, David Dill at Standford University, Bruce Krogh at CMU, and Haralampos Stratigopoulos at TIMA for their very helpful insights on reachability analysis, analog design, and numerical computation.

To my parents, for their endless support and patience.

1

Introduction

1.1 Background and Motivation

Computing technology permeates nearly all aspects of modern life, from desktop and laptop computers through cellphones and embedded computing devices in everything from automobiles and consumer appliances to life saving medical equipments. Continuing advances in these products relies on the successful design of new integrated circuits with ever increasing capabilities. The design process for these chips has become extremely complicated due to the large number of transistors (now well over a billion) on a single chip and the increasing use of combined digital and analog circuits on the same chip. A single error in a design can be extremely costly to correct, requiring design changes, making new masks, and fabricating new chips. The delay in time-to-market from such errors can cause a project to fail. Thus, there is a large need for better design verification techniques that can be used before a chip is fabricated.

This thesis develops new methods for circuit verification. Verification at the circuit-level is import for several reasons. First, *analog and mixed signal* (AMS) circuits are widely used in electronic devices, *e.g.*, cellphones, GPS, and DSPs. Second, physical effects affect transistor behavior in deep submicron processes designs; therefore, low-level phenomena (*e.g.*, leakage currents) must be considered even for digital circuits. Third, circuit-level bugs account for a growing percentage of critical bugs in real circuits. Digital design has become a relatively low error

process because there are systematic specifications, design flows, and test methodologies using gate and higher level abstractions. However, AMS designs rely on designers' intuition and expertise and lack a systematic validation flow. Furthermore, circuit-level bugs generally require re-spins and are expensive to fix. For example, Intel discovered a design flaw in the 6-Series chipset, which is code-named Cougar Point, and is used in systems with Sandy Bridge processors [7]. The SATA (Serial-ATA) ports within the chipset are susceptible to degradation over time, which could impact performance or functionality of storage devices such as hard drives. The problem in the chipset was traced back to a transistor in the 3Gbps PLL clocking tree. The transistor has a very thin gate oxide to turn it on with a very low voltage. However, the leakage current of the transistor is higher than expected because the transistor is biased with too high of a voltage. The leakage current can increase over time and cause bit errors on a SATA link. Transfers retry if there is an error which degrades the performance and results in failure on the 3Gbps ports. The transistor is a vestige of an earlier design retained in an engineering oversight, and it can be completely disabled without any ill effect. However, to disable the transistor, the entire chip set (or motherboard) has to be replaced. This design flaw has lead to a recall with an estimated cost of about one billion to repair and replace affected materials and systems in the market. Furthermore, the delay of the widely anticipated Sandy Bridge processors has a significant effect on sales for major hardware vendors, *e.g.,* Apple and its MacBook Pro, and also on sales of software, *e.g.,* Windows.

Simulation is the most widely used method to validate both digital and analog circuits. This is because simulation can find errors (especially trivial bugs) quickly, and the simulation results make sense with respect to designer intuition, and thus can help to identify the sources of bugs. However, simulation based methods have several limitations. First, simulation only provides incomplete coverage and cannot guarantee the correctness of the circuit. Simulation only covers some input signals, incomplete states and a limited number of operating conditions. Therefore, fabricated chips may fail to work even if the circuit passed all simulations before tape-out. For example, the 6-series chipset described above passed all of Intel's internal qualification tests as well as all of the OEM qualification tests. These tests include functionality, reliability and behavior at various conditions, such as high-

/low temperature, and high/low voltage. However, the simulation coverage was still not high enough to find the degradation bug. This is especially true for deep sub-micron process designs and AMS designs as the number of corner cases is huge. For example, the PLL circuit designed in [205] has three feedback paths: an analog proportional path, a digital integral path, and an additional software control-loop. The digital part can be precisely controlled by hundreds of inputs from the software part. It is impossible to simulate all combinations of the control signals. Second, simulation is often based on highly abstracted models and ideal conditions, which might be unverified especially for analog circuits. Therefore, circuit-level bugs, such as wiring errors and simple parametric faults (wire resistance too high, too much cross-talk, *etc.*), can go undetected even if the simulations (with the abstract model and ideal condition) were exhaustive! Designers have to use these abstractions and assumptions; otherwise, the simulation is too slow (typically several weeks or longer [215]). For example, it is generally very expensive to simulate the start-up behavior of analog circuits because the start-up time is too long. Therefore, simulations are typically performed from user-specified, ideal initial states. However, these assumption are not checked and might cause re-spins of chips. Take a ring-oscillator from Rambus Inc as an example [129]. Researchers reported that the circuit failed to start to oscillate in fabricated chips. The bug eluded detection because all initial states used in simulations were in the oscillation orbit. Furthermore, it is extremely difficult for designers to find appropriate parameters of simulations to expose circuit-level bugs. For example, Greenstreet designed a FIFO circuit based on the C-element circuit [82, Chapter 4.4]. An analog timing race problem was found in the fabricated chip: leakage caused a signal to drop slightly below the threshold voltage of PMOS transistors. This led to unintended oscillations that prevented the FIFO from being initialized properly. Similar to showing correct start-up, it is also important to show that analog circuits can make mode transitions properly. Such transitions occur, for example, when a CPU changes its operating voltage and frequency. Other AMS circuits can include updates of digital control values several times per second or more to track changes in operating conditions. These transitions bring the circuit temporarily out of its intended operating range, but the simulation time to verify that the circuit correctly settles at the intended operating point may be prohibitive. Again, alternatives to

simulation based validation are needed. Because of these limitations of simulation based methods, it is necessary to develop formal techniques for circuit-level verification.

Formal verification conservatively models a design, specifies correct behaviors, and automatically determines if all possible behaviors of the model are correct. Formal methods which employ gate-level models have been well-studied and applied in industry, such as equivalence checking, model checking and theorem proving. For example, STE (symbolic trajectory evaluation) has been used in Intel for several years [182]. The success of digital formal verification motivates the work of extending formal methods to the continuous domain.

However, there are several new challenges of *circuit-level formal verification*. First of all, formal methods require specification languages to describe the behavior of circuits and properties to be verified. Precise specifications are not obvious in traditional analog design practice. For analog circuits, many interacting physical effects and details must be considered. Currently, the work of analog design is largely an art: highly dependent on intuition and experience. Furthermore, analog circuits are often designed to work in a particular context and lack precise descriptions. It is challenging to extend specification methods for digital formal methods to analog circuits. While temporal logics have been very successful for formally specifying properties of digital designs; most such logics are based on a discrete notion of time and discrete states. However, analog properties require continuous time and states to be described in the specification. There are also many properties which are difficult to express by current methods, especially many properties of interest are not time-domain properties, *e.g.*, frequency, and transfer functions. As another example, the properties of circuits with metastable behaviours cannot be expressed by most current specification methods because they do not support probability which are necessary for specifying metastable behaviors.

Another new issue is to develop novel verification techniques. Circuit-level models are generally described by nonlinear ordinary differential equations, which in general do not have closed form solutions. Numerical methods must be applied to solve complex dynamics. Therefore, it is unlikely that symbolic methods can produce accurate results with general models. It is also very expensive to solve nonlinear ODEs using numerical methods, thus, efficient ODE solvers are neces-

sary. To make the verification sound, over-approximated results are required which exclude most available numerical integrators. Furthermore, state explosion becomes a problem in representing and manipulating moderate- to high-dimensional continuous space. Typical analog circuits have tens of (or more) nodes which corresponds to phase spaces with tens of dimensions. However, current representation methods have either expensive operators (*e.g.*, polytopes) or large approximation errors (*e.g.*, hyper-rectangles), and thus are not capable of representing moderate-dimensional regions efficiently. A new challenge is that the regions for all possible circuit states are generally non-convex. For example, different converging rates often lead trajectories to hyperbolic (*i.e.*, "banana-like") shapes. This makes it difficult to develop an efficient representation method.

Because of these challenges, most prior results in circuit verification have been either low-dimensional (often two-dimensional, never more than four-dimensional) or unrealistically simple models (linear or quasilinear). For example, the well-studied timed automata model [16] is too abstracted to model circuit-level behavior of interest to designers. Several simple circuits have been studied, such as $\Delta - \Sigma$ modulator [50], tunnel diode oscillator [97], voltage controlled oscillator [73], biquad lowpass filter [97]. However, verified properties based on simple models can be checked trivially by paper-and-pencil or several simulations. More details are given in Chapter 2. Therefore, current methods cannot be applied to verify most properties of interest of practical circuits.

However, there is urgent need for CAD tools that can find design flaws of AMS circuits automatically during hardware development. For example, the Pentium IV processor used self-resetting domino circuits to implement a fast ALU which completes one ALU operation in half a clock cycle [120]. However, manual checking of the ALU functionality was required every time any changes were made to the design, because current CAD tools do not support domino circuits. This delayed the release of the Pentium IV processor and the novel techniques were not used in the next generation products.

Figure 1.1 illustrates the motivation of this thesis. As described above, current formal methods can only be applied to verify very simple properties of small circuits, such as simple oscillators and filters. However, circuit designers are interested in important properties of practical, complex circuits, such as PLLs [205],

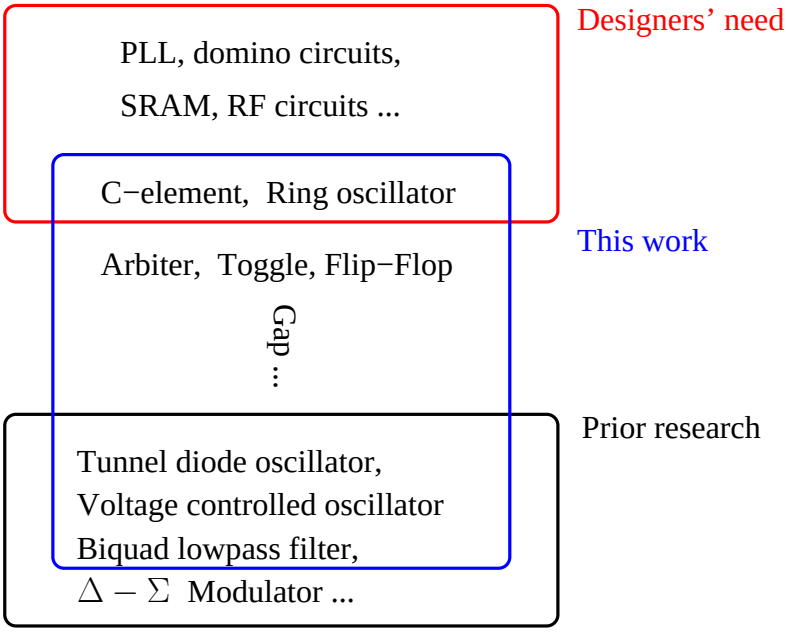


Figure 1.1: Motivation

SRAMs, self-resetting domino circuits, and RF circuits. There is a large gap between the simple examples of prior work and the verification need of designers. Our goal is to bridge the gap and verify some practical circuits. In particular, I present verifications of an arbiter, a toggle circuit, a C-element, and the Rambus ring oscillator circuit.

Our solution to the circuit-level formal verification problem is based on reachability analysis, which can be viewed as model checking in continuous domains. The method models a circuit as a hybrid system, computes all reachable states by solving the discrete as well as the continuous dynamics, and then validates that the circuit's specification holds for all reachable states. We designed a specification language to express analog properties, developed a new representation method and an efficient algorithm to bound solutions of ODEs. These techniques enable us to develop a general verification flow for AMS circuits which has been applied to several practical circuits.

1.2 Problem Statement

Circuit-level verification is necessary to spot critical bugs before fabrication for both AMS designs and deep sub-micron designs. Extending digital formal methods to continuous domains requires novel techniques for modeling circuits, specifying analog signals and desired properties, and solving non-linear dynamics to compute circuit states.

Reachability analysis is a promising method for formal verification using circuit-level models. To verify significant properties of large circuits, it is important to develop efficient and accurate methods to support moderate-dimensional (*e.g.,* 5-20) systems with highly non-linear dynamics and non-convex reachable regions.

1.3 Contributions

This thesis demonstrates the feasibility of formally verifying circuit behaviors for circuits modeled by non-linear, ordinary differential equations. This verification is performed using projectagon-based reachability analysis.

In particular, this thesis explores reachability analysis techniques and provides a reachability computation tool COHO for formal verification of digital or analog circuits. This thesis makes contributions in the following areas:

- We proposed techniques for modeling and specifying analog circuits and their behaviors, which make it possible to perform circuit verification through reachability analysis.
  - We developed a method to model a circuit as a system of non-linear differential equations (ODEs) automatically. Transistors are modeled using a simple, table-based method, and other devices can be supported similarly.
  - We applied Brockett's annulus to specify a family of analog signals. Based on it, we presented an extended LTL logic that supports dense time and continuous state to specify analog properties of circuits. We also introduced probability into the logic to describe circuit properties such as metastability behaviors.

  - We proposed a framework to convert verification problems to reachability computation problems by a method that we believe could be performed automatically. We also suggested several techniques to obtain a good trade-off between performance and error during the computation.

- We designed and implemented a robust and efficient reachability computation tool, COHO, for moderate-dimensional, non-linear, hybrid systems.

  - We use *projectagons* to represent moderate-dimensional, non-convex regions. We avoid performing operations with exponential complexity on the high-dimensional objects. Instead, all operations are implemented using efficient algorithms on the two-dimensional projections or by linear programming on convex approximations of projectagon faces.

  - Highly non-linear dynamic systems are over-approximated by linear differential inclusions which are solved efficiently. Linearization is performed locally for each face of a projectagon to reduce approximation error.

  - We applied interval computation and arbitrary precision rational arithmetic to develop a robust linear program solver and projection algorithm which are essential to make COHO numerically stable. We also proposed novel algorithms to reduce computation error and improve performance of the reachability computation, including interval closure and an approximate LP solver.

  - The COHO tool has been released to the public research community[1].

- We have formally verified practical circuits including both synchronous and asynchronous digital circuits, and analog circuits.

  - We verified the Yuan-Svensson toggle circuit [217]. The output of the toggle should transition once for every two transitions of the clock input; in particular, the output makes a low-to-high or high-to-low transition for each rising transition of the clock. We found an invariant subset

[1] Available on http://coho.sourceforge.net

of circuit states and verified that all trajectories in this set have a period twice that of the clock signal [210, 211]. Because the output and clock signal satisfy the same specification, an arbitrarily large ripple-counter can be composed by using the output of one toggle to drive the input of another one. This verification also revealed that we had neglected to add keepers circuits to the design to ensure correct operation in spite of the leakage currents in deep sub-micron designs. Once we added these keepers, COHO verified the design.

- We showed that the output of a pass-gate latch circuit is stable when its clock signal is at logical low value. Further, we demonstrated that a flip-flop consisting of two latches works properly. The clock-to-q delay and maximum clock frequency of this flip-flop have also been measured.
- We formally specified and verified both safety and liveness properties of a two-input, asynchronous arbiter circuit [212, 213]. In this verification, we encountered the *stiffness* problem for reachability computations and proposed two solutions. We showed that all trajectories of the arbiter are safe, and we extend the method from [160] to show that the arbiter is live for all trajectories except for a set of measure zero.
- The Rambus oscillator challenge was posed by researchers from Rambus, Inc [129]. The challenge is to show that a differential ring oscillator with an even number of stages starts properly from all initial conditions. We combined static analysis and reachability computation to find the conditions under which the circuit can oscillate as expected from all initial states.

1.4 Organization

The thesis is organized as follows:

- Chapter 2 describes prior research in circuit verification and reachability analysis. It also explores related formal verification methods and reachability analysis techniques as well as developed tools. Several circuits are

presented as verification examples to show abilities and limitations of available verification methods. Prior research on COHO is also presented at the end of this chapter.

- Chapter 3 presents our framework for translating a circuit verification problem to a reachability analysis problem. It describes methods to construct an ODE model from circuit netlists and obtain table-based models for transistors based on simulations. It also presents our specification method for analog signals and properties which is based on Brockett's annulus construction and LTL logic. It introduces circuit examples used in this dissertation and provides formal specifications of properties to be checked. It also describes implementation issues that arise when computing linearized models and modeling input signals.

- Chapter 4 describes our reachability analysis tool COHO. It first describes the hybrid automata based interface and gives a high-level description of the reachability analysis algorithm. It then presents details of the projectagon representation method and operations on it. Based on these operations, algorithms to compute continuous successors are developed, including solving linear differential inclusions, computing projections and constructing a feasible projectagon. Techniques and approximation algorithms to improve performance and accuracy are also discussed. Several implementation issues such as the architecture of the COHO system are described at the end.

- Chapter 5 describes the digital and analog circuits that we have verified. It first presents the general process for circuit verification using COHO and then provides four examples: the toggle circuit, the flip-flop, the arbiter, and the Rambus ring oscillator.

- Chapter 6 concludes the thesis and proposes future research topics.

2

Related Work

This chapter presents a survey of prior research related to AMS circuit verification. Section 2.1 gives an overview of formal methods and discusses their pros and cons. This includes equivalence checking, model checking and theorem proving. As reachability analysis is a promising and widely used technique for model checking, Section 2.2 presents existing solutions for its main challenges: modeling, specification and reachability computation. Section 2.2 also describes methods to reduce system complexity and compares currently available tools. Section 2.3 presents applications of these techniques, mainly focusing on four circuits that have been widely used as benchmarks. In addition to others' work, Section 2.4 describes the development of COHO by others and myself prior to my Ph.D. program. Section 2.5 summarizes both the contributions and the unresolved issues from prior research.

2.1 Formal Verification of AMS Circuits

This section explores existing formal techniques for verifying circuits using analog models. In practice, nearly all designers rely on simulations using SPICE and similar programs to validate their AMS designs. Many extensions have been made to the basic circuit simulation programs to improve simulation performance such as Monte Carlo simulation (Spectre [1]) and fast Spice (Ultrasim [2]), increase coverage [54], monitor simulation and check properties automatically (*i.e.,* run-

time verification) such as AMT (Analog Monitoring Tool) [150, 151, 163] and others [43, 59, 60, 138, 179, 220], or apply conservatively approximated models such as FSPICE [193]. However, none of these tools can guarantee full coverage. Formal techniques provide full coverage by considering all trajectories of a circuit starting from all possible initial conditions, and under all admissible variations on parameter values. Like digital verification, formal methods for AMS circuits can be grouped into three classes: *equivalence checking*, *model checking* and *proof-based methods*. To be sound, both equivalence checking and model checking must determine all reachable circuit states in order to perform comparisons or verify properties. Computing the complete reachable space is, in general, an undecidable problem unless the system dynamics are extremely simple [12, 109, 137, 173]. Therefore, approximation techniques must be applied. *Discretization* methods discretized the continuous state space into a discrete one, for which reachable sets can be computed by well-developed digital verification tools. On the other hand, *reachability analysis* approaches try to find a reasonable approximated result using efficient methods to represent continuous regions and solve continuous dynamics. Theorem-proving based methods attempt to avoid the state-space explosion problem by constructing a formal proof. However, the problems they are addressing are still undecidable. Furthermore, it relies on human insight and effort to create such a proof.

2.1.1 Equivalence Checking

Equivalence checking determines whether two systems are equivalent according to some criteria such as input/output behaviors. In [188], Steinhorst and Hedrich proposed an equivalence checking method for analog circuits based on their system dynamics. Given two circuits, the method samples their state spaces, constructs a linear mapping between sampled points in each small region, transforms dynamics into a canonical state space, and checks if they are the same to within some tolerance. Another approach was developed in [178] for comparing two VHDL-AMS designs. It applies rewriting rules and pattern matching to simplify analog components and uses classical SAT/BDD equivalence checkers for digital components.

2.1.2 Model Checking

Model checking is a powerful technique for determining whether a mathematical model of a system meets a specification automatically. The first practical successes of model checking were for discrete systems, and this has motivated extending these techniques to handle designs with continuous models. There are two main approaches: *discretization* which approximates continuous models by discrete ones, and *reachability analysis* which solves continuous dynamics directly.

Discretization

The idea of *discretization* techniques is to convert a model checking problem in a continuous space to a discrete problem by discretizing space and time. Typically, these approaches partition the entire state space into hyper-rectangles, calculate transitions between these boxes using simulations or approximation techniques, and generate a finite-state system such as finite-state machines, transition systems, or graphs. Conventional model checking algorithms can be applied to these discrete systems. Refinement is used when the approximation error is large.

The first work using circuit-level models was by Kurshan and McMillan [133]. The algorithm first partitions the continuous state space representing the characteristics of transistors into fixed size hyper-cubes and divides continuous time into uniform time steps. Input signals are divided similarly but only logic low and high regions are used with the assumption of instantaneous transitions. Second, the algorithm computes the transition relation between these hyper-cubes using the lower and upper bounds of the continuous dynamics. The final constructed model is verified against properties defined by ω-language using a language containment tool, COSPAN [96]. The partition is refined manually and the procedure is repeated if the verification fails. A similar technique is used in [56] to check AnaCTL specifications[1]. However, transitions are constructed using SPICE simulations.

The simple approach for discretization proposed by Kurshan and McMillan has been generalized in the AMCHECK [97, 98] tool by Hartong *et al.* AMCHECK makes several improvements on Kurshan and McMillan's approach. First, it uses a

[1] Specification languages in this section, including AnaCTL, CTL, CTL-AT, CTL-AMS, and ASL, will be described in Section 2.2.2.

varied time step rather than a constant one. Second, refinement is performed automatically on the initial uniform partitions. This procedure is continued recursively until behaviors of every box are uniform which is defined based on the length and direction of vector fields. Third, they proposed three algorithms for computing the transition relation between boxes. The first method computes an overestimated solution by interval analysis. The second approach uses simulations from a number of test points. The method used in Kurshan's work is a special case of this approach, which exploits the fact that the transistor drain-to-source current is monotonic. This is valid for the device models used in practice and allows Kurshan and McMillan to use the lower and upper corner values to bound the dynamics. The third approach makes the second process rigorous using Lipschitz constants of nonlinear functions. However, similar to Kurshan's work, it also assumes that the values of input signals do not change at all or change instantaneously over the whole input value range. AMCHECK converts the nonlinear analog systems to a transition graph on which CTL specifications can be verified. The transition graph is augmented with delay information in [81]. Therefore, properties specified by CTL-AT [81] or ASL [187] can be checked. A similar tool MSCHECK is implemented in [126] where properties are specified by CTL-AMS.

Discretization methods leverage the extensive work in developing model checkers for digital designs. However, the number of hyper-rectangles in the discretization increases exponentially with the number of dimensions. Refinement strategies increase the number of hyper-rectangles, and this increase can be dramatic. Therefore, discretization methods are only suitable for small circuits.

Reachability Analysis

Reachability analysis completely explores the state space of a system by solving both the continuous and discrete dynamics. There are two main types of analysis. *Forward reachability* starts with initial states and follows trajectories forward in time. *Backward reachability* starts with target states and follows trajectories backward in time. In this dissertation, we distinguish two different kinds of *reachable regions* that a reachability algorithm might generate: a *reachable set* is the set of states occupied by trajectories at some specified time, and a *reachable tube* is the

set of states traversed by those same trajectories over all times in a closed or unbounded interval. Forward and backward versions of both reachable sets and tubes can be specified.

A general framework of reachability algorithms can be obtained based on fixed-point computations. In each iteration, a new (forward) reachable set is computed by applying the $post_c$ and $post_d$ operators to the current reachable set S, where $post_d(S)$ is the *discrete successor* defined as the set of states reachable by taking a transition from a state in S, and $post_c(S)$ is the *continuous successor* defined as the set of states that result by letting time elapse without state transitions. The computation of $post_d$ is the same as for discrete model checking. Therefore, solving the continuous dynamics of the $post_c$ operator is, in general, the biggest challenge and the most expensive step of reachability analysis for continuous or hybrid systems. Various of techniques have been proposed which will be discussed in Section 2.2.4. The reachable tube over this time step is usually overestimated based on reachable sets S and $post_c(S)$, *e.g.*, the bloated convex hull of S and $post_c(S)$. Backward reachability analysis is performed similarly using pre_c and pre_d operators. Forward algorithms terminate when no new reachable states are found. Conversely, backward algorithms terminate when no further restrictions of the safety set are found. However, termination of algorithms is not guaranteed even for highly restricted models [109]. Thus, each of these algorithms must fail for some inputs. This failure could be a failure to terminate, an incorrect rejection of a correct design, or an incorrect acceptance of an incorrect design. In a verification context, it is important that the particular limitations of a particular tool are clearly and correctly identified.

There are several reachability analysis tools for systems with continuous state and/or time that have been developed in recent years. We list these tools here for the discussion in the remainder of this chapter. Detailed features of these tools will be presented in Section 2.2 and summarized in Section 2.2.6. These tools include MOCHA [18], UPPAAL [20], KRONOS [216], TAXYS [30, 45], RED [203] for real time systems; HYTECH [104], PHAVER [70], LEMA [144] for hybrid systems with constant dynamics, and HYPERTECH [107], D/DT [48], CHECKMATE [40] for hybrid systems with linear or non-linear dynamics. There are also several tools that have been developed by researchers in the control community includ-

ing VERISHIFT [31], TOOLBOXLS [159] and zonotope based analysis [77, 79]. In [75], Frehse and Ray present a tool framework, SPACEEX, to integrate and compare different algorithms and features. Some constraint based solvers such as HYSAT [116], HSOLVER [177] have also been used in circuit verification.

Most circuit behaviors can be modeled by nonlinear dynamics, with non-determinism as needed to account for uncertainties in the model, parameter values, input, and operating conditions; thus, reachability analysis has the potential of verifying complex properties of real circuits. However, these dynamics, *e.g.*, differential equations, generally do not have closed form solutions. Therefore, approximation techniques must be applied. Furthermore, reachability tools suffer from the state-space explosion problem. In addition to solving complex dynamics, all model checking methods require formal models for circuits and properties to be verified. Solutions to these challenges will be discussed in Section 2.2.

2.1.3 Proof-Based and Symbolic Methods

Theorem proving establishes design properties by using formal deduction based on a set of inference rules. In addition to deductive based methods, induction and symbolic based methods have also been proposed to verify circuits. In [76], Ghosh and Vemuri used the higher-order-logic proof checker PVS to verify DC and small signal behaviors of synthesized analog circuits. They used piece-wise linear approximation to model each component, and a subset of VDHL-AMS language to specify properties. A similar but more elaborate approach was taken by Hanna in [94] for digital systems with analog-level abstraction. The circuit behavior is characterized by conservative rectilinear [95] or piece-wise linear predicates over the voltages and currents at the devices' terminals. Al Sammane *et al.* [180] transformed circuits to system of recurrence equations (SRE) by rewriting rules, and proved correctness using an induction based verification strategy. The work was extended in [219], where Taylor approximations and interval arithmetic were applied in a bounded model checker to generate the SRE model and check properties.

In principle, symbolic theorem proving methods do not suffer from the state-space explosion problem of model checking techniques. However, they require substantial human insight and intervention. First, they require a formalization of

the underlying theory. Embedding calculus including dynamical systems theory and circuit modeling into a theorem prover would be a huge undertaking. Second, we would still face the problem that the models do not have symbolic solutions, *i.e.,* most ODEs do not have solutions in terms of polynomials and elementary functions. Therefore, approximation techniques must be applied even if we use a theorem prover. Then, all of the questions of how to represent high-dimensional regions, how to approximate solutions to ODEs, and how to bound reachable sets would still apply. Furthermore, many problems are not decidable. There is no guarantee that a proof (or counterexample) exists, or that the human and theorem-prover can find it if does.

Discretization, reachability analysis, and theorem proving offer three basic approaches for formally verifying properties of circuits. In this thesis, we focus on reachability methods. We will show that by using a suitable representation of regions in the continuous state space, reachability computations can overcome the state-space explosion problems that have restricted discretization methods to low-dimensional models. Furthermore, reachability methods do not face the need of finding symbolic solutions to ODEs, and thus can be used with realistic circuits more readily than the theorem proving based methods that we have seen.

2.2 Reachability Analysis of Hybrid Systems

As described in the previous section, reachability analysis, which models AMS circuits as hybrid systems, is a promising technique for practical circuit verification. This section presents reachability analysis techniques and tools. Any verification method must start with a model and a specification, which translates a physical problem into a mathematical problem. Typically, models build on well understood abstractions such as automata or Petri nets with extensions to incorporate continuous dynamics. Section 2.2.1 examines various models that have been developed by the hybrid-systems community. Section 2.2.2 goes on to look at specification methods, for example, extensions of traditional temporal logics to systems with continuous state. Most prior work has focused on verifying safety properties of hybrid systems; this amounts to computing (over-approximations of) the regions that can be reached by the model. Such computations require a tractable way to represent

multi-dimensional regions and a way to compute the evolution of such regions according to the continuous dynamics of the system. Section 2.2.3 describes many of the most common representations for multi-dimensional regions, and Section 2.2.4 presents algorithms for advancing these regions according to continuous dynamics. The challenges of representing and manipulating multi-dimensional objects motivates developing methods to reduce the complexity of the models and analysis. Section 2.2.5 describes such methods. Other surveys of tools for hybrid systems can be found in [22, 28, 185, 201, 221].

2.2.1 Models

This section introduces some commonly used models for hybrid systems, including hybrid automata, hybrid Petri nets and transition systems. Methods of extracting continuous dynamics from circuit netlists are summarized at the end of this section.

A formal model for hybrid systems is a *Hybrid Automaton (HA)* [9, 105]. Hybrid automata have several similar definitions from different research groups. Informally, a hybrid automaton is a finite state machine augmented with continuous variables and dynamic equations. It consists of a graph in which each *vertex*, also called *location*, or *mode*, is associated with a set of ordinary differential equations (ODEs), $\dot{x} = f(x)$, or ordinary differential inclusions (ODIs), $\dot{x} \in F(x)$, that define the time driven evolution, referred to as *rate*, *derivative* or *flow*, of *continuous variables*. A *state* consists of a location and values for all continuous variables. The *edges* of the graph, also called *transitions*, allow the system to jump between locations, thus changing the dynamics, and instantaneously modifying variable values according to a *jump condition*. The jump may only take place when variable values satisfy a certain condition, specified by a *guard*, associated with each transition. The modified values for continuous variables after the transition are also referred to as the *reset map*. The system starts from one or more locations labeled as *initial* and may only remain in a location as long as the variable values are in a region called the *invariant* associated with the location.

Hybrid automata can be classified by their associated dynamics. *Timed Automata (TA)* [12] are a simple class of hybrid automata in which all continuous variables have a derivative of $+1$, *i.e.*, they are "clocks". *Linear Hybrid Automata*

(LHA) [100] represent dynamics using linear differential inequalities of the form $A\dot{x} \leq b$. However, TA or LHA are generally not expressive enough to accurately model systems with complex dynamics, especially nonlinear AMS circuits. A more powerful model is *Linear Dynamical Hybrid Automata (LDHA)*[2] which has linear dynamics, such as linear ODEs or linear differential inclusions. *Nonlinear Hybrid Automata (NHA)* support arbitrary nonlinear differential equations. The *reachability problem* of hybrid automata is to determine if a target state is reachable from an initial state. It is undecidable even for quite simple automata such as LHAs [12]. More results about decidability of hybrid automata can be found in [109, 137, 173]. Thus, verification procedures for hybrid automata must use approximate algorithms. We examine trade-offs made in making these approximations when we describe various tools in the remainder of this chapter.

Hybrid automata are widely used by many tools, such as TAs by KRONOS [57], LHAs by HYTECH [105], LDHAs by D/DT [25], and NHAs by TOOLBOXLS [194]. Models employed by other tools are listed in Table 2.1. A complex automaton is usually approximated by several simpler ones. For example, UPPAAL [140] and HYTECH [105] developed methods to transform LHAs to TAs, and PHAVER [70] approximates LDHAs by LHAs.

Several similar modeling frameworks have been used by reachability analysis tools. For example, the linear *Hybrid Input Output Automata (HIOA)* [74] used in PHAVER extends LHAs by specifying some variables as inputs and outputs. *Hybrid Petri nets* combine discrete Petri nets and continuous Petri nets. *Timed Hybrid Petri Nets (THPN)* [147] and enhanced *Labled Hybrid Petri Nets (LHPN)* [146] are employed in LEMA. However, rates of continuous variables in THPNs or LHPNs are either constant values or interval values, hybrid Petri nets with complicated dynamics have not been studied. *Transition systems* [149] consist of a set of finite or infinite states, a transition relation and a set of initial states. They are widely used to abstract away continuous behaviors of hybrid automata in the abstraction-refinement strategy which will be discussed in Section 2.2.5. For example, CHECKMATE [41] constructs a *Discrete-Trace Transition System (DTTS)* from its *Polyhedral Invariant Hybrid Automaton (PIHA)* model in a bisimulation

[2]It is called linear hybrid systems in some papers.

based model checking algorithm.

To model an analog or mixed-signal circuit, continuous dynamics must be extracted from its netlist in advance. The first approach is based on *modified nodal analysis*. For example, *bond graphs* are used to describe a circuit in [58], from which ODEs can be generated automatically. Another approach is based on tableau data from simulation traces. For example, LEMA uses a *Simulation Aided Verification (SAV)* [145] method to generate LHPN models automatically. The method partitions the state space into boxes based on user provided thresholds and calculates bounds of dynamics from the simulation data. FSPICE [193] also uses conservative tables which represent the I-V relationships of circuit devices by intervals. The first approach is similar to the one used in simulators and is well-studied. However, the second approach can handle uncertain inputs, PVT variations, disturbances and noise. It is especially attractive for small circuits. For large systems, an intractably large number of simulations are often required to obtain a reasonable coverage.

In summary, formal models for circuits are often constructed by deriving the continuous dynamics from the netlist using modified nodal analysis, and then creating a hybrid automaton to partition the trajectories of the model into bundles of interest. We follow this framework in our tool as shown in Section 3.3 and Section 4.1.

2.2.2 Specification Languages

Having examined some of the most common methods for modeling hybrid systems, we now consider how the analog properties can be specified. Temporal logics are the most popular formalism for specifying properties of digital circuits, such as *Linear Time Temporal Logic (LTL)*, *Branching Time Temporal Logic* (*e.g.*, CTL). The *Accellera Property Specification Language* (PSL, a.k.a. IEEE P1850) [64] is a specification language that contains LTL and CTL as subsets and is supported by various commercially available verification tools. These discrete temporal logics can be applied directly to represent properties of a hybrid system. For example, Kurshan *et al.* used ω-languages to specify properties of the transition graph in their discretization based algorithm [133]. CHECKMATE checks properties spec-

ified by *ACTL* [40, 89], which is a universal fragment of CTL without existential paths.

However, conventional temporal logics are based on discrete time and state and cannot express properties with continuous variables and dense metric time. Therefore, several researchers have extended temporal logics with time and real-valued variables. Generally, timed logic is obtained by putting constraints on temporal operators to limit their scope in time. For example, *Real Time CTL (RTCTL)* [63] uses superscripts to bound the maximum number of permitted transitions along a path. *Timed CTL (TCTL)* [8] puts subscripts on the temporal operators to limit the lower or upper bound of accumulated time over paths. *Metric Interval Temporal Logic (MITL)*[3] [13] constrains the LTL temporal operators with time intervals. On the other hand, continuous space is supported by introducing real-valued variables and predicates to the logic. For example, *Analog CTL (AnaCTL)* [56] adds propositions based on linear predicates over continuous variables to CTL. Similarly, PSL has been extended to support continuous space by using linear predicates in the boolean layer [179]. However, these logics still use discrete time. Temporal logics that support both dense time and continuous state space have also been developed. *Analog and Timed CTL (CTL-AT)* [81, 98] constrains temporal operators by intervals and expresses continuous regions by linear predicates. *CTL-AMS* [125] extends CTL-AT by supporting unconstrained time (or the time interval is $[0,\infty]$) over temporal operators. *Continuous-Time CTL (CT-CTL)* [220] extends TCTL with predicates. *Signal Temporal Logic (STL/PSL)* [150, 152] combines MITL with linear predicates which map analog signals to boolean variables. Furthermore, *Integrator CTL (ICTL)* [100] supports accumulated time by introducing integrator variables. These extensions have been applied in many tools. For example, LEMA [199] uses TCTL, HYTECH [16] and PHAVER [70] uses ICTL. Table 2.1 on page 35 lists some of the main tools from the research literature along with the temporal logics that they support.

Although these temporal logics can express many important properties of hybrid systems, they cannot specify many analog properties directly. Therefore, designer-oriented languages have been proposed. AnaCTL supports waveform

[3] It is called $MITL_{[a,b]}$ in some papers.

propositions by comparing signal values with reference waveforms provided by equations or tables generated by users. STL specifies continuous variables by partial functions and supports operations on signals such as concatenation, projection, and comparison with a reference signal. STL/PSL [163] extends STL with a layered approach in the fashion of PSL. It uses an analog layer to reason about continuous signals directly. *Mixed-Signal Assertion Language (MSAL)* [138] is based on PSL and supports digital, analog and software properties. However, these languages are for assertion based verification and only cover signal-based properties. The *Analog Specification Language (ASL)* [187, 189] is designed for describing properties of analog systems over a continuous region. For an operator and a bounded region, it applies the operator to every point in the region and calculates the range of values based on interval arithmetic. It also supports operations such as derivative computation, oscillation and start-up time. It is compatible with CTL-AT and has been implemented in AMCHECK.

There are some other methods, such as *timed regular expression* [21], and the method proposed in [74] which constructs a LHA for a property. However, most specification techniques are still based on temporal logics. Digital temporal logics have been extended to express properties of real-time systems but are not yet powerful enough to express most properties of interest for AMS circuits. For example, we are not aware of any specification approaches that formalize frequency domain properties which are very important for circuit analysis. Many of these extended logics are for signal-based properties and thus cannot be applied to reachability analysis based verification directly. It is still a major challenge to make the analog verification as fully automated as the current state of the art for digital model checking tools. Furthermore, temporal logics are not familiar to most circuit designers. Therefore, more expressive designer-oriented languages are needed.

2.2.3 Representation Methods

Given a mathematical model and a formal specification, reachability analysis computes reachable regions according to the model and checks if all these regions satisfy the specification. This requires techniques to represent multi-dimensional regions and algorithms to compute the evolution of these regions according to the

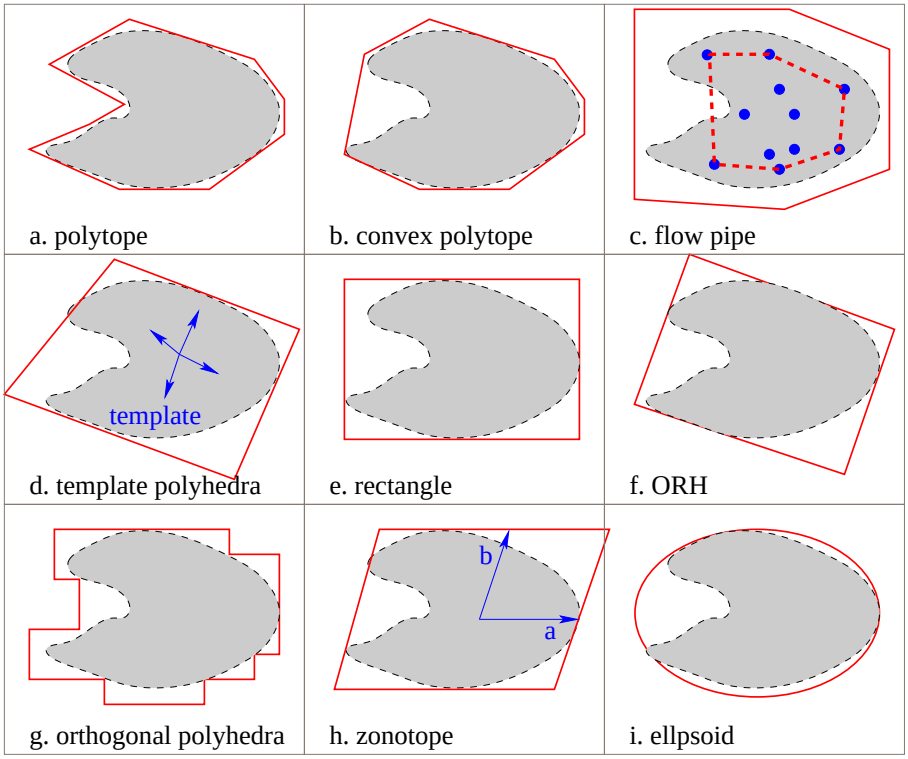


Figure 2.1: Representation Examples

model. In this section, we describe approaches to representing multi-dimensional regions, and Section 2.2.4 examines algorithms for computing reachable sets according to continuous dynamics.

The representation of regions in continuous state spaces is crucial for reachability algorithms as it usually determines the efficiency of algorithms and accuracy of results. This section describes several commonly used representation methods along with operations on them. We first explore geometry based methods, including polytopes which have the advantage of accuracy, hyper-rectangles or intervals aimed to maximize efficiency, zonotopes which are closed under several important operations, and ellipsoids. Figure 2.1 illustrates these methods by a simple two-dimensional example. We then examine some symbolic data structures, including widely used BDD-like structures and support functions. The op-

erations used in reachability analysis include union, intersection, and intersection with hyperplanes. Some reachability algorithms also require the *Minkowski sum* operation. The Minkowski sum of two sets A and B in Euclidean space is defined as the result of adding every element of A to every element of B, *i.e.*, the set $A \oplus B = \{a + b | a \in A, b \in B\}$. The selection of a good representation depends on reachability algorithms, complexities of dynamics, trade-off of performance and accuracy, and so on. Exact representation is generally impossible due to the complexity of the geometry or dynamics. Therefore, approximation is widely used. For many reachability analysis algorithms, the errors from approximating the reachable region accumulate over successive time steps of the computation. This is known as the *wrapping effect*.

Polytopes

Polytopes can represent a bounded convex or non-convex region with arbitrarily small errors. However, the space and time complexity of operations on non-convex polytopes are generally exponential with the number of dimensions. Therefore, *convex polytopes* are used in practical tools. There are two commonly used representations for convex polytopes: the *inequality representation* and the *frame representation*. The first approach represents a half-plane by a linear inequality; thus some operations such as intersection can be implemented efficiently by manipulating system of inequalities. The second approach represents an object by points and rays and has other efficient operations such as convex hull. Translations between these two representations can be computed by several algorithms [39, 143].

Convex polytopes are generally used to represent reachable sets for TAs or LHAs which have exact reachability algorithms. Both HYTECH and PHAVER employ convex polytopes, furthermore, HYTECH also supports unbounded regions by *widening* [9] or *extrapolation* [102] techniques. HYTECH uses Halbwachs' library [92, 93] for polytope operations which uses limited precision rational numbers. Therefore, HYTECH suffers from the overflow problem. To overcome the limitation, PHAVER uses the Parma polyhedra library [27] which supports arbitrary precision rational numbers. For more complex dynamical systems, reachable regions cannot be represented exactly and the wrapping effect must be considered.

CHECKMATE developed a *flow pipe* representation [41], which is essentially a convex polytope with the inequality representation, to over-approximate reachable tubes estimated from simulation traces. It avoids the wrapping effect by restarting simulations from initial regions in each step.

When a system has non-linear dynamics, a line-segment can evolve to a more general curve. Accordingly, polytopes are not closed under evolution with non-linear dynamics, and approximations must be used. In principle, these approximations can be made arbitrarily precise by using a polytope with enough faces, but the space and time required to represent and operate upon such polytopes quickly become intractable. *Template polyhedra* [181] have been proposed to limit the complexity of reachable sets. These are polytopes whose inequalities have fixed expressions (template) but with varying constant terms. Therefore, the number of faces of a template polyhedron does not increase with successive time steps. However, it can produce large approximation errors, and it is a challenging problem to find a good template[4].

Rectangles

Polytope-based representations are accurate but expensive. At the other extreme, the *hyper-rectangle* representation optimizes performance at a cost of large approximation errors. The space complexity of hyper-rectangles is linear with the number of dimensions, and the time complexities of operations on hyper-rectangles are typically small. Many interval arithmetic algorithms use interval-valued variables where the valid solution is equivalent to a hyper-rectangle. For example, HYPERTECH [107] uses an interval based ODE solver. HYSAT [66] also applies interval arithmetic to solve nonlinear constraints and ODEs.

Several variations of hyper-rectangles have been developed to improve accuracy. D/DT developed techniques based on orthogonal polyhedra [33]. Orthogonal polyhedra represent a region as the union of a finite number of uniform or non-uniform hyper-rectangles. This representation allows arbitrarily small approximation errors by using sufficiently small hyper-rectangles, but the number of rectangle needed to represent a region grows rapidly with decreasing hyper-rectangle size.

[4]Flow pipes are essentially template polyhedra where the template is computed from simulation results.

The *Oriented Rectangular Hull (ORH)* representation [191] reduces approximation error by rotating a rectangle to an orientation that is chosen according to the dynamics of the system. The space complexity of ORH is quadratic with the number of dimensions. However, ORH is not closed under operations such as union and intersection. The *face region* [167, 172] representation over-approximates a n-dimensional region by the convex hull of a set of $(n-1)$-dimensional hyper-rectangular faces with one dimension fixed to a constant value.

Zonotopes

A zonotope is a polytope which can be represented as the Minkowski sum of segments. The order of a zonotope is defined as the ratio of the number of segments to the number of dimensions. Particularly, a hyper-rectangle[5] is a special zonotope with order 1. Zonotopes have many attractive features. First, they are closed under linear transformations and Minkowski sum operations, and there are efficient algorithms for implementing these operations. Second, the representation is very compact. However, zonotopes have two main drawbacks. First, the order of zonotopes increases after each Minkowski sum operation. To reduce the order of a zonotope, an efficient algorithm was proposed in [77] to compute an approximation of the zonotope. However, the approximation causes wrapping effect errors. Second, it is expensive to compute the intersection of a zonotope and a hyperplane. An efficient approximate algorithm was proposed in [78] by projecting the zonotope onto two-dimensional planes.

Ellipsoids

A promising representation from the control community is *ellipsoids* [134]. A d-dimensional ellipsoid is specified by a center point and its d axis vectors. Algebraically, an ellipsoid can be described as the set of all points x satisfying[6]: $x^T Ax \leq 1$. This is similar to an oriented hyper-rectangle that can be expressed by systems of inequalities: $Ax \leq 1$. Like the ORH representation, the space complexity for the ellipsoidal representation is quadratic in the number of dimensions,

[5] More generally, zonotopes of order 1 are the set of parallelepipeds.

[6] The mathematical form $x^T Ax \leq 1$ is for ellipsoids centered at the origin; a more general form is $(x-x_c)^T A(x-x_c) \leq 1$.

and the time complexity of ellipsoidal operations is also polynomial. Furthermore, a reachable region can be approximated with an arbitrarily small error through intersection (union) of a family of external (internal) ellipsoids. Operations on ellipsoids are implemented in the Ellipsoidal Toolbox [135] and in [31].

Symbolic Data Structures

In addition to geometry based representations, symbolic functions have also been used in reachability analysis tools. For example, TOOLBOXLS uses implicit surface functions to represent sets [161]. An implicit surface function for a subset of a given state space is a scalar function defined over the entire state space whose value is negative inside the subset, positive outside, and zero on the boundary. Implicit surface functions can represent sets with an interior exactly; however, representation of the implicit surface function itself often requires an amount of data (*e.g.*, function values on a grid) which increases exponentially with dimension. A *support function* [90], employed in the SPACEEX tool, is a function that bounds the maximum value on all possible directions. They can represent a convex set with arbitrary precision and have efficient algorithms for union, Minkowski sum as well as affine transformation operations. However, it lacks an efficient intersection operation. An over-approximation can be computed based on the projection concept [90] which is similar to the algorithm for zonotopes [78]. The well-developed BDDs and BDD-like structures are also used in reachability analysis tools, including *Difference Bound Matrices (DBMs)* [146] in LEMA, *Multi-Terminal BDDs (MTBDDs)* [126] in MSCHECK, *Clock Difference Diagrams (CDDs)* [20] in UPPAAL, *Clock Restriction Diagrams (CRDs)* [200], and *Hybrid-Restriction Diagrams (HRDs)* [202] in RED.

Summary

The choice of representation for reachable regions has a strong impact on the efficiency of analysis techniques. Prior representations have various trade-offs between the complexity of the model and the number of dimensions of a region that can be efficiently represented. To support complicated, non-linear dynamics, these methods are restricted to low-dimensional models. Conversely, some represen-

tations such as zonotopes [79] and support functions [71] have been used with higher-dimensional models, but these applications have been restricted to linear, and piecewise-linear models. Furthermore, most prior methods can only represent convex regions. The exceptions to this are level sets and collections of hyper-rectangles that are only practical for representing low-dimensional sets. The trajectories that arise from circuit models are often roughly hyperbolic, and this gives rise to non-convex reachable regions, where the non-convexity is critical to verifying the circuits. There remains a need for ways to represent moderate-dimensional (*e.g.*, ten to twenty dimensions), non-convex regions with nonlinear dynamics.

2.2.4 Solving Dynamics

One of the most challenging aspect of computing reachable regions is solving continuous dynamics. Commonly used dynamics for hybrid systems include *Ordinary Differential Equations (ODEs)*, *Ordinary Differential Inclusions (ODIs)*, *Differential Algebraic Equations (DAEs)* and *Difference Equations (DEs)*. Usually, ODEs are used to describe deterministic dynamics; and ODIs are used to describe non-deterministic dynamics (*e.g.*, inputs, and noise) or to conservatively approximate complicated dynamics. DEs are used for discrete-time system and can be solved easily in many cases. DAEs are generally solved by transforming them to ODEs, for example, they are converted to the semi-explicit form in [50]. Therefore, this section focuses on computing the forward reachable set $post_c(S)$ as shown in Section 2.1.2 with dynamics modeled by ODEs or ODIs.

ODEs include *clock ODEs* of the form $\dot{x} = 1$, *constant ODEs* of the form $\dot{x} = c$, *linear ODEs* of the form $\dot{x} = Ax + b$, and *nonlinear ODEs* of the form $\dot{x} = f(x)$. Two commonly used ODIs are *constant ODIs* of the form $\dot{x} \in [c_l, c_h]$ or $A\dot{x} \leq b$ and *linear ODIs* of the form $\dot{x} \in Ax + U$, where U is a set or region used to model uncertainty, non-determinism or errors. Table 2.1 on page 35 lists the forms of dynamics supported by many of the tools reported in the research literature. We first describe reachability algorithms for constant dynamics briefly as they can be solved efficiently and exactly. We then describe methods for solving linear ODEs or ODIs, including the *optimal control* and the *Minkowski sum* methods. We focus on nonlinear ODEs because they are used to model AMS circuits. We survey

related techniques to compute approximated solutions, including *hybridization* and *level sets*.

Constant ODEs and Constant ODIs

Mathematical solutions exist for some simple dynamics, including clock ODEs, constant ODEs, and constant ODIs. Therefore, reachable regions are generally computed exactly for TAs and LHAs by either symbolic manipulation or geometric operations. Model checking algorithms for TAs and LHAs can be found in [8, 110, 112, 201] and are used in several tools including UPPAAL [139], KRONOS [32], LEMA [198, 199], HYTECH [122], and PHAVER [73].

Linear ODEs and Linear ODIs

The solution of a linear ODE, $\dot{x} = Ax$, is $x(t) = e^{At}x(0)$. However, the problem of finding reachable sets for ODIs is more difficult. The challenge lies in the uncertainty set U. Typically, U has the form of a hyper-rectangle or other convex polytope. The vertex of U that leads to an extremal trajectory can change during a time-step. Exact computation of the extremal trajectories is difficult; thus, practical tools use various forms of over-approximation. Two approaches have been developed for computing a conservative approximation R of forward reachable set with linear ODIs. The first method, *optimal control*, is based on the maximum principle of the optimal control theory [197]. This method finds the optimal input $u^*(t)$ which leads to the boundary of R and numerically computes the integration. The optimal input for a hyperplane with normal vector d is a function $u^*(t)$, where at time t, $u^*(t)$ is a value $u \in U$ that maximizes $e^{-A^T t}d \cdot u$. The second method is based on *Minkowski sum* and approximates the reachable region by the Minkowski sum of a region $\hat{R}$ and an error region E as the equation $R \in \hat{R} \oplus E$. The region $\hat{R}$ is computed using the autonomous dynamics $\dot{x} = Ax$ (*i.e.*, $\hat{R}_{i+1} = e^{A\delta t}R_i$) and the error region E accounts for the influences of inputs or disturbances. The error region can be bounded using error analysis. For example, for any nonlinear ODE with Lipschitz constant L, the radius r of the error region E is bounded by $r = ||R - \hat{R}|| \leq \frac{\mu}{2}(e^{L\delta t} - 1),\ \mu = \max_{u \in U} ||u||$, which is proved by the fundamental inequality theorem from the theory of dynamical systems [23].

An example of the first approach is the D/DT tool [197]. It assumes the uncertain set U is a bounded convex polytope and finds the optimal input by linear programming. The zonotope based method in [77] uses the Minkowski sum method. It constrains set U as a hyper-rectangle, and approximates the error region E as a hyper-rectangle and consequently a zonotope with radius $r = \frac{e^{||A||\delta t}-1}{||A||}\mu$. Based on this method, a reachability algorithm was developed in [79] which is free of the wrapping effect. VERISHIFT also employs the Minkowski sum approach and approximates the error region E using the Hausdorff semi-distance of reachable sets [31].

Nonlinear ODEs

Unlike linear ODEs, nonlinear ODEs do not, in general, have closed-form solutions; thus approximation techniques must be applied. The *hybridization* method partitions the state space into small regions and bounds the nonlinear dynamics by simple ODIs in each region. The *set integration* method extends numerical integration to reachability analysis by interval arithmetic and Taylor expansion. The *constraint based* approach transforms ODEs to constraints which can be solved by constraint solvers. Other methods such as *level set*, *flow pipe* and methods for some special classes of nonlinear ODEs are presented at the end of this section.

The idea of *hybridization* is to (over-)approximate complex dynamics by simpler ones which can be solved efficiently. The approximation is calculated in a small region in order to obtain small error. Therefore, the hybridization procedure usually consists of two steps: partitioning the state space of the system into small regions, and computing an (over-)approximation of the solution. Often, the state space is discretized into disjoint regions, such as hyper-rectangles or simplexes [24]. To obtain a conservative result, nonlinear ODEs are over-approximated by ODIs. The commonly used ODIs are constant ODIs or linear ODIs, which can be solved as described above. As the method is based on discretizing the state space, refinement strategies can be applied. Globally, the dynamics of the approximated system changes when moving from one region to another. Therefore, the intersection of reachable sets and the region boundary must be computed as the initial set for the reachability computation in the next region. *Dynamic hy-*

bridization was presented in [51] to avoid possibly expensive intersection computations. Rather than partitioning the state space into distinct regions, this method generates overlapped regions. However, it may compute a forward reachable set twice with different approximated dynamics in two adjacent regions. The hybridization approach has been applied in several tools. D/DT developed two classes of approximate systems: the piecewise affine [23] system based on simplex regions and the piecewise multi-affine [24] system based on hyper-rectangular regions. In [53], the orientation of simplex regions is adapted to achieve better time efficiency for affine hybridization. The idea of constructing hybridization domains bases on the dynamics has been extended to more general nonlinear dynamics in [55]. D/DT also presented a *face lifting* method [52] which calculates the maximum derivative of the nonlinear dynamics in a small neighborhood of each face of an orthogonal polyhedra. Constant ODIs are applied in PHAVER [70], HYTECH [105] and LEMA [148].

Set integration computes all possible trajectories from a set of initial points by using intervals to represent the initial points and using interval arithmetic throughout the algorithm. ADIODES [186] used in HYPERTECH [107, 174] is an example of this approach. ADIODES approximates a nonlinear ODE by using its Taylor expansion with its remainder. The wrapping effect is a serious problem for interval-arithmetic based methods such as set integration.

Closely related to set integration are *constraint-based methods*. Here, the approach is to represent an integration algorithm by a series of constraints. Then, properties of trajectories can be verified by solving the constraint systems. For example, Hickey *et al.* [118, 119], developed the *Constraint Logic Programming (Functions) (CLP(F))* which combines constraint languages programming with interval arithmetic. It transforms ODEs to constraints based on Taylor expansions. HSOLVER [177] converts nonlinear ODEs to constraints without differential operators. HYSAT [62] integrates set integration and CLP(F) methods into its iSAT [65, 66] constraint solver. As with set integration, the constraint solvers rely on interval arithmetic for much of their computation, and these methods suffer from serious errors due to the wrapping effect.

There are several other approaches for solving nonlinear ODEs. The *level set* [161] method solves nonlinear ODEs based on the theorem that the solution

of a particular Hamilton Jacobian *Partial Differential Equation* (PDE) corresponds to the boundary of the reachable region of a nonlinear ODE. However, PDEs are even more difficult to solve than ODEs, and the number of grid points increases exponentially with the number of dimensions. CHECKMATE developed the *flow pipe* technique based on simulations. It samples several points in the initial region, runs simulations from these points until time points t_k and t_{k+1}, and computes a convex hull of all simulation points. To contain the real reachable tube, the convex hull is bloated outward. The bloat distance is bounded by error analysis assuming the ODE is a Lipschitz continuous function [40, 42]. This approach is free of the wrapping effect and allows parallel simulations. However, the bloat distance is largely over-estimated, and the approximation error increases rapidly with the number of system dimensions. Several techniques have been developed for some special forms of nonlinear ODEs. For example, a projection based method was proposed in [24] to solve multi-affine systems. Polynomial systems are solved in [49] by using Taylor expansions in the integration. There are also some techniques to analyze properties of nonlinear dynamics, such as barrier certificates [170], polynomial invariants [192] and Lyapunov functions [36, 128].

2.2.5 Reducing System Complexity

In order to apply reachability analysis to high-dimensional, nonlinear hybrid systems, it is important to reduce system dimensionality and the complexity of the system dynamics. There are two main approaches, namely *abstraction* and *modular analysis*.

The *abstraction* method maps a given model into a less complex model that retains the behaviors of interest. It is usually based on an *abstract-verify-refine* paradigm: build an abstract model, compute transitions, and check desired properties; if the abstract model is too coarse to analyze the specified properties, it can be refined, and the checking process is repeated. The abstraction step tries to transform an infinite state system into a finite state system by grouping together states that have similar behaviors. Such a grouping of states is usually implemented based on partitioning the state space into hyper-rectangles [177] or regions according to a set of predicates [11]. The computation of transitions between abstract states is

generally based on reachability analysis as described in Section 2.2.4. For example, CHECKMATE [41] uses the flow pipe method to compute over-approximated *quotient transition systems* in each iteration of its modified *bisimulation procedure* algorithm [99, 136]. Other examples are available in PHAVER [73] and D/DT [11]. *Lazy abstraction* was proposed in [108] which builds and refines an abstract model on demand to improve performance. One framework of lazy abstraction is *Counterexample Guided Abstraction Refinement (CEGAR)* which has been extended to hybrid systems in [44]. CEGAR has been applied in CHECKMATE [190], PHAVER [73] and D/DT[10].

Modular analysis or *compositional reasoning* is a divide-and-conquer mechanism for decomposing a verification problem of a large system into subtasks for each individual component of the system. Modular analysis examines parts of the system and verifies properties of the entire system in a deductive way. A particularly effective form of compositional reasoning is *Assume-Guarantee Reasoning (AGR)*, which analyzes a subsystem using the specification of a subsystem as an assumption that can be made about its behavior when verifying other subsystems. AGR has been studied in [15, 111, 114, 115] and applied in PHAVER [67, 72].

2.2.6 Summary and Reachability Analysis Tools

This section presented the four main challenging problems that must be addressed in the design of a reachability analysis tool: 1) constructing mathematical models for the system to be verified; 2) formally specifying the properties to be verified; 3) representing reachable regions and 4) solving nonlinear dynamics. We explored the prevalent techniques for these four problems and also discussed methods to reduce system complexities. Representing reachable regions and solving continuous dynamics are crucial when performing reachability analysis of complex systems. Many representations of multi-dimensional regions have been proposed based on geometric objects and symbolic functions. However, most of them do not work efficiently for systems with more than three dimensions, and they do not address the problem of representing non-convex regions. There are widely used methods to solve linear ODEs and linear ODIs, whereas approximation techniques such as hybridization must be applied to solve nonlinear ODEs. Furthermore, standard

mathematical models for circuits and their properties are necessary to apply reachability analysis tools to verify AMS circuits. Hybrid automata are a widespread formal model for hybrid systems, and there are general methods to extract continuous dynamics from circuit netlists. Discrete temporal logics have been extended to continuous domain. However, there remains much work to be done to develop a logic that can express the full range of properties needed for analog designs, and can be integrated in reachability algorithms.

This section described techniques of many reachability analysis tools. We summarize features of these tools in Table 2.1[7]. Tools for real-time systems are well-developed, including MOCHA [18], UPPAAL [20, 29], KRONOS [57, 216], and RED [200, 202, 203]. These tools typically model systems using clock variables and timed automata, specify properties by timed temporal logics (*e.g.,* TCTL), and represent reachable regions by BDD-like structures. In addition to verification, they usually also support simulation, parametric analysis, counterexample generation and so on. HYTECH [103, 107, 113] (The HYbrid TECHnology Tool) is one of the earliest tool for hybrid systems modeled by LHAs; it was developed by Henzinger *et al.* The earliest prototype was implemented based on symbolic computation in MATHEMATICA [16]. The second version [101] represented reachable regions by convex polytopes to improve performance. The third version [103] reimplemented the whole system in C++. It was further extended to support nonlinear dynamics based on interval ODEs solver as HYPERTECH [107]. In this dissertation, the term "HyTech" is used to refer to the third version unless otherwise stated. It also supports parametric analysis and diagnostic error generation [106]. However, it is limited to simple (*e.g.,* linear and low-dimensional) hybrid systems because it uses limited precision rational numbers for exact computation. To solve the overflow problem, Frehse *et al.* employed arbitrary precision rational numbers and implemented the PHAVER [68, 70] tool. They also implemented a separate engine for assume-guarantee reasoning [69, 72]. LEMA [144] (LHPN Embedded/Mixed-signal Analyzer) is a verification tool specified for AMS circuits developed by Scott Little *et al.* It models AMS circuits as Petri net based models which are compiled from VHDL-AMS codes or generated from simulation

[7] Details of our tool COHO will be presented in Chapter 3 and Chapter 4.

Tool	Model	Spec	Representation	Linear	Nonlinear	Reduction	Others
MOCHA	Reactive module [14]	ATL [17]	BDD	clock ODEs [18]		AGR [114]	diagnostic trace [18]
UPPAAL	TA [140]	TCTL [29]	CDD [20]	clock ODEs			diagnostic trace [20]
KRONOS	TA [57]	TCTL [216]	symbolic [32]	clock ODEs			forward & backward [216]
RED	TA [201], LHA [202]	TCTL [204]	CRD [200], HRD [202]	constant ODIs			parametric analysis of LHA [203]
HYTECH	LHA [105]	ICTL [16]	symbolic [16], convex polytope [102]	constant ODIs [100]	hybridization [105]		diagnostic error, parametric analysis [103]
PHAVER	linear HIOA [74]	ICTL [70]	convex polyhedron [70]	constant ODIs [70]	hybridization [70]	f/b refinement [73], AGR [67, 72]	same algorithm with HYTECH
LEMA	THPN [147], LHPN [146]	TCTL [199]	DBM [146]	constant ODIs [144]	hybridization [148]		SAV [148]
HYPERTECH	NHA [107]	ICTL [16]	interval [107]		set integration [107]		extension of HYTECH
D/DT	LDHA [25]		orthogonal polyhedron [33]	linear ODIs [197]	hybridization [23, 24], face lifting [52]	CEGAR [10]	
CHECKMATE	PIHA, DTTS [41]	ACTL [40]	flow pipe [41]	linear ODEs [42]	flow pipe [40, 42]	CEGAR [190]	
VERISHIFT	LDHA [31]		ellipsoid [134]	linear ODIs [31]			
Zonotope	LDHA [79]		zonotope [77]	linear ODIs [77]			
SPACEEX	LDHA [75]		support functions [75]				development platform
TOOLBOXLS	NHA [194]		level set [161]		level set [161]		
HYSAT	NHA [116]		interval [66]		constraint based [62]		BMC [65]
HSOLVER	NHA [177]		interval [176]		constraint based [176]	abstraction refinement [177]	
COHO	NHA	Brockett's annulus, LTL	projectagon	linear ODIs	hybridization	AGR	

Table 2.1: Comparison of Reachability Analysis Tools

traces automatically. It implemented three engines: a DBM based model checker, a BDD based model checker and a SMT based bounded model checker. These tools (*i.e.*, HYTECH, PHAVER, LEMA) support constant dynamics directly and nonlinear dynamics by hybridization.

More complicated dynamics are supported by D/DT, CHECKMATE, VERISHIFT, TOOLBOXLS, SPACEEX, *etc.* D/DT [23, 48] is a reachability analysis tool developed originally by Thao Dang. It represents reachable regions by orthogonal polyhedra and solves linear ODIs by the maximum principle method and nonlinear ODEs by face-lifting. Later extensions include support for hybridization [23, 24] and CEGAR [10, 11]. CHECKMATE [40, 184] is a Matlab based tool for modeling, simulating and verifying properties of nonlinear systems, developed by Krogh *et al.* It models systems by PIHAs, which are converted to transition systems by a modified bisimulation procedure, and applies the flow pipe technique to estimate transitions between abstract states. TOOLBOXLS [159, 161] is a MATLAB toolbox for level set methods, developed by Mitchell. It represents reachable regions by level sets and solves nonlinear ODEs directly by converting them to PDEs. VERISHIFT [31] is a bounded time reachability analysis tool for LDHAs and represents reachable regions by ellipsoids. SPACEEX [75] is a development platform from Verimag labs on which several different verification algorithms are implemented. The current implementation includes the PHAVER scenario which uses the PHAVER algorithm and the LGG scenario which computes reachable sets of linear systems using support functions. HYSAT [62, 66, 116] is a satisfiability checker for nonlinear arithmetic constraints and a bounded model checker for hybrid systems. SAT solving techniques and interval-based arithmetic constraint solving have been integrated in its iSAT algorithm. HSOLVER [176, 177] is a similar verification system based on the constraint solver, RSOLVER [175].

2.3 Verified Circuits

The verification methods and tools presented in Section 2.1 and Section 2.2 have been applied to verify some circuit examples. Table 2.2 summarizes these examples. In short, equivalence checking and proof-based methods have only been applied to verify simple circuit elements, such as NAND gates [188] and TTL logic

gates [95]. Simulation based tools including FSPICE and AMT have been applied to relatively larger circuits such as the DDR2 memory interface [130], and the Rambus ring oscillator (RRO) [193]. Both discretization and reachability analysis are restricted to simple circuits (*e.g.*, 2-3 variables) and simple properties. To show the capabilities of these verification methods, we describe the four, simple circuits that have been widely used as benchmarks by researchers in the formal methods community: a $\Sigma - \Delta$ *modulator (*$\Delta\Sigma$*M)*, a *tunnel diode oscillator (TDO)*, a *voltage controlled oscillator (VCO)*, and a *biquad lowpass filter (BLF)*.

2.3.1 A $\Delta - \Sigma$ Modulator

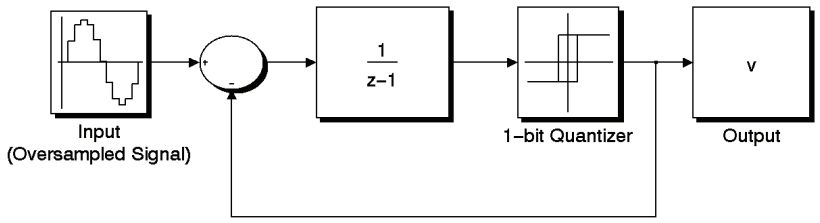


Figure 2.2: The First Order $\Delta - \Sigma$ Modulator

Figure 2.2 shows the block diagram of a first order $\Delta - \Sigma$ modulator ($\Delta\Sigma$M) [26]. The $\Delta\Sigma$M is an analog-to-digital converter circuit which takes an analog value as input and encodes it into a digital value. The difference between analog and digital values is called the quantization error. The modulator uses an *integrator* to sum the error based on a feedback loop. When the accumulated error reaches a certain threshold, the quantizer switches the value of the output. The *order* of the modulator is given by the number of integrators it uses. These integrators and feedback loop perform noise shaping that moves most of the quantization error out of the frequency range of interest. Thus, higher-order modulators can achieve a desired resolution with a lower sample rate (or higher resolution with the same sample rate). However, they can be *unstable*. One form of this instability is that the integrator output values can exceed their specified range. Such saturation can compromise the quality of the analog-to-digital conversion.

Ignoring noise, PVT variation, *etc.*, an ideal model of the $\Delta\Sigma$M circuit is lin-

Method	Tool	ΔΣM	TDO	VCO	RRO	BLF	PLL	Others
Non-Formal	fSPICE			[193]	[193]			
	AMT							DDR2 [130], flash memory [163]
	Others	[43, 127, 179]	[220]				[60]	
Equivalence Checking								bandpass filter [188], NAND [188]
Discretization	AMCHECK		[97]	[187]		[97]		Schmitt trigger [189], amplifier [189], charge pump [187]
	MSCHECK	[125]					[125]	
	Others			[56]				arbiter [133]
Reachability Analysis	KRONOS							half [34], a 4-input circuit [153], XOR, 4-input AND [35]
	PHAVER		[73]	[73]				
	LEMA		[144]		[144]		[144]	integrator [144]
	D/DT	[50]				[50]		
	CHECKMATE	[91]	[91]					
Theorem Proving		[180]						telephone receiver [76], TTL [95], Colpitts oscillator [218]

Table 2.2: Verified Circuits

ear and all dynamics are monotonic functions. Therefore, the extreme value of integrators can be found by considering finite number of corner cases. Computing worst case trajectories is an optimization problem. In [50], Dang *et al.* modeled a third-order modulator as a discrete-time hybrid automaton with four variables and checked the stability problem using mixed integer linear programming. Their results show that the modulator circuit is stable up to 30 steps for some input signals and initial states. In [91], Gupta *et al.* studied the same problem using a reachability analysis approach by applying a discrete-time version of the CHECKMATE tool. They reported initial conditions and input values under which saturation levels can be reached. In [222], a method was presented for bounding the state variables of a second order $\Delta\Sigma$M. The method uses a piecewise-affine equation to model the circuit, employs polygons to represent circuit states, and finds an invariant set in state-space using either an analytic or algorithmic approach. Symbolic methods have been applied to prove the stability of the circuit or find counterexamples in [180]. The modulator circuit has been used as an example of many simulation-based methods. For example, Sammane *et al.* applied their symbolic monitoring algorithm [179] to the third-order modulator and Jesser *et al.* applied assertion-based verification to the first order modulator. In [43], Clarke *et al.* applied their random input generator to the third-order modulator and measured the probability of stability failure within a bounded time.

2.3.2 A Tunnel Diode Oscillator

Figure 2.3 shows the tunnel-diode oscillator (TDO) circuit that has been studied by many verification researchers. It consists of a capacitor, an inductor, and a tunnel diode. An ideal LC oscillator can oscillate forever if the initial current through inductor or voltage across the capacitor is non-zero. However, the parasitic resistance of the inductor makes the oscillation decay exponentially. The tunnel diode compensates for the parasitic resistance because it has negative resistance characteristics at low voltages. As shown in Figure 2.4, the current through the tunnel diode decreases as the voltage is increased in its negative resistance region.

Oscillation conditions of oscillator circuits refer to all conditions that ensure the circuit oscillates stably. These conditions include circuit parameters and ini-

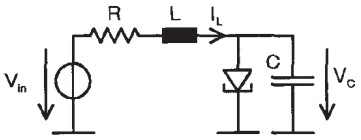


Figure 2.3: Tunnel Diode Oscillator

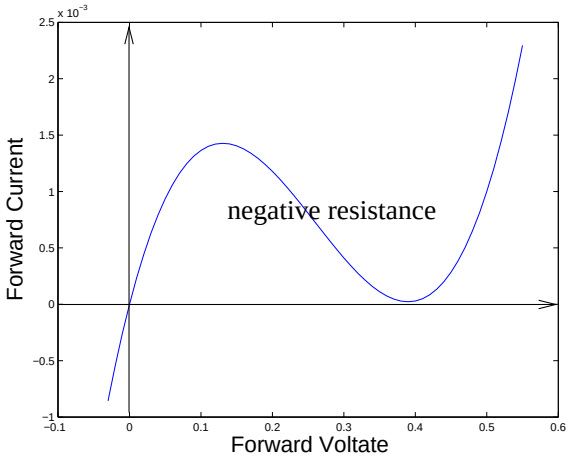


Figure 2.4: Tunnel Diode's I-V Characteristic

tial node voltages. The circuit can be modeled by a two-variable system, *i.e.*, the voltage V_c across the capacitor and the current I_l through the inductor. All DC equilibrium points can be computed based on the I-V characteristic of the tunnel diode and the inductor. Then two simulations can bound the stable oscillation region because trajectories cannot cross and there is no chaotic behaviors in a two-dimensional system. Figure 2.5 shows the equilibrium point, the stable orbit and two simulations: a trajectory spiraling out from near the equilibrium point, and one spiraling in from outside of the stable orbit.

Reachability regions of the TDO circuit have been computed by AMCHECK [97], PHAVER [73], CHECKMATE [91], and LEMA [144]. The stable oscillation property can be verified by checking whether the current I_l cycles above an upper bound and below a lower bound periodically. In [97], Hartong *et al.* applied AMCHECK to find an invariant set that contains the oscillation orbit. On the other hand, CHECKMATE combined reachability analysis over finite time horizon and Lyapunov function to show the stability over infinite time [91]. A condition where the oscillation may die out was also found. In [220], Zaki *et al.* checked the property by monitoring circuit behaviors generated by an interval based simulator. HYTECH cannot complete the verification due to arithmetic overflow errors [144]. Two further properties were studied by Frehse *et al.* [73]. They used a monitor automaton to

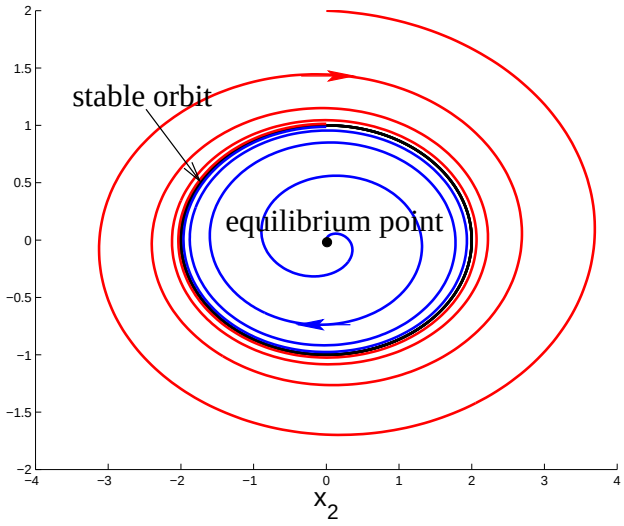


Figure 2.5: Simulations of the Tunnel Diode Oscillator

measure cycle amplitude variations and period jitters during the reachability computation in PHAVER. However, this makes the reachability computation much slower ($> 20x$) and there is no jitter in the simple circuit model which does not include noise. A forward/backward refinement strategy [73] was used to reduce the memory usage, but this increases the running time further.

2.3.3 Voltage Controlled Oscillators

A voltage controlled oscillator (VCO) is an oscillator whose oscillation frequency is controlled by a voltage input. Two kinds of VCOs have been considered in the formal verification literature: differential, LC VCOs (Figure 2.6, Figure 2.7) and RC ring VCOs (Figure 2.8, Figure 2.9). The differential VCOs are constructed using a parallel inductor and capacitor combination whose impedance is very large at the resonant frequency. The MOSFET transistors form two inverters in a positive feedback loop, and the circuit will oscillate at a frequency close to the resonant frequency of the inductor and capacitor if the gain of the inverters at this frequency is sufficient to overcome the parasitic losses of the rest of the circuit. The capacitances are controlled by an input voltage which changes the resonant frequency and hence the oscillation frequency. The ring VCOs are based on a ring with an

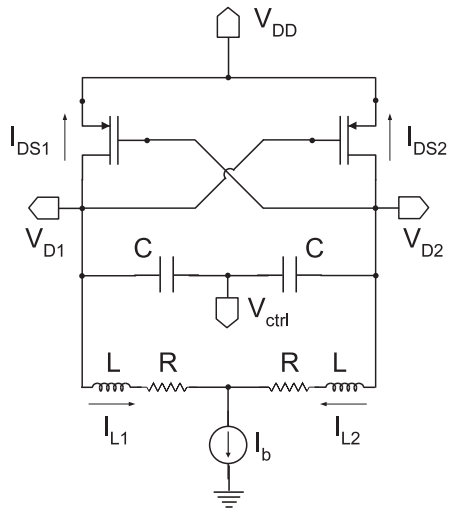


Figure 2.6: A Differential VCO Circuit (Fig.3 of [73])

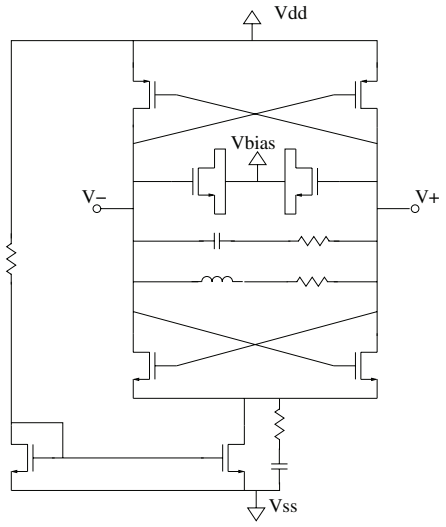


Figure 2.7: A RF VCO Circuit (Fig.13 of [193])

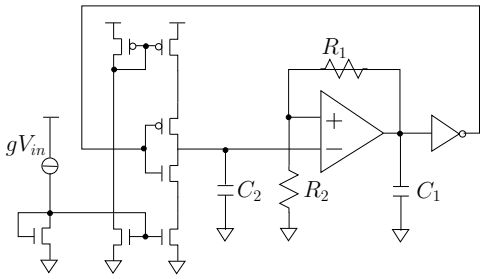


Figure 2.8: A Opamp-Based VCO Circuit (Fig.4 of [187])

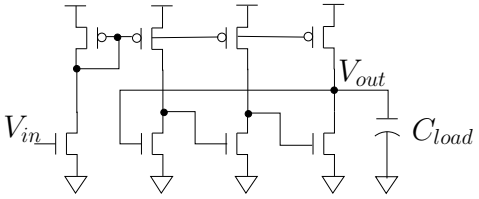


Figure 2.9: A Ring VCO Circuit (Fig.3 of [56])

odd number of inverters. The oscillation frequency is controlled by changing the delay of the inverters by adjusting their output current.

The differential VCO as shown in Figure 2.6 has three continuous variables[8] and its dynamics is less contractive than the TDO circuit. Therefore, the reachability computation of this VCO circuit is more challenging. For example, PHAVER failed to find an invariant set because of large over-approximation error [73]. To

[8]There are four variables: V_{D_1}, V_{D_2}, I_{L_1} and I_{L_2} in Figure 2.6. Noting the sum of I_{L_1} and I_{L_2} is a constant value I_b, the circuit can be modeled by three-state equations.

solve the problem, Frehse *et al.* developed an abstraction-refinement technique which performs forward as well as backward reachability and refines models in a smaller region if the verification fails. FSPICE has been applied to simulate a similar differential VCO as shown in Figure 2.7 and analyze the oscillation frequency by periodic steady state analysis. Discretization methods have been applied to ring VCOs. In [187], Steinhorst *et al.* applied their AMCHECK tool to compute the period of the VCO circuit as shown in Figure 2.8. Similarly, Dastidar *et al.* applied discretized models and showed that the circuit in Figure 2.9 oscillates when parameter values such as transistor widths have appropriate values.

2.3.4 A Biquad Lowpass Filter

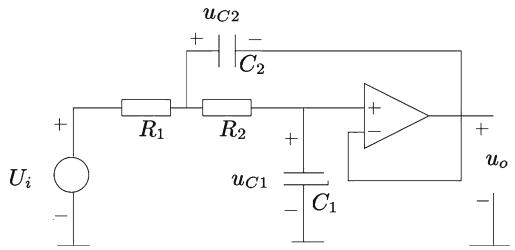


Figure 2.10: A Second Order Biquad Lowpass Filter

Figure 2.10 shows a biquad low-pass filter which consists of an operational amplifier and a feedback capacitor. The dynamics of the circuit are not linear, thus, nonlinear models, corresponding to saturating the operational amplifier output voltage and current capabilities, must be applied in order to accurately analyze the circuit. However, from the transfer function of the circuit, it is easy to see that the circuit passes low-frequency signals but attenuates signals with frequencies higher than its cutoff frequency.

One property that has been studied is the absence of overshoot in the filters C_1, C_2, *i.e.*, u_{C1}, u_{C2} never exceed their steady state values. This property can be verified by computing reachable regions of the circuit and checking the bounds of u_{C1} and u_{C2}. In [97, 98], Hartong *et al.* used a linear operational amplifier with finite gain and unlimited bandwidth. They partitioned the entire three-dimensional (*i.e.*, u_i, u_{C1}, u_{C2}) state space into boxes and computed transitions between them by

the AMCHECK tool. The method has been applied to a highly damped filter circuit and a less damped one. The result shows that the value of u_{C1} remains in a specified range (*e.g.*, $\pm$ 2 V) in the highly damped case whereas it reaches higher levels in the other case. In [24, 50], Dang *et al.* checked the same property of the filter using their D/DT tool in a more efficient way. The circuit is modeled by differential algebraic equations (DAEs). A hyperbolic tangent function was used to model the non-linearity of the operational amplifier output voltage. They introduced the output voltage in to the equations but treated the input voltage as a parameter. Therefore, their model has three variables (*i.e.*, u_o, u_{C1}, u_{C2}). To compute reachable regions, DAEs are transformed to the semi-explicit form which can be solved by combining projection and reachability computations for ODEs. Nonlinear ODEs are transformed automatically to a piecewise-affine dynamics using the hybridization techniques (as described in Section 2.2.4).

Summary

As described above, existing verification tools have been applied to a variety of simple circuits, typically with two- or three-dimensional state spaces, and with linear or quasi-linear models. Some of the circuits, such as the tunnel-diode oscillator, are not practical for implementation on VLSI chips. The properties that have been verified are simple, time-domain properties of the circuit, that in many cases are straightforward to prove manually or with a few simulations. The models used in the published results are very abstract and do not capture the key behaviors of VLSI circuits. For example, the $\Delta\Sigma$M circuit is modeled as a discrete-time system which excludes important phenomena such as noise, and PVT variations. Usually, these examples only show that their methods or tools "work" for simple demos but fail to show that they can verify properties that actually matter. Therefore, there is still a huge gap between the ability of available formal verification tools and complex circuits and properties that could benefit from formal methods.

2.4 Prior Research of COHO

Development on COHO started several years before I entered the Ph.D. program. Many of the initial ideas for using reachability analysis to verify circuits were de-

veloped by Mark Greenstreet. This led to the idea of using projection-based reachability analysis which was developed by Mark Greenstreet and his students from 1997 to 2003. I joined this effort with my M.Sc. research in 2005 where I implemented numerically robust methods for COHO's linear program solver. This re-engineering finally provided COHO with the robustness needed to be used with some hybrid system examples and enabled the research described in this dissertation. The remainder of this section gives a more detailed description of the work on COHO that preceded my Ph.D. research.

As described in Section 2.1.2, Kurshan and McMillan published the first paper reporting the formal verification of circuits using ODE models. This was followed by a series of papers [83–85] that introduced the idea of using Brockett's annuli as the basis for abstraction mappings from continuous to discrete signals, and gave the first examples of projection-based representations to obtain a tractable representation of moderate-dimensional objects. In [83], these projections were onto rectilinear polygons. For each bloated face, *i.e.*, a hyper-rectangle, the maximum outward derivative was computed easily as the first-order model is a convex function[9]. A fourth-order Runge-Kutta integrator was used to integrate ODEs to move forward each face. The toggle circuit that we examine in Section 3.2.1 was verified as a proof-of-concept example. Manual model reduction was employed to obtain a three-dimensional system by ignoring capacitances at some nodes and rewriting the ODEs to be an integration with respect to the input clock voltage rather than time.

The use of rectilinear projectagons in [83] resulted in large over approximations that prevented the verification of other circuits. In 1997, Greenstreet and Mitchell proposed a generalization of the projection-based method to use general, non-convex polygons. These ideas were implemented in the first version of COHO [86] where the numerical integration and LP solver were implemented in MATLAB, and the geometric operations were implemented in JAVA. They manually constructed circuit models using the modified nodal analysis techniques described in Section 3.3. The tool was demonstrated using some simple examples [87]. However, it was observed that the linear programs associated with projectagons are

[9]This motivates the idea of *face lifting* and *orthogonal polyhedra* of D/DT. However, orthogonal polyhedra do not use the projecting idea and limit vertices of rectilinear polygons as fixed grid points.

often highly ill-conditioned, and this prevented the use of COHO on more complicated examples.

In 2000, Greenstreet, Laza, and Varah recognized that the structure of linear programs associated with projectagons can be exploited to produce an efficient and robust implementation of the Simplex algorithm. Laza developed these ideas in his M.Sc. thesis [142] including the error analysis of the algorithm and a preliminary implementation in COHO. However, Laza's implementation only handled uses of the LP solver for optimization, but did not support projection operations. In the summer of 2003, Karen Brennan studied the COHO codes and wrote a proposal for using Laza's algorithms for all linear programming based operations.

From 2005 to 2006, I re-engineered the COHO software and solved the numerical problems of COHO to make it more robust [208]. Interval arithmetic was applied to both the LP solver and the geometry computation engine. The interval-based LP solver [214] overestimated the optimal value and was also integrated with a new projection function. However, the LP solver may not find the correct optimal basis if the LP is ill-conditioned, which may generate large approximation errors in the projection operations. These enhancements made it possible to apply COHO to two hybrid system examples [208].

To summarize, the work on COHO preceding my Ph.D. research established the basic algorithms for projectagon based reachability analysis and showed that these methods had promise for circuit verification. However, many questions needed to be answered and problems needed to be solved to show that these methods are useful in practice. First of all, we needed a systematic way of specifying the properties that we wanted to verify about circuits. Next, trying these methods on real circuits revealed places where COHO's algorithms introduced unacceptable approximation errors. By revising these algorithms, I have reduced the errors to enable verification of real circuits. In some cases, this required developing new data structures and algorithms to replace or complement those used in the original COHO. Finally, the run-time for COHO's algorithms tended to be very large, limiting the application of the tool. I made many improvements to the efficiency of COHO by reformulating the details of the reachability problem, developing new algorithms, and tuning the existing implementation. These contributions are described in the remaining chapters of this dissertation.

2.5 Summary

This chapter explored formal methods of analog verification, especially reachability analysis, and their application to circuit examples. This provides background for the methods and tools in this dissertation: Section 2.1 describe the main approaches used in prior work on circuit verification: equivalence checking, model checking and theorem-proving based approaches. We argued that reachability analysis is the most promising of these approaches, and Section 2.2 described the various algorithms and data-structures that have been used in reachability analysis tools. Then, Section 2.3 described circuits and their properties that have been verified using these techniques. The circuits that have been considered have been very simple, and the properties that have been verified often reflect the capabilities of the tools more than the concerns of designers.

Verifying relevant properties of practical circuits will require capabilities that are well beyond those of current tools. High-dimensional (*e.g.*, > 4) regions should be represented and manipulated efficiently because most practical analog circuits have tens to hundreds of nodes. Existing methods use representations that either have expensive operations (*e.g.*, polytopes) or have large approximation error (*e.g.*, hyper-rectangles). Because many circuits have highly non-convex reachable regions, it is important to support non-convex regions otherwise the wrapping effect will often generate false-negative results. It is necessary to support nonlinear dynamics because ODEs extracted from circuits are neither linear nor even weakly non-linear. Most current methods are very expensive (*e.g.*, representation methods do not scale) or have large approximation errors (*i.e.*, they only work for models with linear or nearly-linear dynamics). Furthermore, tight approximations are required to verify complicated systems without false-negatives. Therefore, we develop an efficient, robust, and accurate reachability analysis tool for moderate-dimensional, nonlinear hybrid systems which will be presented in Chapter 4.

In addition to a powerful reachability computation tool, it is also important to convert circuit verification problems to reachability analysis automatically. This includes constructing mathematical models from circuit netlists and formally specifying analog properties. Although there exist many techniques for modeling and specification as shown in Section 2.2.1 and Section 2.2.2, most available tools do

not support circuit verification tasks directly, *i.e.*, users have to create mathematical models for circuits and check properties manually. Therefore, we propose a general solution to model and specify AMS circuits in Chapter 3.

3

Circuit Verification as Reachability

This chapter describes our method for translating circuit verification problems to reachability analysis problems. Section 3.1 presents the phase-space representation of circuit behaviours and our verification strategy. Section 3.2 provides a set of real circuits that we use as a testbench suite for verification methods. Section 3.3 describes how to model circuits as ODE systems and Section 3.4 presents our methods to specify analog signals and properties. Finally, some implementation issues are discussed in Section 3.6.

3.1 Phase Space and Reachability Based Verification

Circuit simulators usually present circuit behaviours to designers as waveforms, *i.e.*, signal voltages as functions of time. The *phase-space* representation provides another view of circuit behaviors which is well-suited for use in formal verification. In the phase-space representation, each circuit state is represented by a unique point, and the set of all possible states is represented as a high-dimensional region. For most of the circuits that we consider, the state of the circuit can be represented by the voltage on each node of the circuit. Thus, the dimensionality of the phase space is the same as the number of nodes for these circuits. For example, an inverter circuit has an input node and an output node, thus its phase space is

two-dimensional as shown in Figure 3.2.

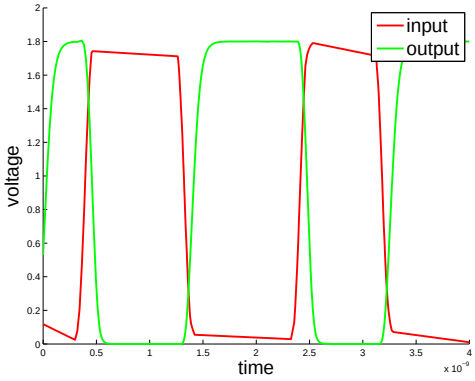


Figure 3.1: Waveforms of Inverters

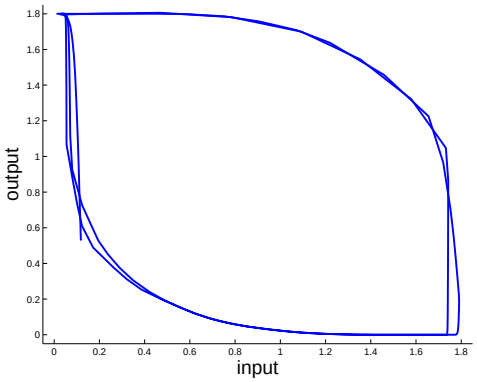


Figure 3.2: Phase-Space View

Based on the phase-space representation, many verification problems can be solved as reachability analysis problems. For example, a safety property of a circuit can be verified by exploring the entire reachable region of the circuit and demonstrating that the property holds everywhere in this region. Our verification strategy consists of the following four steps:

1. Construct a mathematical model, *i.e.*, ODEs as described in Section 3.3, for the circuit to be verified.

2. Formally specify properties to be checked using an extension of LTL for continuous behaviours as described in Section 3.4.

3. Apply our reachability analysis tool COHO (see Chapter 4) to compute an overapproximation of the entire reachable space of the circuit.

4. Check the specification on the reachable space as computed (*i.e.*, over approximated) above.

This chapter focuses on how to formally model circuit systems and specify analog properties. Steps 3 and 4 will be discussed in Chapter 4 and Chapter 5.

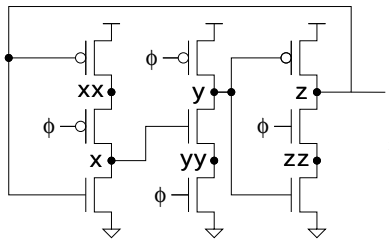
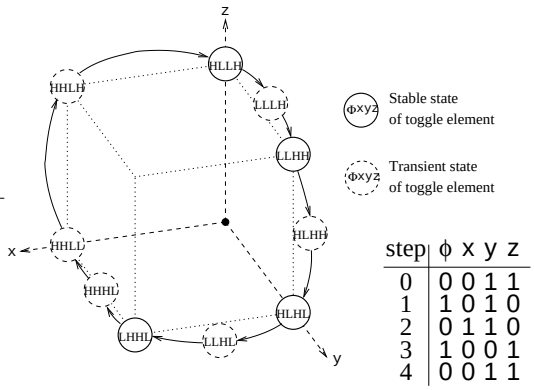


Figure 3.3: Toggle Circuit



step	ϕ	x	y	z
0	0	0	1	1
1	1	0	1	0
2	0	1	1	0
3	1	0	0	1
4	0	0	1	1

Figure 3.4: State Transition Diagram

3.2 Circuit Examples

In this dissertation, we use four circuits as examples of circuit verification problems. Our testbench includes a toggle circuit, a flip-flop, an arbiter and a ring oscillator. It covers synchronous circuits (toggle and flip-flop), asynchronous circuits (arbiter) and analog circuits (oscillator). It also contains circuits with no inputs (oscillator), one input (toggle), two inputs that can change at arbitrary time with respect to each other (arbiter) and two inputs whose transitions must occur according to a specified timing relationship (flip-flop).

3.2.1 The Yuan-Svensson Toggle

Figure 3.3 shows a toggle circuit that was originally published by Yuan and Svensson [217]. The operation of this circuit can be understood by using a simple switch model starting from a state where the ϕ input is low. In this case, y is driven high; z is floating; and x is the logical negation of z. Figure 3.4 shows the state transition diagram for the toggle starting from the state where z is high when ϕ is low – the other case, with z high, is reached on step 2 of the figure. Note that from step 2 to 3 in the figure, all three of x, y and z change values. This is a critical race for the toggle. As a consequence, if the rise time or fall time for ϕ are too large, the toggle will fail.

We specify the behavior of the toggle as a safety property. In particular, there is an invariant subset of $\mathbb{R}^d$ such that all trajectories in this set have a period twice

that of the clock signal. Another property is to show that the input clock and output signal satisfies the same specification in order to construct a ripple-counter.

3.2.2 A Flip-Flop

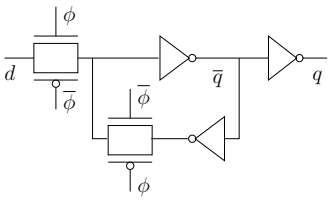


Figure 3.5: Latch Circuit

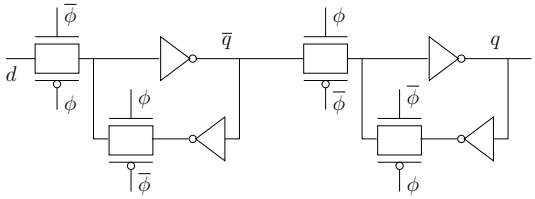


Figure 3.6: Flip-Flop

A latch is a circuit that has two stable states and thereby is capable of serving as one bit of memory. Figure 3.5 shows a static, transparent *pass-gate latch* [207, Chapter 7.3]. The output of the latch circuit holds its old value when the clock signal ϕ is low and is set to the value of the input when ϕ is high. Flip-flops are typically implemented as master-slave devices. Figure 3.6 shows a D flip-flop which consists of two latches. The flip-flop only updates its output value on the rising edge of the clock signal ϕ. However, the output value may not be stable if the input signal changes on the rising edge of ϕ.

The input specification of the flip-flop requires that the input value cannot change during the rising transition of the clock signal ϕ. The output of the flip-flop must guarantee that the output signal is stable within a specified time after the rising clock edge. Other interesting properties includes the maximum clock frequency and the lower (upper) bounds of the clock-to-q delay.

3.2.3 An Arbiter

An arbiter is a circuit that provides mutually exclusive access to a resource for some set of clients. We consider an asynchronous arbiter with two clients as shown in Figure 3.7(a). The two clients interact with the arbiter using a four-phase handshake protocol: client i raises r_i to request the privilege; the arbiter raises g_i to grant client i the privilege; when the client is done with the privilege, it lowers r_i; and finally the arbiter lowers g_i to complete the handshake.

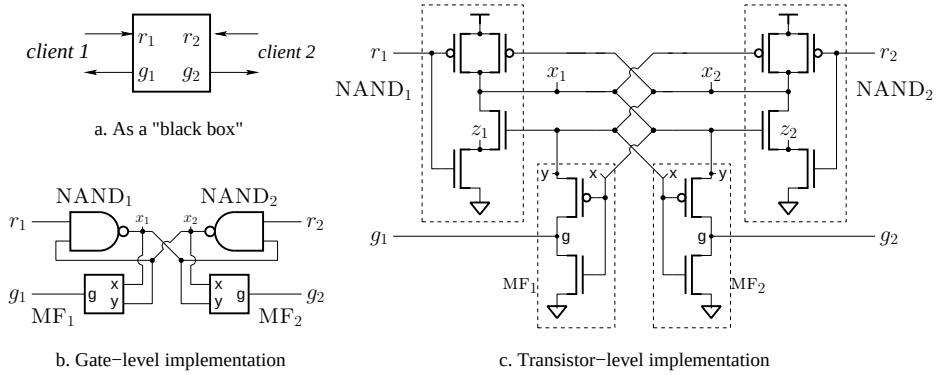


Figure 3.7: Arbiter Circuit

Figure 3.7(b) shows an implementation of an arbiter based on a SR-latch using a pair of cross-coupled NAND gates (see [155, Fig. 5]). As illustrated in Figure 3.8, when r_1 and r_2 are both low, the NAND gate outputs, x_1 and x_2 are both driven high, and the *metastability filters* (to be described shortly), MF$_1$ and MF$_2$ will drive the grant signals g_1 and g_2 low. If request r_1 is asserted (i.e. driven to a high value), then the x_1 will go low and g_1 will go high to grant the request. If r_2 makes a request while r_1 holds the grant, then the request from r_2 will be blocked. In particular, x_2 will remain high even when r_2 goes high because x_1 is low. When the first client drops, r_1, x_1 will go high and g_1 will go low; if there is a pending request on r_2, then x_2 will go low in response to the rising of x_1, and this will cause g_2 to go high. The arbiter operates in an analogous manner if r_2 is asserted well-before r_1.

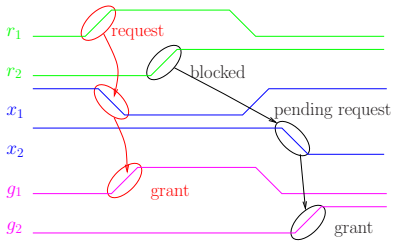


Figure 3.8: Uncontested Requests

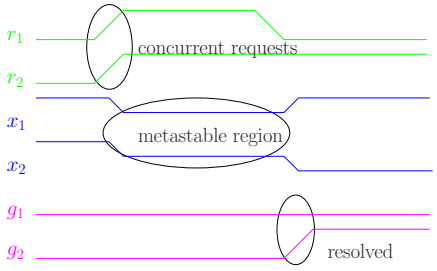


Figure 3.9: Contested Requests

As shown in Figure 3.9, if both requests, r_1 and r_2 are asserted at roughly the

same time, then signals x_1 and x_2 will both start to fall. This results in a falling input for each NAND-gate and the consequent possibility of metastability if the two NAND-gates reach a balance point with their outputs at an intermediate level between the power supply voltage and ground. This condition can persist for an arbitrarily long time [38, 154, 183] but eventually resolves (with probability one) to a state with one of x_1 or x_2 going high and the other going low. The metastability filters (MF_1 and MF_2 in Figure 3.7(c)) prevent the outputs of the arbiter from changing until metastability resolves.

The metastability filter is a modified inverter. Consider circuit MF_1. The gates of the transistors in the inverter are connected to x_1. Unlike a traditional inverter, the source of the PMOS pull-up is connected to x_2. With this configuration, the pull-up transistor remains in cut-off until x_1 is at least the PMOS threshold voltage below x_2. This prevents g_1 from moving any significant amount above ground until client 1 has clearly won the arbitration.

The arbiter must satisfy the handshake protocol and guarantee *mutual exclusion, i.e.,* signals g_1 and g_2 may not both be high at the same time. Obviously, it would be desirable if the arbiter were guaranteed to eventually issue a grant when a request is pending. It is well known that a real arbiter cannot satisfy this requirement along with the safety requirements described above (see, for example, [154, 157]).

3.2.4 The Rambus Ring Oscillator

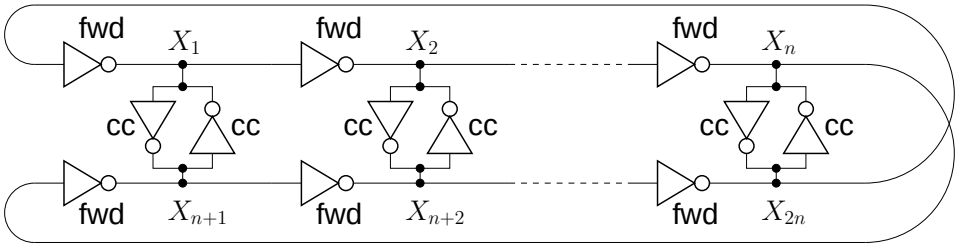


Figure 3.10: The Rambus Ring Oscillator

Figure 3.10 shows a differential ring oscillator. The oscillator consists of an even number of stages, n; and each stage has two *forward* (labeled fwd in the figure)

inverters connected by a pair of *cross-coupling* (labeled cc) inverters. Consider an initial state where x_i is high if i is odd and $i < n$, or i is even and $i > n$, and x_i is low otherwise. As shown in Figure 3.11, if the forward inverters are strong enough to overpower the cross-coupled inverters, then x_i will be excited to make a high-to-low transition and x_{n+1} will be excited to make a low-to-high transition. This will lead to a state where signals x_2 and x_{n+2} are excited to change, and so on. The cross-coupled inverters ensure that the rising and falling transitions happen at roughly the same time. In particular, if x_i transitions earlier than x_{n+i}, then the inverter from x_i to x_{n+i} will accelerate the transition of x_{n+i}, while the inverter from x_{n+i} to x_i will retard the transition of x_i.

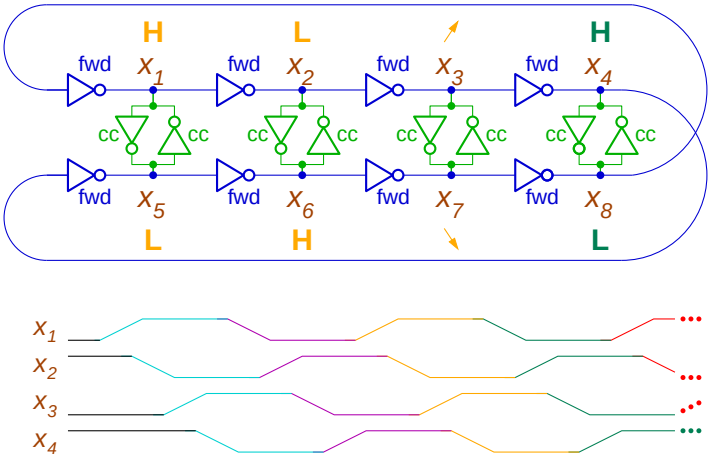


Figure 3.11: Expected Oscillation Mode

Researchers reported that some implementations of the circuit did not oscillate as described above in real, fabricated chips. They posed a verification problem of showing that the oscillator starts from all initial conditions for a particular choice of transistor sizes. Furthermore, they posed an additional problem of determining the sizes of the inverters that guarantee that the circuit will enter a stable oscillating condition from any initial condition.

The failure occurred when the forward and cross-coupling inverters do not have comparable strength. If the forward inverters are much larger than the cross-coupling inverters (as shown in Figure 3.12), then the circuit acts like a ring of $2n$

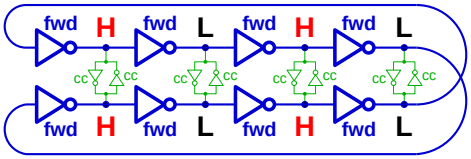


Figure 3.12: Forward Inverters Too Large

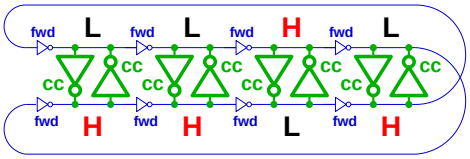


Figure 3.13: Cross-Coupling Inverters Too Large

inverters and will settle to one of two states:

$$\begin{array}{lll} \text{State 1:} & x_1, x_{n+1}, x_3, x_{n+3}, \cdots, x_{n-1}, x_{2n-1} & \text{are high, and} \\ & x_2, x_{n+2}, x_4, x_{n+4}, \cdots, x_n, x_{2n} & \text{are low.} \\ \text{State 2:} & x_1, x_{n+1}, x_3, x_{n+3}, \cdots, x_{n-1}, x_{2n-1} & \text{are low, and} \\ & x_2, x_{n+2}, x_4, x_{n+4}, \cdots, x_n, x_{2n} & \text{are high.} \end{array} \tag{3.1}$$

Conversely, if the cross-coupling inverters are much larger than the forward ones (as shown in Figure 3.13), then the circuit acts like n separate static latches and has 2^n stable states.

3.3 Modeling Circuits as ODE Systems

This section describes how we obtain ODE models from circuit schematics (or equivalently, netlists). Section 3.3.1 describes the qualitative operation of n-channel and p-channel MOSFETs, the basic building block of most integrated circuits. We then describe our use of modified nodal analysis to derive ODEs from the circuits; this process is essentially the same one that is used by most circuit simulators. Device models for state-of-the art processes are very complicated with a large number of parameters. Rather than implementing these models in our codes, we use a table-driven approach described in Section 3.3.2. The tables are generated using standard circuit simulators. Thus, we use the same models for formal verification as the circuit designers are using for simulations.

3.3.1 Circuit Models

We construct mathematical models based on nonlinear ODEs from circuit netlists. The circuits that are used on chips are typically composed of transistors, capaci-

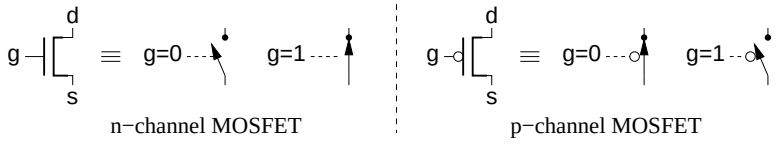


Figure 3.14: Transistors: Switch-Level Models

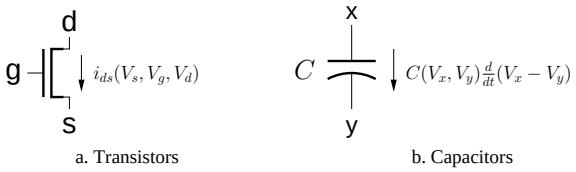


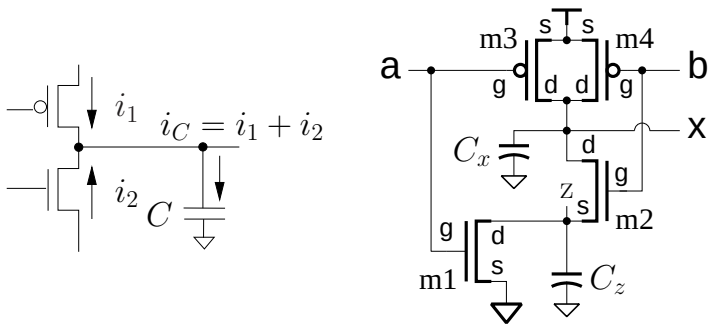
Figure 3.15: Device Models

tors and resistors. For simplicity, we will not consider inductors or diodes in this dissertation.

For CMOS technologies, transistors come in two main types, n-channel and p-channel, as shown in Figure 3.14. Both types of transistors have three terminals marked s (the "source"), g (the "gate") and d (the "drain") in the figure[1]. For digital circuits, a transistor can be viewed as a voltage controlled switch. The behavior of a n-channel (*resp.* p-channel) transistor is like that of a switch that makes a connection between the source and drain when the gate is high (*resp.* low) and does not make a connection when the gate is low (*resp.* high). However, we model transistors as voltage-controlled current sources for circuit-level behaviors. That is, the drain-to-source current of a transistor is a function of the voltages of its terminals. As shown in Figure 3.15(a), we write $i_{ds}(V_{sgd})$ to denote the current function of transistors, where V_{sgd} is the vector of node voltages.

A capacitor is a device that holds a charge that is a function of the voltages of its terminals as shown in Figure 3.15(b). Current is the time derivative of charge which yields $i_C = C(V_{x,y})\frac{d}{dt}V_{x,y}$, where C is the capacitance of the capacitor and V_{xy} is the vector of terminal voltages. While on-chip capacitors can be between

[1]There is actually a fourth terminal, the "body" or "channel" that corresponds to the silicon substrate or well in which the transistor is fabricated. For simplicity, we assume that all n-channel transistors have their bodies connected to ground and all p-channel transistors have their bodies connected to V_{dd} (i.e. the positive voltage from the power supply).



a. KCL b. NAND

Figure 3.16: Kirchoff's Laws

arbitrary pairs of nodes and have capacitances that depend on node voltages, for simplicity we model capacitances as being to ground and having constant values independent of node voltages. Likewise, we omit resistances for simplicity and brevity but note that they can be treated as voltage-controlled current sources in a manner analogous to our treatment of transistors described above.

Given models for all devices of a circuit, we construct a nonlinear ODE model for the whole system using standard nodal analysis techniques based on Kirchoff's current law (KCL). As depicted in Figure 3.16(a), KCL states that the sum of all currents flowing into a node must be equal to zero. Let V be the vector whose elements are the voltages of nodes of the circuit of interest, which consists of V_i for input nodes and V_o for internal and output nodes. Let I_{ds} be a function from vectors of node voltages to vectors of transistor currents. Let n be the number of nodes of the circuit and m be the number of transistors. Let $M \in \mathbb{R}^{n\times m}$ be the connectivity matrix for transistors to nodes:

$$M(i,j) = \begin{cases} +1, & \text{if the source of transistor } j \text{ is connected to node } i \\ -1, & \text{if the drain of transistor } j \text{ is connected to node } i \\ 0, & \text{otherwise} \end{cases}$$

Finally, let C be the diagonal matrix where $C(i,i)$ is the capacitance from node i to ground. We now have:

$$\frac{d}{dt}V_o = C^{-1}M\, I_{ds}(V). \tag{3.2}$$

This gives us the time derivative function for all internal and output nodes of the circuit we wish to verify. The ODE models for external inputs are determined using a Brockett's annulus abstraction which will be described in Section 3.4.3.

Let us take the NAND gate shown in Figure 3.16(b) as an example. In this circuit, we have $V = [V_{ss}, V_{dd}, V_a, V_b, V_x, V_z]$ and $I_{ds} = [i_{m1}, i_{m2}, i_{m3}, i_{m4}]$. By KCL, we obtain:

$$\begin{aligned} -i_{m3}(V_{dd}, V_a, V_x) - i_{m4}(V_{dd}, V_b, V_x) - i_{m2}(V_z, V_b, V_x) - C_x \tfrac{d}{dt} V_x &= 0 \\ i_{m2}(V_z, V_b, V_x) - i_{m1}(V_{ss}, V_a, V_z) - C_z \tfrac{d}{dt} V_z &= 0, \end{aligned}$$

which yields

$$\begin{aligned} \tfrac{d}{dt} V_x &= -\tfrac{1}{C_x}(i_{m3}(V_{dd}, V_a, V_x) + i_{m4}(V_{dd}, V_b, V_x) + i_{m2}(V_z, V_b, V_x)) \\ \tfrac{d}{dt} V_z &= -\tfrac{1}{C_z}(i_{m2}(V_z, V_b, V_x) - i_{m1}(V_{ss}, V_a, V_z)). \end{aligned}$$

When constructing the ODE model, we impose the physically realistic requirement that all circuit nodes have some non-zero capacitance to ground, which ensures that the diagonal matrix C is positive definite and therefore invertible. These assumptions avoid complexity while retaining the key features of realistic circuits. More generally, the procedure we described above provides a general way for a verification researchers to obtain a system of non-linear ODEs from a schematic diagram; in other words, a schematic is simply syntactic sugar for a system of ODEs according to this interpretation.

3.3.2 Circuit-Level Models Based on Simulations

We now describe how we obtain the i_{ds} functions. Of course, there are plenty of textbooks that present models for transistors (for example [123, 207]). However, these are simplifications of the models that are used in industry for real fabrication processes. These simplified models provide designers with intuition to understand qualitative circuit behaviors, but they do not provide the accuracy to verify detailed behaviors. For the latter, designers rely on the models used by commercial circuit simulators such as HSPICE or Spectre [1]. In other words, circuit simulators provide the interface between device modeling experts and circuit design experts. In the same manner, we did not write our own code to implement state-of-the-art

transistor models. Instead, we use the HSPICE simulator to obtain tables of i_{ds} data on a relatively fine grid and use bilinear (or other kinds of) interpolation to compute transistor currents. The key advantage of this approach is its simplicity and generality. We can generate accurate models for any process with vendor provided SPICE models. Furthermore, our table is interval value based, that is, the lower and upper bounds of the current are provided for each grid point. COHO generates differential inclusions from the ODE circuit models; thus, we can incorporate the intervals from the tables in the differential inclusion to ensure that our verification includes the exact SPICE model in the set of behaviors that it considers. Furthermore, we can use wider intervals in the table to model PVT variations, add error bounds to the device models themselves, *etc.*

Compared with other simulation-aided verification techniques such as the one used in LEMA [148], our approach has the benefit that we only simulate a small number of basic devices, *e.g.*, transistors, instead of the whole circuit or macro-models. Therefore, it is much easier to simulate more corner cases in order to obtain a higher coverage. Our table-based method can be applied similarly to obtain accurate models of circuit capacitance, resistance or inductances. It has not been implemented but it is an obvious future work.

We extended our table-based approach to create macro models for small circuit blocks (see Section 5.4 for an example). However, the number of grid points increases exponentially as the number of device terminals increases. For example, the tables for transistors have three indices, one each for the source, gate and drain voltages, and we sampled these on a relatively fine grid of 0.01V for the 180nm CMOS process. This means that each table has $181^3 \approx 6 \times 10^6$ entries. A four-dimensional table using the same grid would have about 10^9 entries which would consume a prohibitively large amount of memory and cause high cache miss-rates. Therefore, we developed a *polynomial table* technique which uses a coarser grid and uses a higher-order polynomial to approximate the current function with acceptable errors. We found that second-order polynomials generally provided sufficient accuracy for our verification work, thus each entry of the table holds the coefficients for the quadratic polynomial model. Using this technique, the table with a 0.1V grid only holds roughly 10^6 values for a four-terminal device[2], which

[2] A three-variable quadratic has ten coefficients, and a table with a 0.1V grid has roughly $19^4 \approx$

is quite practical.

3.4 Specification

Section 3.3 described how to construct a mathematical model for a circuit. This section describes our approach for specifying circuit properties. We extend linear temporal logic (LTL) to continuous time and values to specify analog properties. Atomic propositions in this logic correspond to regions of the circuit's state space. A particular class of such regions that is of interest to our work are the regions that can be specified using Brockett's annulus construction – such regions correspond to the intuitive notion of a signal being "digital".

3.4.1 Extended LTL

To specify circuit-level properties of circuits, we generalize conventional temporal logics to formulas with continuous time and values. Our specification language is based on *linear-time temporal logic (LTL)* [168] rather than CTL for simplicity because we are mainly interested in safety properties and only use universal quantifiers. The approach to support dense time is very similar to MITL [16] as described in Section 2.2.2. We then introduce continuous variables to describe continuous trajectories similar to the approach used in STL/PSL [150]. The novelty of our method is that we introduce probability into the logic for specifying important circuit properties such as metastable behaviors.

Given a set of states S, we call p an *atomic proposition* if it is a function from states S to booleans (*i.e.,* $p : S \to \{0,1\}$). A *trace* t is an infinite sequence of valuations in S: $t = t_1 t_2 \cdots$, where $\forall i \in \mathbb{R}^+, t_i \in S$. We say p is a LTL formula that is satisfied for trace t if $p(t_1)$ is true. If p and q are LTL formulas, then $\neg p$ is the formula that holds iff p does not hold, and $p \wedge q$ is the formula that holds if both p and q holds. Other boolean operators are just syntactic sugar for combinations of $\neg$ and $\wedge$. Thus, we will use $p \vee q \equiv \neg(\neg p \wedge \neg q)$ and $p \Rightarrow q \equiv \neg p \vee q$ for disjunction and implication.

Based on atomic propositions and logical operators defined above, temporal operators of the discrete LTL are defined as:

10^5 entries. Therefore, there are roughly 10^6 numbers in the four-dimensional table.

$\mathbf{N}\, p$	next	p holds in the next state.
$\Box p$	always	shorthand for $p \wedge (\mathbf{N}\, \Box p)$; p holds this and all subsequent states.
$\Diamond p$	eventually	shorthand for $\neg(\Box \neg p)$; p holds in this or some future state.
$p\, \mathbf{U}\, q$	until	shorthand for $p \wedge (q \vee \mathbf{N}\, (p\, \mathbf{U}\, q))$; p holds in this state and continues to hold until a state in which q holds.
$p\, \mathbf{W}\, q$	weak until	shorthand for $(p\, \mathbf{U}\, q) \vee \Box p$; p holds in the current state and continues to hold forever or until a state in which q holds.
$p \overset{\mathbf{U}}{\Rightarrow} q$	imply until	shorthand for $p \Rightarrow (p\, \mathbf{U}\, q)$; if p holds in the current state, p will continue to hold until a state in which q holds.
$p \overset{\mathbf{W}}{\Rightarrow} q$	imply weak until	shorthand for $p \Rightarrow (p\, \mathbf{W}\, q)$; if p holds in the current state, p will continue to hold forever or until a state in which q holds.

In order to specify analog properties of circuits, we define LTL-like formulas for continuous trajectories and introduce a few basic concepts from probability theory into the logic.

The state of a circuit is represented by a d-dimensional vector of real numbers; we say that d is the *dimension* of the model. We write $\mathscr{V}$ to denote this d-dimensional state space of the circuit, and $d\mathscr{V}$ to denote the d-dimension time-derivative of the state. The circuit is modeled as a *differential inclusion*: if $A \subseteq \mathscr{V}$ and $x \in A$ then

$$\dot{x} \quad \in \quad F(A), \tag{3.3}$$

where $\dot{x}$ denotes the derivative of x with respect to time. In other words, $F : \mathscr{V} \to d\mathscr{V}$ maps regions of the state space to regions of the derivative space. By using an inclusion rather than an equation, the time derivative of the circuit is not fully determined. Deterministic circuit models (even HSPICE) are only approximations of reality, the differential inclusion model solves the problem and could model non-deterministic behavior of the environment, such as the ordering, timing, and

details of the waveform shape for input transitions. A behavior of the circuit is a function from time (the non-negative reals) to states that starts in the initial region and satisfies the derivative relation. Such a behavior is called a *trajectory*, and a circuit is characterized by the (infinite) set of trajectories allowed by its model:

$$\Phi(Q_0,F) \;=\; \{\phi : \mathbb{R}^+ \to \mathscr{V} |\; (\phi(0) \in Q_0) \;\wedge (\forall t \in \mathbb{R}^+.\; \dot{\phi}(t) \in F(\{\phi(t)\}))\}, \quad (3.4)$$

where we assume $\phi(t)$ is C^1. For our circuit models, signal voltages are bounded by simple invariants of the form $gnd \leq \mathscr{V} \leq vdd$. Therefore, $\frac{d\mathscr{V}}{dt}$ is bounded and $\dot{\phi}(t)$ is defined for all $t \geq 0$ if $\phi \in \Phi(Q_0,F)$. Our continuous model for circuits is a tuple, (Q_0,F), where $Q_0 \subseteq \mathscr{V}$ is the initial region for the model, and $F : 2^{\mathscr{V}} \to 2^{d\mathscr{V}}$ is the time-derivative relation.

We extend LTL to specify continuous behaviors. Let $\phi : \mathbb{R}^+ \to \mathscr{V}$ be a trajectory, and define

$$shift(\phi,t_0)(t) \quad = \quad \phi(t+t_0). \quad (3.5)$$

If ϕ is a trajectory and S is a continuous LTL formula (defined below), we write $\phi \models S$ iff S is satisfied by ϕ. For a model $M = (Q_0,F)$, we write $M \models S$ iff $\forall \phi \in \Phi(Q_0,F),\ \phi \models S$.

A continuous LTL formula has a set of atomic propositions $\mathscr{P}$. These propositions can correspond to subsets of $\mathscr{V}$; for $P \subseteq \mathscr{V}$, a trajectory $\phi \models P$ iff $\phi(0) \in P$. To specify properties related to variable derivatives such as the Brockett's annulus in the next section, we also support atomic propositions corresponding to subsets of $\mathscr{V} \times d\mathscr{V}$. For $P \subseteq \mathscr{V} \times d\mathscr{V}$, $\phi \models P$ iff $(\phi(0),\dot{\phi}(0)) \in P$. Our continuous-time logic has no equivalent to the next-state operator; instead, we define $\neg$, $\wedge$, $\Box$ and $\mathbf{U}$ directly on trajectories:

$$\begin{aligned}
\phi \models \neg S &\equiv \phi \not\models S && (3.6)\\
\phi \models S_1 \wedge S_2 &\equiv (\phi \models S_1) \wedge (\phi \models S_2) && (3.7)\\
\phi \models \Box_{[\mathbf{t_l},\mathbf{t_h}]} S &\equiv \forall t \in [t_l,t_h].\; shift(\phi,t) \models S && (3.8)\\
\phi \models \Diamond_{[\mathbf{t_l},\mathbf{t_h}]} S &\equiv \exists t \in [t_l,t_h].\; shift(\phi,t) \models S && (3.9)\\
\phi \models S_1 \,\mathbf{U}_{[\mathbf{t_l},\mathbf{t_h}]}\, S_2 &\equiv (\phi \models S_1) \;\wedge\; \exists t_2 \in [t_l,t_h].\; (shift(\phi,t_2) \models S_2) \wedge && (3.10)\\
&\qquad (\forall 0 \leq t_1 < t_2.\; shift(\phi,t_1) \models S_1).
\end{aligned}$$

We put subscripts on the temporal operators to limit the lower and upper bound of accumulated time. We omit the subscripts if there is no time restriction, *i.e.*, $t_l = 0, t_h = \infty$. Other logical and temporal operators, including $\vee$, $\Rightarrow$, $\mathbf{W}$, $\overset{\mathbf{U}}{\Rightarrow}$ and $\overset{\mathbf{W}}{\Rightarrow}$, can be defined from these in a manner analogous to the definitions of their continuous counterparts.

3.4.2 Probability for Metastable Behaviors

Our extended LTL also includes some qualitative concepts from probability. The need for a probabilistic formulation arises from circuits such as the arbiter where metastable behaviors can occur. In particular, any arbiter described by a continuous model must have input conditions that result in an unbounded delay between asserting a request by a client and the assertion of the corresponding grant by the arbiter. For a well-designed arbiter, the probability of a request being ungranted should go to zero in the limit that time goes to infinity. As we will describe in Chapter 5, similar issues occur for the Rambus oscillator, and we expect that metastability will be an issue for verifying any circuit that has multiple distinct modes of operation. The key idea is to find a set $B \subset \mathbb{R}^d$ where d is the dimensionality of the ODE model for the circuit and its environment, such that B is a surface of dimension less than or equal to $d-1$. If we can show that at some time, t, all trajectories that fail to satisfy the specification must be in B, then we argue that such failures only occur with zero probability. However, our ODE models are deterministic; so, the rest of this section describes two approaches for introducing probability into the framework.

We recognize that the ODE models that we use in COHO are an approximation of the actual physical circuit. For example, ODE models are deterministic and neglect the noise that is present in real circuits due to thermal noise, crosstalk and other disturbances. In principle, one could use stochastic differential equations to account for this noise, but to do so would make a challenging verification problem even harder.

One way to handle this problem would be to view the ODE model of a circuit and its environment as an abstraction of a stochastic ODE model. For example, the ODE $\dot{x} = f(x)$ could be concretized to the Langevin equation $\dot{x} = f(x) + \eta(t)$, where $\eta(t)$ represents the contribution of random processes (a.k.a. "noise") to the

dynamics at time t. Often, in the treatment of stochastic ODEs, $\eta(t)$ is assumed to be Gaussian, but this raises a problem in our verification context: for any bound M and any time interval ΔT, the integral of $\eta(t)$ over the interval ΔT has some non-zero probability of being greater than M. Thus, any signal could be perturbed by any amount. This means that noise, with some small but non-zero probability, could cause any digital circuit to fail, and no properties can be verified. From a practical perspective, we'll note that noise-margin analysis is an area where extensive research has been done (e.g. [132, 141, 223]), and regard the details of noise-margin analysis as beyond the scope of the current dissertation. Instead, we will assume that the noise margins are sufficient so that the probability of failure of a logic gate is so small that it can be ignored. In particular, we will assume that $\eta(t)$ is bounded. We also assume that all nodes of the circuit are perturbed by noise, and that these perturbations are independent for different nodes and different times. We will say that such a noise model is "reasonable". With this approach, we will say that a specification, S, is satisfied "almost-surely" (or "with probability one") if S is satisfied almost-surely for any stochastic ODE with a "reasonable" η. In this case, the bounded noise, η, of the stochastic model provides the underlying randomness that allows us to consider the probability of various events.

An alternative approach is to stay with an ODE model for the circuit and its environment but to assume that there exists a probability distribution of the initial states for trajectories. We require the ODE model to be C^2, in which case solutions of the ODE are unique. Thus, the initial state of the circuit and its environment determine the entire trajectory. We will not specify the details of the probability distribution of the initial states, but we will again require it to be "reasonable." If the model for the circuit and its environment has d variables, the state space for the model is $\mathbb{R}^d$. If B is a measurable set, we write $\|B\|$ to denote the Lesbesgue measure of B (intuitively, $\|B\|$ is the volume occupied by B), and we write $\mu(B)$ to denote the probability measure of B. We say that μ is "reasonable" if $\mu(B)$ is zero for any set B for which $\|B\|$ is zero. With this approach, we will say that S is satisfied "almost-surely" (or "with probability one") if it is satisfied for all trajectories except for those starting from points in a set Z with $\|Z\| = 0$. In practice, this amounts to showing that at any time t, all "bad" trajectories must lie on a manifold whose dimensionality is less than that of the full-system. Thus, nearly-all small

perturbations to such a trajectory would move it off of the manifold so as to obtain a behavior that satisfies the specification.

We conjecture that these two mechanisms of introducing probability into our logic are equivalent. In particular, the "cone" arguments that we use for the arbiter and the oscillator circuits appear to be robust to the introduction of a noise component, η, into the model. However, we do not have a proof of this equivalence. Thus, we will use the second approach to define "almost-surely" for our version of LTL. If our conjecture is true, then the "almost-surely" results extend to the more physical notion of randomness of the first approach as well.

We define an *almost surely* version of the LTL "always" operator as shown below:

$$\phi \models \Box_Z S \quad \equiv \quad (\phi \models (\Box S) \vee ((\phi \in Z) \wedge (\|Z\| = 0)), \tag{3.11}$$

where the term $((\phi \in Z) \wedge (\|Z\| = 0))$ is treated as an atomic proposition in the formula. In other words, ϕ satisfies $\Box_Z S$ iff S holds everywhere along ϕ or if ϕ is in a negligible set, Z. This means that the probability of S holding everywhere along ϕ is equal to 1. Note that there is one Z for all trajectories, and only a negligible subset of the trajectories are in Z. Furthermore, we only have to show the existence of a negligible Z that contains all of the failure trajectories. It is not necessary to construct the set explicitly. In fact, in many cases finding the exact set may be very difficult or even uncomputable, but showing its existence can be straightforward.

3.4.3 Brockett's Annuli

To specify analog signals, we make extensive use of Brockett's annulus construction [37] as shown in Figure 3.17. When a variable is in region 1, its value is constrained but its derivative may be either positive or negative. Thus, region 1 of the annulus specifies a logically low signal: it may vary in a specified interval around the nominal value for low signals. When the variable leaves region 1, it must be increasing; therefore, it enters region 2. Because the derivative of the variable is positive in region 2, it makes a monotonic transition leading to region 3. Regions 3 and 4 are analogous to regions 1 and 2 corresponding to logically high and mono-

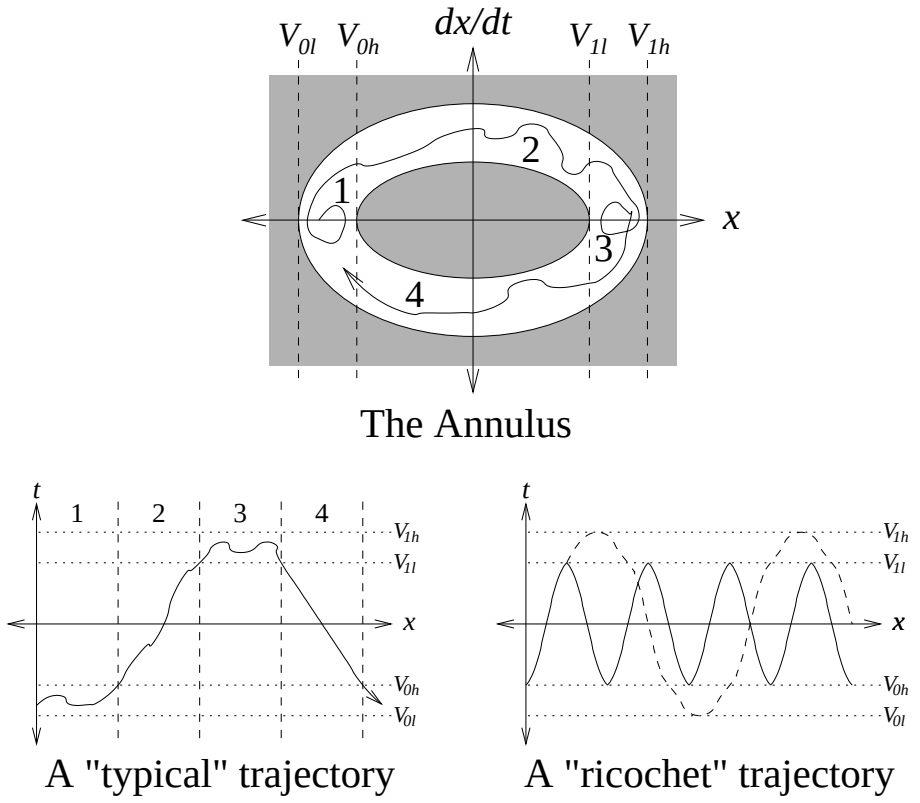


Figure 3.17: A Brockett's Annulus

tonically falling signals respectively. This provides a topological basis for discrete behaviors – the hole in the middle of the annulus forces rising and falling transitions to be unambiguous – regions 2 and 4 of the annulus admit signals with the same levels, but are distinguished by the value of the signal's time derivative. This construction forbids a signal from making a partial transition to some value in region 2 or 4 and then returning to where it came from without making a complete transition.

Furthermore, the horizontal radii of the annulus define the maximum and minimum high and low levels of the signal (i.e. V_{0l}, V_{0h}, V_{1l}, and V_{1h} in Figure 3.17). The maximum and minimum rise time for the signal correspond to trajectories along the upper-inner and upper-outer boundaries of the annulus respectively. Likewise, the

lower-inner and lower-outer boundaries of the annulus specify the maximum and minimum fall times. For simplicity, we often specify a Brockett's annulus using two ellipses as shown below:

$$\begin{array}{rcl} \frac{(v-v0_i)^2}{a_i^2}+\frac{\dot{v}^2}{b_i^2} & = & 1 \\ \frac{(v-v0_o)^2}{a_o^2}+\frac{\dot{v}^2}{b_o^2} & = & 1 \\ v0_i=\frac{V_{0h}+V_{1l}}{2} \qquad a_i=\frac{V_{1l}-V_{0h}}{2} & & \\ v0_o=\frac{V_{0l}+V_{1h}}{2} \qquad a_o=\frac{V_{1h}+V_{0l}}{2}, & & \end{array} \tag{3.12}$$

where b_i and b_o are the minor radii of inner and outer rings. Trajectories corresponding to the inner and outer boundaries of the annulus are sine waves. However, it is not the case that these waves give upper and lower bounds of the signal period. First, a signal may remain in regions 1 or 3 for an arbitrarily long time. Furthermore, the signal is not required to spend any time in regions 1 or 3. The minimum period signal corresponds to a "ricochet" trajectory as depicted by the solid curve in the right most plot of Figure 3.17. The period of such a signal is much less than that of the sine wave corresponding to the outer boundary of the annulus (the dashed curve). To exclude "ricochet" trajectories, we add constraints specifying the minimum low time t_l and minimum high time t_h, *i.e.*, the minimum duration of sojourns in region 1 and 3. Therefore, we specify a Brockett's annulus by $\mathscr{B}_{(V_{0l},V_{0h},V_{1l},V_{1h},b_i,b_o,t_l,t_h)}$ (or $\mathscr{B}_{(V,b,t)}$ for short). To express that a signal x satisfies the annulus $\mathscr{B}_{(V,b,t)}$, we include propositions of the form $\mathscr{B}_{(V,b,t)}(x)$ in the set of atomic propositions for our continuous LTL from Section 3.4.1.

This Brockett's annulus construction allows a large class of signals to be described in a simple and natural manner. Given any trajectory, $x(t)$ that is contained in the interior of the annulus, any trajectory $x'(t)$ that is obtained from a small, differentiable perturbation of $x(t)$ is also in the annulus. This is in contrast with traditional circuit simulators that test a circuit for specific stimuli such as piecewise linear or sinusoidal waveforms. Thus, a Brockett's annulus can be given that contains all trajectories that will occur during actual operation, something that traditional simulation cannot achieve. Of course, such an annulus also includes trajectories that will never occur during actual operation. Thus, this abstraction is sound in that false positives are excluded, but false negatives could, in principle, occur.

In our experience, the Brockett's annulus abstraction has not been a cause for false negatives.

A Brockett's annulus provides the mapping from continuous trajectories to discrete traces. We write $B_i(x)$ to indicate that variable x is in region i of the annulus, and $B_{i,j}(x)$ to indicate that it is in region i or region j. If a trajectory is in region B_1 for variable x, then its discrete abstraction is unambiguously low (i.e. false); likewise if it is in region B_3, then it is clearly high. If the trajectory is in region B_2 (*resp.* B_4), then it *may* be treated as high (*resp.* low), but it is not required to do so until the signal enters region B_3 (*resp.* B_1). We say that a signal makes a rising transition when it enters region 2 of its Brockett's annulus and a falling transition when it enters region 4. Because Brockett's annuli impose minimum rise and fall times for signals, the number of rising and falling transitions is countable. Furthermore, this mapping connects the discrete specification and continuous specification in Section 3.4.1.

A Brockett's annulus also provides a method to specify input signals and calculate the time derivative to construct a mathematical model of a circuit. For regions 1 and 3, the derivative is bounded by the outer ring. For regions 2 and 4, extra information of the current discrete region is required to distinguish positive or negative derivatives as they share the same level signals. Therefore, the ordinary differential inclusion (ODI) model of the input signals is

$$\frac{d}{dt}V_i = f(V_i, \mathscr{B}_{(V,b,t)}, s), \tag{3.13}$$

where s is the index of the current discrete region. Equation 3.2 and Equation 3.13 give a formal model for a circuit and its environment.

3.5 Specification Examples

We applied our LTL and Brockett's annulus based method to formally specify properties of circuits in Section 3.2. In the following, we first describe how to define a discrete specification for an arbiter, and how to translate it to a continuous specification. Then we present continuous specifications for other circuits.

Initially:

$\forall i \in \{1,2\}.\ \neg \mathsf{r}_i \wedge \neg \mathsf{g}_i$

Assume (environment controls r_1 and r_2):

$\forall i \in \{1,2\}. \quad \Box(\mathsf{r}_i \overset{\mathbf{W}}{\Rightarrow} \mathsf{g}_i) \wedge \Box(\neg \mathsf{r}_i \overset{\mathbf{W}}{\Rightarrow} \neg \mathsf{g}_i) \wedge \Box(\mathsf{g}_i \overset{\mathbf{U}}{\Rightarrow} \neg \mathsf{r}_i)$

Guarantee (arbiter controls g_1 and g_2):

Handshake:

$\forall i \in \{1,2\}.\ \Box(\neg \mathsf{g}_i \overset{\mathbf{W}}{\Rightarrow} \mathsf{r}_i) \wedge \Box(\mathsf{g}_i \overset{\mathbf{W}}{\Rightarrow} \neg \mathsf{r}_i)$

Mutual Exclusion:

$\Box \neg (\mathsf{g}_1 \wedge \mathsf{g}_2)$

Liveness:

$\forall i \in \{1,2\}.\ (\Box(\mathsf{r}_i \overset{\mathbf{U}}{\Rightarrow} \mathsf{g}_i)) \wedge (\Box(\neg \mathsf{r}_i \overset{\mathbf{U}}{\Rightarrow} \neg \mathsf{g}_i))$

Figure 3.18: Discrete Specification for an Arbiter

3.5.1 Arbiters

Figure 3.18 gives an LTL specification for a discrete arbiter using an assume-guarantee approach [114] for separating the assumptions made about the clients, from the requirements for the arbiter. The "assume" clause describes what the environment can do: it can only modify r_1 and r_2, and it must do so in a way that satisfies the formulas in the assume clause. Conversely, the "guarantee" clause describes what the arbiter must do: it can only modify g_1 and g_2, and it must do so in a way that satisfies the formulas in the guarantee clause.

In English, the specification says that if the clients observe the four-phase handshake protocol, then the arbiter will observe the protocol and ensure that grants are mutually exclusive. The "initially" section states that initially, neither client is making a request and the arbiter is issuing no grants. The "assume" section describes the expected behavior of the clients. For example, $\Box(r_i \overset{\mathbf{W}}{\Rightarrow} g_i)$ states that once a client makes a request, it will continue to assert the request until the arbiter asserts the corresponding grant. In other words, a client may not withdraw a pending request. The specification uses the weak-until, **W** because the environment is not responsible for issuing a grant. Likewise, the clause $\Box(\neg r_i \overset{\mathbf{W}}{\Rightarrow} \neg g_i)$ states that when a client has withdrawn a request, it must wait until after the arbiter has lowered the

corresponding grant signal before the client can make another request. This specification does not require clients to make requests. This is deliberate: the arbiter should function correctly even if one or both of its clients never make a request. On the other hand, the clause $\Box(g_i \overset{\mathbf{U}}{\Rightarrow} \neg r_i)$ states that once a request is granted, the request must be eventually withdrawn. Without this requirement, we would not be able to require that all requests are eventually granted, as one client could hold the grant forever.

The "guarantee" section describes the required behavior of the arbiter. The clause $\Box(\neg g_i \overset{\mathbf{W}}{\Rightarrow} r_i)$ states that the arbiter may not issue a grant until after the corresponding request has been made. Likewise, $\Box(g_i \overset{\mathbf{W}}{\Rightarrow} \neg r_i)$ states that the arbiter must continue to issue a grant until the environment withdraws the request. As in the "assume" section, we use the weak until because the arbiter is not responsible for ensuring that the clients eventually lower their requests. The mutual exclusion clause states that both grants cannot be issued at the same time. The liveness properties state that all requests must eventually be granted, and a grant must be lowered following the lowering of the request. Due to metastability, no physical arbiter can guarantee both safety and liveness for contested requests [154, 157]. We address these issues in our specification of the continuous arbiter below.

To specify a continuous arbiter, we use a Brockett's annulus to identify regions of the state space, $\mathscr{V} \times d\mathscr{V}$ that correspond to true or false values of the atomic propositions (such as r_1) from the discrete specification, and we modify the liveness conditions to use an almost-surely formulation for situations with contested requests. Of course, uncontested requests and releases of requests should receive responses for all trajectories and not just a subset with probability 1.

To show that contested requests are granted (almost-surely), we follow the approach of [160]. Their approach introduced a concept called "α – insensitivity." It excludes clients which are feedback controllers and thereby trap the arbiter in its metastable region by exquisite design or unimaginable coincidence. This constraint on the clients is expressed by requiring that the clients be relatively insensitive to variations of the two grant signals when waiting for a grant. In real circuits, "accidentally" designing clients that act as feedback controllers would be extremely far-fetched. We are much better off worrying about the approximations used in HSPICE models and other more probable causes of failure. Thus, in our continu-

Initially:

$\forall i \in \{1,2\}.\ B_1(r_i) \wedge B_1(g_i)$

Assume (environment controls r_1 and r_2):

$$\forall i \in \{1,2\}. \quad \Box(B_3(r_i) \overset{\mathbf{W}}{\Rightarrow} B_{2,3}(g_i)) \ \wedge\ \Box(B_1(r_i) \overset{\mathbf{W}}{\Rightarrow} B_{4,1}(g_i)) \wedge \Box(B_3(g_i) \overset{\mathbf{U}}{\Rightarrow} B_{4,1}(r_i))$$

Guarantee (arbiter controls g_1 and g_2):

Handshake:

$$\forall i \in \{1,2\}.\ \Box(B_1(g_i) \overset{\mathbf{W}}{\Rightarrow} B_{2,3}(r_i)) \ \wedge\ \Box(B_3(g_i) \overset{\mathbf{W}}{\Rightarrow} B_{4,1}(r_i))$$

Mutual Exclusion:

$$\Box\neg(B_{2,3}(g_1) \ \wedge\ B_{2,3}(g_2))$$

Liveness:

$$\forall i \in \{1,2\}. \quad (\mathit{Parameters} : t_r, t_f \in \mathbb{R}^+)$$
$$\alpha\text{-ins} \ \Rightarrow\ (\Box_Z(B_3(r_i) \overset{\mathbf{U}}{\Rightarrow} B_{2,3}(g_i)))$$
$$\wedge \quad (B_3(r_i) \overset{\mathbf{U}_{[\mathbf{0},\mathbf{t_r}]}}{\Rightarrow} (B_{2,3}(g_i) \vee B_3(r_{\sim i}))) \wedge (\Box(B_1(r_i) \overset{\mathbf{U}_{[\mathbf{0},\mathbf{t_f}]}}{\Rightarrow} B_4(g_i)))$$

Figure 3.19: Continuous Specification for an Arbiter

ous specification, we assume that α-insensitivity holds and write α-ins to denote this α-insensitivity assumption. On the other hand, a faulty design could produce an arbiter with a "dead-zone" where the circuit could hang and never resolve contested grants. Thus, it is important to include the almost-surely liveness condition to ensure the correctness of a proposed design.

Figure 3.19 shows our specification for the behavior of an arbiter with a continuous model. Here, we wrote $\sim i$ to denote $3-i$, and thus $r_{\sim i}$ denotes the "other" request. For the most part, this is a direct translation of the discrete specification from Figure 3.18 to a continuous one using the Brockett's annulus construction to provide the required atomic propositions for the continuous version. The only other change was that we rewrote the first clause of the liveness condition from the discrete specification with two clauses. The first liveness clause for the continuous specification,

$$\alpha\text{-ins} \ \Rightarrow\ (\Box_Z(B_3(r_i) \overset{\mathbf{U}}{\Rightarrow} B_{2,3}(g_i))),$$

says that if the clients satisfy the α-insensitivity requirement described above, then all requests are eventually granted except for those in a set of trajectories, Z, where

Initially:

$B_1(\phi) \wedge B_3(z)$

Assume (environment controls ϕ):

$\Box \mathscr{B}_{(V,b,t)}(\phi)$

Guarantee (toggle controls z):

Switch:

$\forall i \in \{1,3\}$

$$\begin{aligned}
\Box \quad & (B_1(\phi) \wedge B_i(z) \wedge (z' = i) \overset{\mathbf{W}}{\Rightarrow} B_2(\phi) \wedge B_i(z) \wedge (z' = i)) && \wedge \\
\Box \quad & (B_2(\phi) \wedge B_i(z) \wedge (z' = i) \overset{\mathbf{U}}{\Rightarrow} B_{2,3}(\phi) \wedge B_{i,i+1,4-i}(z) \wedge (z' = i)) && \wedge \\
\Box \quad & (B_{2,3}(\phi) \wedge B_{i,i+1,4-i}(z) \wedge (z' = i) \overset{\mathbf{U}}{\Rightarrow} B_3(\phi) \wedge B_{4-i}(z) \wedge (z' = i)) && \wedge \\
\Box \quad & (B_3(\phi) \wedge B_{4-i}(z) \wedge (z' = i) \overset{\mathbf{W}}{\Rightarrow} B_4(\phi) \wedge B_{4-i}(z) \wedge (z' = i)) && \wedge \\
\Box \quad & (B_4(\phi) \wedge B_{4-i}(z) \wedge (z' = i) \overset{\mathbf{U}}{\Rightarrow} B_1(\phi) \wedge B_{4-i}(z) \wedge (z' = i)) && \wedge \\
\Box \quad & (B_1(\phi) \wedge B_{4-i}(z) \wedge (z' = i) \overset{\mathbf{W}}{\Rightarrow} B_1(\phi) \wedge B_{4-i}(z) \wedge (z' = 4-i))
\end{aligned}$$

Brockett's Annulus:

$\Box \mathscr{B}_{(V,b,t)}(z)$

Figure 3.20: Specification for a Toggle Circuit

Z has zero probability. To verify this condition, we do not have to explicitly construct the set Z, we simply have to prove that such a set exists. The second clause of the liveness section states that uncontested requests are eventually granted, and that grants are always withdrawn after the corresponding request is withdrawn. These correspond directly to the discrete specification, and say that the continuous arbiter should respond in a bounded time (t_r, t_f for grant and withdrawn respectively) in situations where metastability is avoidable.

3.5.2 The Yuan-Svensson Toggle

Figure 3.20 presents our specification for a toggle such as the one that was shown in Figure 3.3. The "initially" section describes the initial state of the toggle: ϕ is low and z is high. The "assume" section says that the input clock ϕ must satisfy a given Brockett's annulus $\mathscr{B}_{(V,b,t)}$. The "guarantee" section describes the state transition diagram as shown in Figure 3.4. This property ensures the period of the output signal z is twice that of the clock signal ϕ. It says that the output z makes

one low-to-high or high-to-low transition for each period of the clock input. For example, if z is low when ϕ is low, z must remain stable until ϕ enters region 2 of a Brockett's annulus; then z can transit to high value via regions $1, 2$ and 3, and the transition must be completed before ϕ entering region 3; the value of z holds during the falling transition of ϕ. The auxiliary variable z' records the value of z of the previous period. The auxiliary variable is implemented by neither the environment nor the toggle. It is added by the specification and is implicitly existentially quantified: if there exists a function, $z' : \mathbb{R}^+ \to$ bool such that the LTL formulas of the specification are satisfied, then the specification is satisfied. Because a signal specified by a Brockett's annulus can stay as low or high for an arbitrary long time, we apply $\overset{\mathbf{W}}{\Rightarrow}$ rather than $\overset{\mathbf{U}}{\Rightarrow}$ in the first, fourth, and sixth clauses. The specification also requires that the output signal z satisfies the same Brockett's annulus $\mathscr{B}_{(V,b,t)}$.

3.5.3 Flip-Flops

Figure 3.21 shows a specification for a flip-flop circuit such as the one that was shown in Figure 3.6. It says that if the data input d meets its set-up and hold criteria, then the output q updates its value after the clock-to-q delay. The "parameters" section presents three parameters used in the specification: t_{setup}, t_{hold} and t_{clk2q} for set-up time, hold time and clock-to-q delay time respectively. Note that the set-up and hold time could be negative in physical flip-flops, we do not require t_{setup} and t_{hold} to be positive parameters.

The "assume" section presents input specifications for the clock ϕ and data input d. The "Brockett's annuli" clause requires that both ϕ and d satisfy given Brockett's annuli. The "set-up & hold criteria" clause states that the value of d is held steady for at least t_{setup} time before the clock event and for at least t_{hold} time after the clock event. $B_2(\phi) \Rightarrow \Box_{[0,t_{hold}]} B_{1,3}(d)$ says that when ϕ is in B_2, d must remain in B_1 or B_3 for the next t_{hold} time units. This clause specifies the hold requirement when t_{hold} is positive. $B_{2,4}(d) \Rightarrow \Box_{[0,t_{setup}]} B_1(\phi)$ says that that if d is not stable, ϕ may not enter B_2 for the next t_{setup} time units. This clause specifies the set-up requirement for $t_{setup} \geq 0$. The set-up and hold requirements for $t_{setup} < 0$ or/and $t_{hold} < 0$ can be specified similarly as shown in the "set-up & hold criteria"

Parameters :

$t_{setup} \in \mathbb{R},\ t_{hold} \in \mathbb{R},\ t_{clk2q} \in \mathbb{R}^+$.

Assume (environment controls ϕ and d):

Brockett's Anulli:

$\Box \mathscr{B}_{(V1,b1,t1)}(\phi) \wedge \Box \mathscr{B}_{(V2,b2,t2)}(d)$

Set-up & Hold Criteria:

$\Box((t_{setup} \geq 0) \wedge B_{2,4}(d) \Rightarrow \Box_{[0,t_{setup}]} B_1(\phi)) \wedge$
$\Box((t_{hold} \geq 0) \wedge B_2(\phi) \Rightarrow \Box_{[0,t_{hold}]} B_{1,3}(d)) \wedge$
$\Box((t_{setup} \leq 0) \wedge B_2(\phi) \Rightarrow \Box_{[0,-t_{setup}]} B_{1,3}(d)) \wedge$
$\Box((t_{hold} \leq 0) \wedge B_{2,4}(d) \Rightarrow \Box_{[0,-t_{hold}]} B_3(\phi))$

Guarantee (flip-flop controls q):

Stable Output:

$\forall i \in \{1,3\}$
$\Box(B_1(\phi) \wedge B_i(d) \overset{\mathbf{W}}{\Rightarrow} (B_1(\phi) \wedge B_i(d) \wedge (d' = i)) \vee (B_1(\phi) \wedge \neg B_i(d))) \wedge$
$\Box(d' = i) \overset{\mathbf{W}}{\Rightarrow} B_1(\phi) \wedge$
$\Box(B_3(\phi) \ \Rightarrow \ \Diamond_{[0,t_{clk2q}]} B_{d'}(q)) \wedge$
$\Box(B_3(\phi) \wedge B_{d'}(q) \overset{\mathbf{W}}{\Rightarrow} B_4(\phi) \wedge B_{d'}(q)) \wedge$
$\Box(B_{4,1}(\phi) \ \Rightarrow \ B_{1,3}(q))$

Figure 3.21: Specification for a Flip-Flop

section.

The "guarantee" section states that if ϕ has not had a rising edge in the past t_{clk2q} time, then q has the same value that d had the last time ϕ entered B_2. To record the value of d in the past, we use an auxiliary variable d' like we did for the toggle specification. The first clause of the "guarantee" sections says that when ϕ is in B_1 and d in B_1 or B_3, either d' will eventually record the value of d or d transits to other regions. Combined with the set-up requirements, d' records the value of d when ϕ entered B_2 the last time in the past. $(d' = i) \overset{\mathbf{W}}{\Rightarrow} B_1(\phi)$ says that the value of d' is held when ϕ is not in B_1. The remainder of this section states that q must has the same value with d' after the clock-to-q delay. $B_3(\phi) \Rightarrow \Diamond_{[0,t_{clk2q}]} B_{d'}(q)$ says that when ϕ is in B_3, q must update its value to d' in the next t_{clk2q} time units. The last two clauses say that the value of q is held steady before ϕ enters B_2 in the future.

Parameters:

$\varepsilon \in \mathbb{R}^+,\ V_0 \in \mathbb{R}.$

Definition:

$\forall i \in \{1,\cdots,n\},\ s_i \equiv x_i + x_{n+i};\ d_i \equiv (-1)^i (x_i - x_{n+i}).$

$$i \ominus 1 \equiv \begin{cases} i-1 & \text{if } i > 1, \\ 4 & \textit{otherwise}. \end{cases}$$

$lead(x,y) \equiv (B_i(x) \wedge B_{i\ominus 1,i}(y)),\ \forall i \in \{1,2,3,4\}.$

Guarantee:

Common Mode:

$\forall i \in \{1,\cdots,n\},\ |s_i - V_0| \leq \varepsilon$

Differential:

$\exists V,b,t,\ \forall i \in \{1,\cdots,n\},\ \Box_Z \Diamond (\mathscr{B}_{(V,b,t)}(d_i) \wedge (B_{1,3}(d_i) \overset{\mathbf{U}}{\Rightarrow} B_{2,4}(d_i)))$

$(\forall i \in \{1,\cdots,n-1\}, \Box\, lead(d_i, d_{i+1})) \ \wedge\ \Box\, lead(d_n, -d_1)$

Non-harmonic:

$\exists i,j \in \{1,\cdots,n\}, i \leq j$

$\Box\ (\forall k \in [1,i], B_1(d_k) \wedge \forall k \in (i,j), B_4(d_k) \wedge \forall k \in [j,n], B_3(d_k))\ \vee$

$\Box\ (\forall k \in [1,i], B_3(d_k) \wedge \forall k \in (i,j), B_2(d_k) \wedge \forall k \in [j,n], B_1(d_k))\ \vee$

$\Box\ (\forall k \in [1,i], B_4(d_k) \wedge \forall k \in (i,j), B_3(d_k) \wedge \forall k \in [j,n], B_2(d_k))\ \vee$

$\Box\ (\forall k \in [1,i], B_2(d_k) \wedge \forall k \in (i,j), B_1(d_k) \wedge \forall k \in [j,n], B_4(d_k))$

Figure 3.22: Specification for a Rambus Ring Oscillator

3.5.4 The Rambus Ring Oscillator

Figure 3.22 shows the specification for the Rambus ring oscillator from Figure 3.10. It says that any signal of the circuit oscillates as expected with all initial conditions. For each differential pair of nodes x_i and x_{n+i}, we define $s_i = x_i + x_{n+i}$ as its common mode component and $d_i = (-1)^i (x_i - x_{n+i})$ as its differential component. The common mode component is quite stable, *i.e.,* close to a constant value V_{dd} as shown in the "common mode" section. The differential component d_i "almost-surely" oscillates under all initial conditions as shown in the "differential" section. We specify the oscillation behavior by the Brockett's annulus construction, *i.e.,* all differential signals transit from region 1 to 4 in sequence. Note that there is no upper bound for the dwell time in regions 1 and 3 in the Brockett's annulus construction, so we use

$$B_{1,3}(d_i) \overset{\mathbf{U}}{\Rightarrow} B_{2,4}(d_i)$$

to force the trajectory to leave stable regions within a bounded time. However, differential signals may not oscillate if the initial state is an equilibrium point, that is, currents of all transistors are zeros. Under this circumstance, the circuit is not stable and diverges to oscillation mode with small disturbances. We apply the similar "almost-surely" version always operator to denote that all equilibrium points are in a negligible set. Furthermore, we specify the relationship between adjacent variables d_i and d_{i+1} by the "lead" function. We say the value of d_i "leads" the value of d_{i+1} because of the forward inverters of the stage[3]. When d_i is low, it makes d_{i+1} low or start to fall; and d_{i+1} cannot rise before d_i starts to rise. We use the $B_1(d_i) \wedge B_{1,4}(d_{i+1})$ clause to exclude the case that d_{i+1} is in $B_{2,3}$ when d_i is in B_1. Similarly, we define the valid value of d_{i+1} when d_i is in other regions, as shown in the "definition" section. Note that we use $-d_1$ in the last "lead" function because the signal is swapped at the end of the ring.

When n is large, harmonic behaviors can appear if the circuit is not well designed. Under this mode, the circuit may oscillate with a faster frequency than expected. For example, the circuit may oscillate with frequency $3f$ where f is the designed frequency. We specify that there is only one transition phase in the ring for the non-harmonic property. The first clause of the "non-harmonic" specification says that there is only one high-to-low or low-to-high transition in the middle of the ring . The first phase $B_1 d_k$ states that all signals in the beginning of the ring are low; the second phase $B_4 d_k$ states that all signals in the middle of the ring are falling from high to low; and the third phase states that all signals in the end of the ring are high. The second clause describes a low-to-high transition in the middle of the ring, similar to the first clause. Similarly, the last two clauses are for when transitions occur in the beginning or in the end of the ring.

3.6 Implementation

The previous two sections described how circuits can be modeled using non-linear ODEs and an extension of LTL with continuous time and values for specifying circuit properties. This section describes how we convert the nonlinear ODEs to linear

[3]When $n = 2$, d_1 and d_2 are symmetric and it is improper to say that d_1 leads d_2. Therefore, the "lead" clause does not apply to two-stage Rambus ring oscillators.

differential inclusions, and how we incorporate input signals that are described by Brockett's annuli into our reachability computations. These transformations make the modeling and specification methods described earlier in this chapter practical for the reachability methods that will be described in chapter 4.

3.6.1 Linearization Methods

The nonlinear model from Equation 3.13 and Equation 3.2 usually cannot be solved efficiently by reachability analysis tools. Therefore, we compute a linear differential inclusion (LDI) to over-approximate the ODE model for efficient computation as:

$$\frac{d}{dt}V_i \in A_iV_i + b_i \pm u_i \tag{3.14}$$

$$\frac{d}{dt}V_o \in A_oV + b_o \pm u_o, \tag{3.15}$$

where V_i refers to input voltages, V_o refers to voltages of other ODE nodes, A_i is the linear coefficient for input signals, b_i is the constant term, u_i is the error term, and A_o, b_o, u_o are coefficients for ODE nodes. Because input signals are specified by Brockett's annuli, the time derivative $\frac{d}{dt}V_i$ only depends on its own voltage value V_i. The time derivative of ODE nodes $\frac{d}{dt}V_o$ depends on voltages of all nodes.

We have developed two algorithms for computing Equation 3.14 for input signals specified by Brockett's annuli. The first algorithm finds a linear approximation with minimized L_2 norm error term based on the least squares method. Given an input signal v specified by $\mathscr{B}_{(V,b,t)}$ (defined in Equation 3.12), and a range of value $[v_l, v_h]$, the algorithm constructs a linear approximation $\dot{v}_a$ as

$$\begin{aligned} \min E &= \min \textstyle\int_{v_l}^{v_h} (\dot{v}_a - \dot{v}_m)^2 dv \\ \dot{v}_a &= A_v V + b_v \\ \dot{v}_m &= \frac{\frac{b_i}{a_i}\sqrt{a_i^2-(v-vo_i)^2}+\frac{b_o}{a_o}\sqrt{a_o^2-(v-vo_o)^2}}{2}. \end{aligned} \tag{3.16}$$

This optimization problem can be solved by the least squares method as

$$\begin{bmatrix} A_v \\ b_v \end{bmatrix} = \begin{bmatrix} \int_{v_l}^{v_h} v^2\, dv & \int_{v_l}^{v_h} v\, dv \\ \int_{v_l}^{v_h} v\, dv & \int_{v_l}^{v_h} dv \end{bmatrix}^{-1} \begin{bmatrix} \int_{v_l}^{v_h} \dot{v}_m v\, dv \\ \int_{v_l}^{v_h} \dot{v}_m\, dv \end{bmatrix}. \tag{3.17}$$

With this linear approximation, the error bound can be found by calculating error terms of points v_l, v_h and $v_e = v0 - \frac{a^2 A_v}{\sqrt{a^2 A_v^2 + b^2}}$. However, the least squares method minimizes the L_2 norm rather than the L_1 norm. Therefore, the constant term b_v is adjusted at the end to balance the lower and upper error bounds.

On the other hand, the second algorithm minimizes the L_1 norm based on linear programming. The algorithm first conservatively approximates the annulus by two polygons p_i and p_o. Then it finds the best linear approximation by solving a linear program:

$$\begin{array}{lll} \min u_v & s.t. & \\ \dot{v} & \in & A_v v + b_v \pm u_v \\ v & \in & [v_l, v_h, p_{ix}, p_{ox}], \end{array} \tag{3.18}$$

where $p_{ix}(p_{ox})$ denotes signal values of vertices of $p_i(p_o)$.

Linear differential inclusions as shown in Equation 3.15 for ODE nodes can be computed by either the least squares based method or LP based method similarly. From Equation 3.2, we can see that it suffices to linearize the current function of transistors to create a linear model. The linear approximation can be obtained by solving a linear program similar to Equation 3.18 which minimizes the error term with conditions that the linear inclusion is valid for all grid points of our table-based models as described in Section 3.3.2. However, this LP usually contains a huge number of constraints, each of which corresponds to one grid point. Because the LP approach is impractical for most problems, only the least-squares method was implemented. Similar with Equation 3.16, the least-squares method finds the best linear approximation according to the L_2 norm of the error term and adjusts the constant term at the end. This method can work with both i_{ds} tables and polynomial tables.

Given an i_{ds} table as described in Section 3.3.2, our algorithm computes a linear inclusion for a region specified by a user-provided linear program. The algorithm first finds the bounding box of the region, collects all grid points in the box, and computes a linear fit using the least squares method. Given the linear fit, our algorithm then computes the error term by evaluating the error function over all grid points. The time to compute the linear coefficients is constant by applying pre-computed sums of tabulated data. Evaluating the error term dominates the

computation.

For polynomial tables, the current function is approximated by a quadratic polynomial as described in Section 3.3.2. The least squares method is implemented similar to the one for i_{ds} tables. However, we have two improvements. First, we use the linear program to trim hyper-cubes to reduce error and improve performance. Second, noting quadratic polynomials are Lipschitz functions and the Lipschitz constants can be computed efficiently, we over-estimate the error term using the Lipschitz constant and under-estimate the error by evaluating the error function on several sampled points. Therefore, the lower and upper bounds of the error in each hyper-cube can be efficiently computed. With the error bound, most hyper-cubes can be pruned without evaluating the exact error. This process can be refined to narrow down the error bound. Because the polynomial-interpolation tables are on a much coarser grid than the raw i_{ds} data, the algorithm for polynomial tables is more efficient than the one for i_{ids} tables.

3.6.2 Modeling Input Behaviors

In order to obtain conservative results, it is necessary to compute circuit states under all possible input transitions. As described in Section 3.4.3, input signals can be specified by Brockett's annuli. However, regions 2 and 4 of an annulus admit signals with the same range, *i.e.,* they are indistinguishable without information of time derivative of the signal. Therefore, reachability computations of regions 2 and 4 must be separated.

As mentioned in Section 3.4.3, a Brockett's annulus has four regions that correspond to logic low, rising, high and falling signals. Continuous input trajectories can be mapped to discrete sequences, *i.e.,* iteration of low, rising, high and falling stages. Therefore, transitions of one input signal can be modeled by four *states* denoted as B_1, B_2, B_3, B_4. For a circuit with only one input signals, all possible circuit behaviors can be obtained by computing reachable regions within these four states. Reachability analysis is performed in each state B_i with its initial regions. In each computation step, forward reachable regions are sliced by the hyperplane which is the boundary between the current state B_i and the next state B_{i+1} (*e.g.,* $v = 0.2$). These slices are accumulated, and the result is used as the initial region

of state B_{i+1}. Note that there are minimum dwell time requirements in low and high regions, trajectories cannot leave $B_1(B_3)$ before the minimum time t_l (t_h).

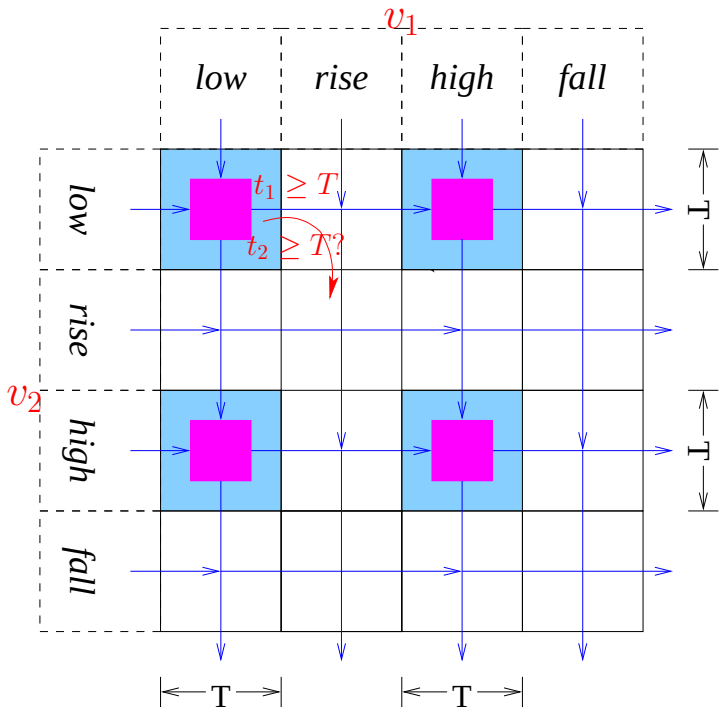


Figure 3.23: Input Transitions without the Dwell Time Requirement: Each signal has four regions: low, rise, high and fall, according to the Brockett annulus specification. There are 4^2 states for circuits with two input signals v_1, v_2. For each state $B_{<i,j>}$, there are two possible transitions to states $B_{<i+1,j>}$ and $B_{<i,j+1>}$. When a signal v_1 leaves a stable region (*i.e.*, low or high), the condition $t_1 \geq T$ must be satisfied according to the dwell time requirement. However, the condition $t_2 \geq T$ is necessary to check if signal v_2 can leave the stable region or not. This requires to record the time of each trajectory spent in a stable state which is generally impossible for reachability analysis.

The method can be generalized to transitions of two (or more) signals v_1, v_2. Assuming these two signals are independent, there are $4^2 = 16$ possible concurrent transitions, which are denoted as $B_{<i,j>}, i, j \in \{1,2,3,4\}$, where $i(j)$ is the region of signal $v_1(v_2)$. As shown in Figure 3.23, trajectores in a state $B_{i,j}$ can go to states

$B_{i+1,j}$, and $B_{i,j+1}$. Accordingly, reachable regions are sliced by two hyperplanes to compute initial regions for states $B_{i+1,j}$, and $B_{i,j+1}$: one hyperplane is the boundary between state $B_{i,j}$ and state $B_{i+1,j}$ (*e.g.*, $v_1 = 0.2$), and the other is for states $B_{i,j}$ and $B_{i,j+1}$ (*e.g.*, $v_2 = 1.6$). In case two signals are not independent, not all states are reachable. For example, states $B_{2,4}$ and $B_{4,4}$ are forbidden for a flip-flop circuit because of the timing constrains between the data input and the clock signal.

However, transitions of two inputs are more complicated when considering the minimum dwell time requirement. We assume that the minimum dwell time is T for regions 1 and 3 of two annuli, otherwise, we can set T as the minimum value of them. Due to the minimum dwell time requirement, any trajectory in state $B_{<1,1>}$ must satisfy the constraint $t_1 \geq T$ before entering state $B_{<2,1>}$, where t_1 is the time of signal v_1 spent in the stable (low or high) region, *i.e.*, the time of the trajectory spent in all states $B_{<1,*>}$. Similarly, any trajectory cannot enter $B_{<1,2>}$ until the second signal v_2 has stayed in region 1 for at least T time. As illustrated in Figure 3.23, when trajectories enter state $B_{<2,1>}$, the condition $t_2 \geq T$ may be satisfied or not. For trajectories where the condition is satisfied, they can enter state $B_{<2,2>}$ immediately or at any time; for trajectories where the condition is not satisfied and will not be satisfied before entering state $B_{<3,1>}$, they cannot goto state $B_{<2,2>}$; for other trajectories, they could go to state $B_{<2,2>}$ after the time when the condition $t_2 \geq T$ becomes to be true. This introduces a challenge to reachability analysis. It is generally impossible to record the exact time of each trajectory during reachability computation[4]. Therefore, it requires infinite number of reachability computation to obtain the accurate result: *e.g.*, v_2 has stayed in the low region for a time $t_l \leq T$ when v_1 leaves the low region. It is similar for trajectories in states $B_{<1,3>}, B_{<3,1>}$ and $B_{<3,3>}$.

We employ a conservative approximation technique to solve this problem. It first measures the maximum rising time t_{max} from the inner bound of the annulus and then uses $T' = T - t_{max}$ as the dwell time requirement of the logic low region[5]. When trajectories leave $B_{<1,1>}$ and enter $B_{<2,1>}$, the new condition $t_1 \geq T'$ must be satisfied. At the same time, the condition $t_2 \geq T$ is also checked. If it is satisfied,

[4] A finite-bisimulation cannot be constructed even though it can be done for timed automata. This is because our linear ODIs models have much more complicated dynamics than timed automata.

[5] T is usually much larger than t_{max} for circuit signals.

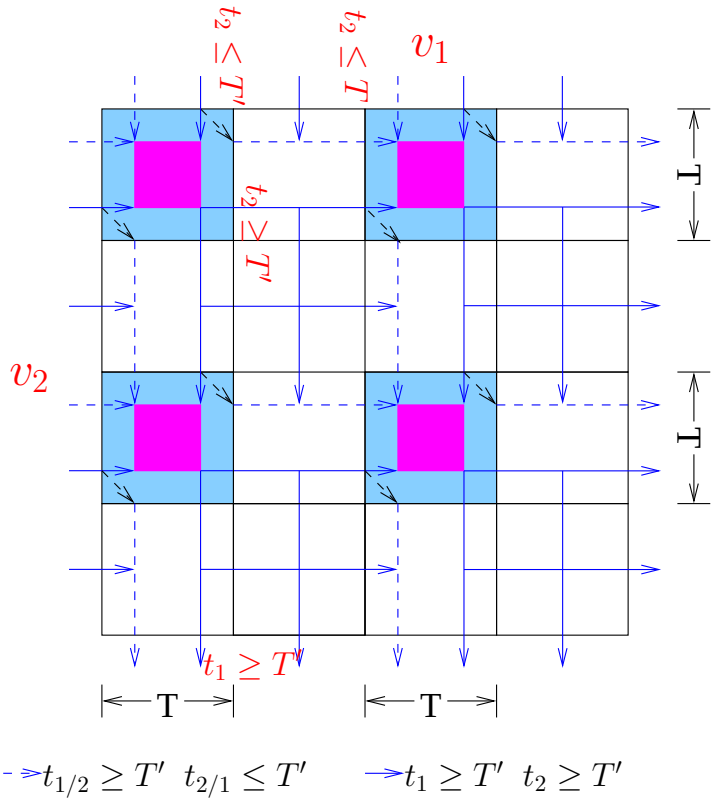


Figure 3.24: Input Transitions with the Dwell Time Requirement: We over-approximate valid trajectories by relaxing the time requirement from T to $T' = T - t_{max}$. All trajectories are grouped by the condition $t_1/2 \geq T'$: solid arrows denote trajectories in which both signals satisfy the time requirement and dashed arrows denote trajectories in which only one signal satisfies the time requirement. With this approximation technique, all possible trajectories are computed within a finite number of reachability computations.

trajectories are allowed to enter $B_{<2,2>}$ at any time. This is an over-approximation because $t_2 \geq T$ may not be true. Otherwise, trajectories must stay in $B_{<2,1>}$ and go to $B_{<3,1>}$. This is because $t_2 \geq T$ can not be satisfied when trajectories are in state $B_{<2,1>}$. By applying this method, all reachable regions in state $B_{<2,1>}$ are over-approximated by two reachability computation. The first one is for the case when both t_1 and t_2 are greater than T', and the second one is for the case when only one of two signals is greater than T'.

The reachability computation phase is illustrated in Figure 3.24. The solid arrows denote transitions of trajectories in the first case, and dashed arrows denote transitions of trajectories in the second case. Whenever a signal transits from the low (high) region to the rise (fall) region, the corresponding time requirement must be satisfied, and trajectories are grouped into two sets by the time requirement of the other signal. For example, when trajectories leaves $B_{<1,1>}$ to $B_{<2,1>}$, the condition $t_1 \geq T'$ must be satisfied. And forward reachable regions are partitioned into two sets by the condition $t_2 \geq T'$. For the first set where the condition is satisfied, reachability computations are performed in states $B_{<2,1>}$ and $B_{<2,2>}$. Noting trajectories can enter $B_{<3,1>}$ and then go to $B_{<3,2>}$ immediately because $t_2 \geq T'$ is true. However, signal v_1 must stay in the high region for T' time; therefore, dashed arrows are used in $B_{<3,2>}$. For the second set where the condition is voilated, only one reachability computation is performed in state $B_{<2,1>}$. After leaving $B_{<2,1>}$, trajectories must stay in $B_{<3,1>}$ because both t_1 and t_2 are smaller than T'. It is similar for trajectories originated from states $B_{<1,3>}, B_{<3,1>}$ or $B_{<3,3>}$. From this analysis, we can see that there are totally 32 reachability computations as shown in Figure 3.24.

This approach can be extended to higher dimensions for circuits with more than two inputs. To reduce over-approximation error, t_{max} in the equation of T' can be replaced by smaller values such as $t_{max}/2$. However, this requires to partition regions 2 and 4 correspondingly, which increases the number of states and reachability computations.

4

Reachability Analysis in COHO

COHO is a reachability analysis tool for computing reachable regions of moderate-dimensional, nonlinear hybrid systems. The reachability algorithm is described in Section 4.1. The representation and computation of continuous successors are presented in Section 4.2 and Section 4.3. Section 4.4 describes techniques to improve computation time and reduce approximation error. Implementation issues are discussed in Section 4.5. Finally, Section 4.6 summarizes COHO and compares our approach with other related techniques.

4.1 Reachability Analysis

4.1.1 COHO Hybrid Automata

In order to analyze hybrid systems, we need a formal model to describe both continuous dynamics and discrete transitions. COHO uses a general hybrid automaton to model hybrid systems. A COHO *hybrid automaton* is a tuple $\mathbf{M} = (\mathbf{Q}, \mathbf{X}, \mathbf{F}, \mathbf{T}, \mathbf{I}, \mathbf{G}, \mathbf{R}, \mathbf{S}_0)$ where

- $\mathbf{Q}$ is a finite set of *discrete states*.
- $\mathbf{X} \subseteq \mathbb{R}^n$ is the *continuous state space*, where n is the number of continuous variables. $\mathbf{S} = \mathbf{Q} \times \mathbf{X}$ is the *state space* of the system.
- $\mathbf{I} : \mathbf{Q} \to 2^{\mathbf{X}}$ is a collection of *invariants*. $\mathbf{I}(q)$ is the condition that the continu-

ous variables must satisfy when the automaton is in the state q. The condition is described by a system of COHO inequalities, *e.g.*, $(x_1 \leq 1) \vee (x_1 + 2x_2 \geq 10)$ (see Section 4.2.2).

- $\mathbf{F} : \mathbf{Q} \to ((\mathbf{X} \to R^n) \cup (\mathbf{X} \to 2^{R^n}))$ is a set of *continuous dynamics*. For each state q, the evolution of continuous variables is governed by the deterministic or non-deterministic dynamics $\mathbf{F}(q)$.

- $\mathbf{T} \subseteq \mathbf{Q} \times \mathbf{Q}$ is a set of *discrete transitions*. Each transition $t = (q, q')$ identifies a *source state* q and a *target state* q'.

- $\mathbf{G} : \mathbf{Q} \to (2^{\mathbf{X}} \to \{0,1\})$ assigns each state a *guard condition* for specifying the pre-condition of discrete transitions. Given the current reachable region x in a state q, the condition $\mathbf{G}(q)(x)$ determines if discrete transitions are triggered or not.

- $\mathbf{R} : \mathbf{T} \to (2^{\mathbf{X}} \to 2^{\mathbf{X}})$ is a collection of *reset maps*. For each transition t, $\mathbf{R}(t)$ alters the continuous variables in the source state q, which will be used in the target state q'.

- $\mathbf{S}_0 \subseteq \mathbf{Q} \times \mathbf{X}$ is the *initial region* of the automaton. It consists of a set of discrete states $\mathbf{Q}_0 \subseteq Q$ and a set of initial regions for these discrete states $\mathbf{X}_0 = \mathbf{Q} \to 2^{R^n}$.

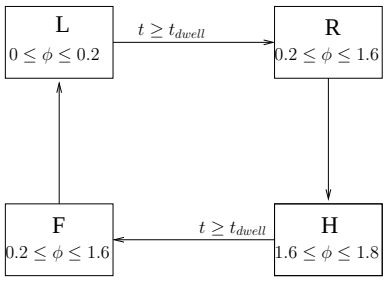


Figure 4.1: Hybrid Automaton for the Toggle Circuit

Figure 4.1 shows the hybrid automaton that we use when verifying the toggle circuit as described in Section 3.2.1 as an example. It has four discrete states L, R, H, F which correspond to Brockett's annulus regions (*i.e.*, low, rise, high, fall,

respectively), and seven continuous variables $X = \{\phi, x, y, z, xx, yy, zz\}$ which correspond to the circuit nodes as shown in Figure 3.3. For each discrete state, the invariant labeled in Figure 4.1 constraint the range of the clock variable ϕ. The continuous dynamics are defined by ODEs extracted from the circuit as described Section 3.3. There are four transitions: $(L,R),(R,H),(H,F),(F,L)$. For transitions (R,H) and (F,L), the guard condition is simply the invariant conditions, *i.e.,* a trajectory leaves state R or F once ϕ no longer satisfies the invariant. The other two transitions, (L,R) and (H,F) can only occur after the minimum dwell time in state L or H has elapsed as required by the Brockett's annulus construction. The reset map is set to be an identify function. The initial discrete state is R and the initial value of continuous variables is estimated based on simulations when ϕ switches from state L to state R.

4.1.2 Reachability Algorithm

We now turn to analyze behaviors of a hybrid automaton **M**. A *trajectory* of the hybrid automaton is a function $s : \mathbb{R}^+ \to \mathbf{S}$ that specifies the evolution of the system state according to time. A state of **M** can change in two ways: *discrete evolution* where the system changes the discrete state and *continuous evolution* where continuous variables change according to the dynamics. The *reachable region* of **M** is the set of all trajectories it can generate. In this dissertation, we use *reachable set* to denote the system states at a specified time t from an initial region, and use *reachable tube* to denote the region that is reachable prior to or at time t from the initial region. We only support forward reachability analysis. In principle, backward reachability can be implemented by negating the continuous dynamics [158]. However, modeling non-linear dynamics requires using over-approximations in the reachability computation to ensure soundness when verifying safety properties. The approximations that we use make COHO only useful for forward analysis where circuit dynamics tend to be well-damped.

Algorithm 1 shows the framework of our reachability analysis algorithm used in COHO. Given a hybrid automaton **M** and its initial region $\mathbf{S}_0$, it explores discrete and continuous successors until no new reachable region found. In the algorithm, we use a set Q to record all reachable discrete states and a set S to record reach-

Algorithm 1: Reachability Algorithm of Hybrid Systems in COHO.

Input: a COHO hybrid automaton $\mathbf{M} = (\mathbf{Q}, \mathbf{X}, \mathbf{T}, \mathbf{F}, \mathbf{I}, \mathbf{G}, \mathbf{R}, \mathbf{S_0})$,
$\mathbf{G} = \{\mathbf{C}, \mathbf{L}\}$, $\mathbf{S_0} = \{\mathbf{Q_0}, \mathbf{X_0}\}$

Output: reachable regions of the hybrid automaton

```
1  begin
2      Q = Q_0;
3      while q = pop(Q) do
4          S_0 = X_0(q), I(···) = Ø;
5          while C(q) do
6              S_{i+1} = post_c(S_i);
7              S_{[i,i+1]} = bloat(convex(S_i, S_{i+1}), ||ẋ||Δt);
8              if L(q) then
9                  for each gate g of I(q) do
10                     I(g) = union(intersect(S_{[i,i+1]}, g ∩ I(q)), I(g));
11                 end
12             end
13         end
14         for each transition t = (q, q', g) of T do
15             push(Q, q');
16             I(g) = R(t)(I(g));
17             X_0(q') = union(X_0(q'), I(g));
18         end
19     end
20 end
```

able regions in each state. Q is initialized by $\mathbf{Q}_0$ (line 2) and updated by adding unvisited states (lines 14,15,18) following discrete transitions. For each state q, S is initialized by its initial region $\mathbf{X}_0(q)$ (line 4). Forward reachable sets (line 6) and reachable tubes (line 7) are computed iteratively from the initial region. The computation of reachable sets is the most challenging task which will be described in Section 4.3 and Section 4.4. The algorithm depends strongly on the method used to represent continuous regions, and Section 4.2 presents the projectagon representation that is central to COHO. The reachable tube is approximated (line 7) by a bloated convex hull of reachable sets as illustrated in Figure 4.2. All reachable regions of the hybrid automaton $\mathbf{M}$ are computed once the algorithm terminates.

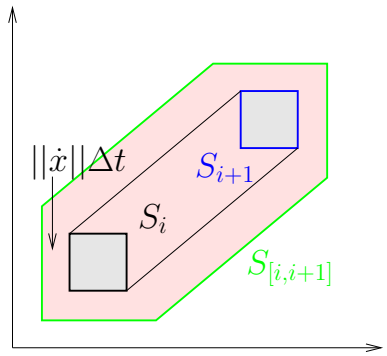


Figure 4.2: Approximate a Reachable Tube Based on Reachable Sets

As described above, each state q has an associated initial region $\mathbf{X}_0(q)$. To compute it efficiently (lines 14-18), we use *slices* to record trajectories that cross state boundaries. A slice is the intersection of a reachable tube and a hyperplane. Each condition of invariant $\mathbf{I}(q)$ defines such a hyperplane, which is also called a *gate*, of the state q^1. Slicing is performed in each computation step and the results are stored in a set I (line 10). The accumulated result is applied to update the initial region of successor states after the reset map (lines 16,17). With the concept of slice, a transition is implemented as a tuple $t = (q, q', g)$ where g is the gate of source state q (line 14). Accordingly, the guard condition $\mathbf{G}(q)$ consists of a *continuous function* $\mathbf{C}(q)$ and a *slicing function* $\mathbf{L}(q)$. The continuous function determines when to terminate the reachability computation (line 5) and the slicing function decides when to compute and record slices (lines 8-12) in each state. Take the automaton as shown in Figure 4.1 as an example: trajectories can leave state R at any time; therefore, reachability computation stops when all trajectories leave this state and slicing is performed in each computation step. However, in state H, the continuous function always returns true until a fixed-point is reached and slicing is performed only after the minimum dwell time has elapsed.

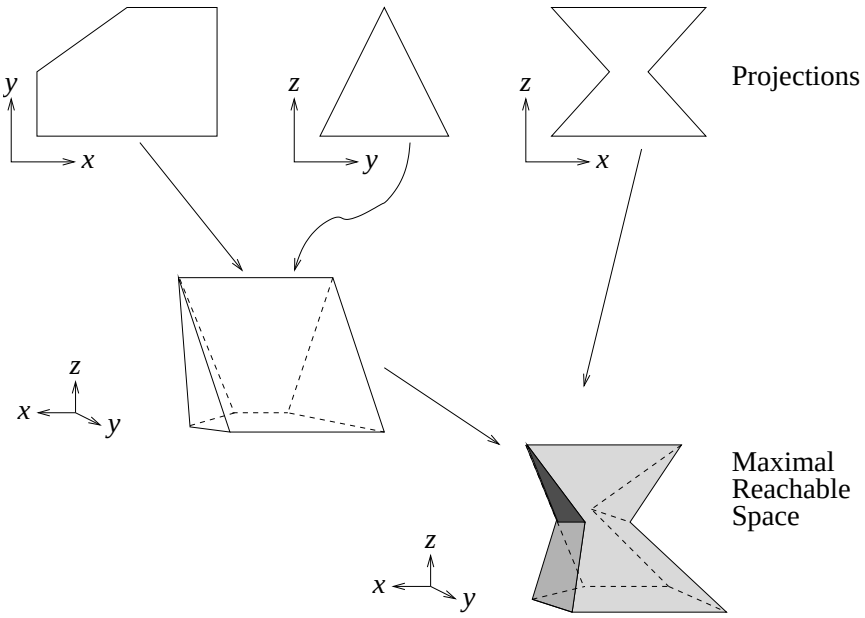


Figure 4.3: A Three-Dimensional "Projectagon"

4.2 Projectagons

Projectagons are a data structure for representing high-dimensional polyhedra by their *projections (projection polygons)* onto two-dimensional *planes*, where these projection polygons are not required to be convex. Conversely, a full-dimensional polyhedron can be obtained from its projections by back-projecting each projection polygon into a *prism* and computing the intersection of these prisms as shown in Figure 4.3. More formally, let $\mathfrak{S} \in \mathbb{R}^n$ be a n-dimensional polyhedra, $\mathfrak{B} = \{u_1, \cdots, u_n\}$ be an orthogonal basis, $\mathfrak{L} \subseteq \{(u_X, u_Y) | u_X \in \mathfrak{B}, u_Y \in \mathfrak{B}, X \neq Y\}$ be a set of planes. If $l = (u_{X(l)}, u_{Y(l)}) \in \mathfrak{L}$ is a plane, we write $p_l = proj(\mathfrak{S}, l)$ to denote the projection of $\mathfrak{S}$ onto this plane:

$$proj(\mathfrak{S}, l) = \{(x_{X(l)}, x_{Y(l)}) | (x_1, \cdots, x_n) \in \mathfrak{S}\}. \tag{4.1}$$

[1] In the current implementation of COHO, all gates of the guard condition are "exactly" on the boundary of the invariant regions.

We use $\mathfrak{P} = \{p_l | l \in \mathfrak{L}\}$ to denote the collection of all projections and write $prism(p_l)$ to denote the inverse projection of p_l back into the full-dimensional space:

$$prism(p_l) = \{(x_1, \cdots, x_n) \in \mathbb{R}^n | (x_{X(l)}, x_{Y(l)}) \in p_l\}. \tag{4.2}$$

The projectagon of a polyhedron $\mathfrak{S}$ with planes $\mathfrak{L}$ is $S_{\mathfrak{L}}(\mathfrak{S})$ where:

$$S_{\mathfrak{L}}(\mathfrak{S}) = \bigcap_{l \in \mathfrak{L}} prism(proj(\mathfrak{S}, l)). \tag{4.3}$$

In addition to this *geometric representation* $S_{\mathfrak{L}}(\mathfrak{S})$ where a projectagon is represented by a collection of projection polygons, we also provide an *inequality representation* $E_{\mathfrak{L}}(\mathfrak{S})$ for projectagons in Section 4.2.2 where the convex hull of a projectagon is over-approximated by a system of linear inequalities. Operations on projectagons are implemented based on these two representations which will be presented in Section 4.2.1 and Section 4.2.2. To convert from the geometric representation to a system of linear constraints, we compute the convex hull of each projection polygon as described in Section 4.2.1. To convert from a system of linear constraints back to projection polygons, we project the feasible region of the constraints onto the appropriate plane(s) described later in Section 4.3.2.

Projectagons can represent non-convex polyhedra efficiently, which is not supported by most other techniques discussed in Section 2.2.3. However, projectagons are not a canonical representation (see Appendix A). Furthermore, there are many polyhedra (even convex ones) that cannot be represented by projectagons exactly. For example, indentations on the surface of the full-dimensional polyhedron will be filled; likewise, many perforated objects and knot-like objects can only be approximated. However, an attractive feature of this approach is it always overestimates the original polyhedron; in particular, it is straightforward to show that:

$$\mathfrak{S} \subseteq S_{\mathfrak{L}}(\mathfrak{S}). \tag{4.4}$$

The approximation error also depends on the set of planes $\mathfrak{L}$ used. Generally speaking, approximation error decreases when the number of planes increases, which lies in the range of $[\lceil \frac{n}{2} \rceil, \frac{n(n-1)}{2}]$.

The projectagon representation offers several advantages over other approaches. First, projection polygons capture circuit designers' intuitive notion of how a circuit works. Typically, the behavior of each signal is determined by a small number of other signals. Pairing a node with each of its controlling nodes naturally captures the causal behavior of the circuit. Because most circuits have limited fan-in and fan-out, the number of such pairs, *i.e.*, planes, is proportional to the number of nodes in the circuit. Second, the geometric representation can capture important non-convexities of the reachable space and provides efficient implementations of key operations including intersection and union. These operations are described in Section 4.2.1. Furthermore, representing the convex hull of a projectagon as a system of linear inequalities allows us to use methods from linear programming to manipulate projectagons which are discussed in Section 4.2.2. This is particularly useful for operations that work on one face of the projectagon at a time, and we approximate the face by its convex hull even though the complete projectagon may be non-convex. Finally, ignoring degeneracies, faces of a projectagon correspond to edges of its projection polygons. Section 4.2.3 presents an efficient method to enumerate all projectagon faces which is an important step for the reachability algorithm in Section 4.3.3.

4.2.1 Manipulating Projectagons via Geometry Computation

The geometric representation of projectagons only tracks two-dimensional projections rather than full-dimensional polyhedra. Therefore, exponential time and space operations on full-dimensional polyhedra can be avoided because the operations needed for reachability computations can be implemented based on polynomial time operations on the two-dimensional projection polygons. For example, intersection and union of two projectagons can be computed by

$$intersection(S^1_{\mathfrak{L}}, S^2_{\mathfrak{L}}) = \bigcap_{l \in \mathfrak{L}} intersection(p^1_l, p^2_l) \tag{4.5}$$

$$union(S^1_{\mathfrak{L}}, S^2_{\mathfrak{L}}) \subseteq \bigcap_{l \in \mathfrak{L}} union(p^1_l, p^2_l). \tag{4.6}$$

Two other operations required by COHO are convex hull and projectagon simplification. COHO's reachability computation tends to increase the number of ver-

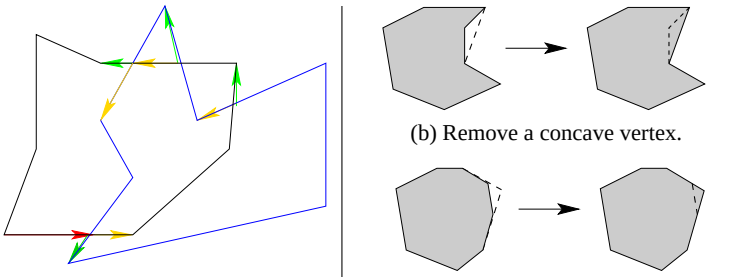
tices of each projection polygon at each time-step. As described below, polygon simplification introduces small over-approximations to keep the polygon degrees manageable. Like intersection and union, both convex hull and projectagon simplification can be computed on a per-projection-polygon basis as indicated below:

$$\mathit{convexhull}(S_{\mathfrak{L}}) \subseteq \bigcap_{l \in \mathfrak{L}} \mathit{convexhull}(p_l) \tag{4.7}$$

$$\mathit{simplify}(S_{\mathfrak{L}}, \varepsilon) \stackrel{\Delta}{=} \bigcap_{l \in \mathfrak{L}} \mathit{simplify}(p_l, \varepsilon). \tag{4.8}$$

The intersection of a projectagon and a hyperplane can be implemented similarly. However, the intersection of a projection polygon and the corresponding projection of a hyperplane is a segment. To keep projectagons closed under intersection with hyperplanes, we use a projectagon to conservatively over approximate the intersection, which will be described with details in Section 4.3.3.

There are efficient algorithms and well-developed tools for polygon operations [171]. However, we found that COHO requires robust implementations of these operations for polygons with nearly parallel edges and similar difficulties. To achieve this, we implemented a new geometry computation package based on *arbitrary precision rational (APR)* numbers to provide robust implementations especially for ill-conditioned problems. For the union operation, our algorithm first computes all intersection points of two or more polygons using the *sweep-line* algorithm [171, Chapter 7.2]. It then finds the union of these polygons by walking from the lower-left-most point in the anti-clock wise order and always selecting the right most edge on each intersection point. An example is shown as green arrows in Figure 4.4(a). The algorithm for the intersection operator is similar except it chooses the left most edge on each intersection point, illustrated as yellow arrows in Figure 4.4(a). To simplify a polygon, our algorithm finds an over-approximated polygon by either deleting a concave vertex or replacing two consecutive, convex vertices with a single vertex of the polygon as shown in Figure 4.4(b) and Figure 4.4(c) respectively.



(a) Union/Intersection of polygons

(b) Remove a concave vertex.

(c) Replace two convex vertices with one.

Figure 4.4: Polygon Operations

4.2.2 Manipulating Projectagons via Linear Programming

The geometric representation can represent non-convex regions accurately. However, it is not efficient for some operations, such as the *project* operation of non-convex polyhedra, and the intersection of prisms. On the other hand, *convex projectagons* have an inequality representation which represents a half-plane by a linear inequality. This representation has an attractive feature that it corresponds to the constraints of a *linear program* which makes it possible to manipulate convex projectagons using techniques from linear programming. Therefore, we also employ an inequality representation to describe the convex hulls of projectagons and obtain efficient algorithms for several operations.

The convex hull of a projectagon, which can be approximated efficiently using Equation 4.7, is described by a COHO *constraint system* in the inequality representation $E_{\mathfrak{L}}(\mathfrak{S})$. A COHO constraint system is a system of inequalities of the form:

$$E_{\mathfrak{L}}(\mathfrak{S}) = Px \leq q = \bigcap_{l \in \mathfrak{L}} \{P_l x_l \leq q_l | \forall x \in \mathit{convexhull}(p_l)\}, \tag{4.9}$$

where the matrix P, called a COHO *matrix*, has only one or two non-zero elements in any row, because each COHO *inequality* corresponds to a polygon edge. Likewise, a COHO equality is a linear equality constraint that involves only one or two variables. The special structure of the COHO matrix is exploited to develop an efficient LP solver in [208, 214].

The inequality representation $E_{\mathfrak{L}}(\mathfrak{S})$ conservatively approximates the convex

hull of the original polyhedron as

$$\mathfrak{S} \subseteq \mathit{convexhull}(\mathfrak{S}) \subseteq E_{\mathfrak{L}}(\mathfrak{S}). \tag{4.10}$$

This representation is used to implement the bloating operator by moving each half-plane outward as

$$\mathit{bloat}(S_{\mathfrak{L}}, \Delta d) \subseteq \mathit{bloat}(E_{\mathfrak{L}}(\mathfrak{S}), \Delta d) = Px \leq q + \Delta d. \tag{4.11}$$

It is also used to implement the project operator in Section 4.3.2, represent projectagon faces, compute bounding boxes of projectagons, *etc.* The intersection of prisms represented by inequalities is trivial by the conjunction of all COHO inequalities.

4.2.3 Projectagon Faces

Finding projectagon faces is an important operation because all computations are performed on projectagon faces in Algorithm 4 of Section 4.3.3. Fortunately, projectagon faces correspond to edges of projection polygons. Therefore, if e is a projection edge on the plane l, its corresponding projectagon face $f(e,l)$ is

$$f(e,l) = \mathit{prism}(e) \bigcap S_{\mathfrak{L}}(\mathfrak{S}). \tag{4.12}$$

The prism corresponding to an edge, $\mathit{prism}(e)$ can be described by one COHO equality and two COHO inequalities. However, it is difficult to compute the intersection of a projectagon and these inequalities. We use the inequality representation instead to compute a conservative result:

$$f(e,l) \subseteq \mathit{prism}(e) \bigcap E_{\mathfrak{L}}(\mathfrak{S}). \tag{4.13}$$

The result is accurate for convex projectagons. However, for non-convex projectagons, the approximation error can be very large. Therefore, we developed an *interval closure* technique for finding a more accurate representation of projectagon faces. The interval closure calculation is based on interval constraint propagation. It interprets a non-convex polygon as constraints of continuous variables

and views each polygon edge as defining interval bounds for the two variables of the projection. The algorithm then applies these intervals to other polygons that include one of these variables in their basis to obtain bounds of other variables. This process continues until no further tightening of the interval bounds is possible or the progress is below a threshold. The algorithm is simple, fast and significantly reduces the approximation error when the projection polygons are highly non-convex. The interval closure based projectagon faces can be expressed as:

$$f(e,l) \subseteq prism(e) \bigcap E_{\mathfrak{L}}(\mathfrak{S}) \bigcap intervalClosure(e, S_{\mathfrak{L}}(\mathfrak{S})). \tag{4.14}$$

4.3 Computing Continuous Successors

This section presents our algorithm for computing continuous successors used in Algorithm 1. First, the method to move projectagon faces forward in time is shown in Section 4.3.1. Second, advanced faces are projected onto two-dimensional planes to maintain the structure of projectagons, as described in Section 4.3.2. Finally, the algorithm for computing continuous successors is presented in Algorithm 4 of Section 4.3.3 and the feasibility problem of projectagons is discussed.

4.3.1 Advancing Projectagon Faces

As shown in Algorithm 4 (line 13), an essential step of computing continuous successors is to advance projectagon faces according to nonlinear dynamics. A projectagon face is represented by a COHO constraint system as described in Section 4.2.3. In this section, we consider the problem of computing an over-approximation $\delta_t(S)$ of the reachable set at time t from an initial region S, which is described by a COHO constraint system $Px \leq q$. Nonlinear dynamics are conservatively approximated by a linear differential inclusion (LDI) of the form

$$\dot{x} = Ax + b \pm u, \tag{4.15}$$

where the u term defines a uncertainty set U for which we use a hyper-rectangle.

These non-deterministic dynamics are approached with the *maximum principle* from optimal control theory [197]. The key idea is to find a *critical value* $u^* \in U$

for each face of S which moves its corresponding hyperplane furthest outward. A face f_i of S is represented by an inequality from the COHO constraint system as

$$P_i x \leq q_i, \tag{4.16}$$

where P_i is the i^{th} row of COHO matrix P and q_i is the i^{th} element of vector q. Equation 4.16 also defines a halfspace where $\vec{n} = P_i^T$ is the *outward normal* of the corresponding hyperplane.

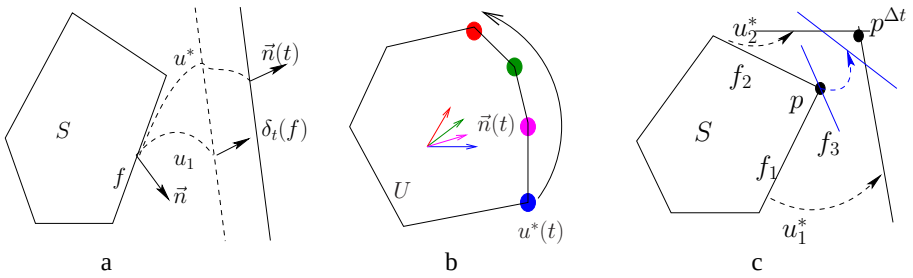


Figure 4.5: Maximum Principle. a) the normal $\vec{n}(t)$ is determined by the linear system; b) find critical value by linear programming c) redundant faces can reduce approximation error.

By integrating Equation 4.15, we have

$$x_u(t) \quad = \quad e^{At}x(0) + (e^{At} - I)A^{-1}b + \int_0^t e^{A(t-s)}u(s)\,ds.$$

We remark that all initial points satisfy the COHO constraint system $Px(0) \leq q$; therefore, we have

$$P_i e^{-At} \cdot x_u(t) \quad \leq \quad q_i + P_i(I - e^{-At})A^{-1}b + \int_0^t P_i e^{-As} \cdot u(s)\,ds. \tag{4.17}$$

As illustrated in Figure 4.5(a), we can see that the evolution of the normal to f_i does not depend on the uncertainty term u, and is only governed by the linear term as

$$\vec{n}(t) = e^{-A^T t} P_i^T. \tag{4.18}$$

To bound all trajectories with all possible values of the uncertainty term u, we

maximize the integration in Equation 4.17 as

$$\vec{n}^T(t)x_u(t) \quad \leq \quad q_i + P_i(I - e^{-At})A^{-1}b + \max_{u:[0,t]\to U} \int_0^t (\vec{n}^T(s)u(s))\,ds. \qquad (4.19)$$

It is straightforward to compute $q_i + P_i(I - e^{-At})A^{-1}b$, and we now derive a bound for the integral.

$$\begin{aligned}
& max_{u:[0,t]\to U} \int_0^t \vec{n}^T(s)u(s)\,ds \\
= \quad & max_{u:[0,t]\to U} \int_0^t \vec{n}^T e^{-As}u(s)\,ds \\
= \quad & max_{u:[0,t]\to U} \int_0^t \vec{n}^T \left(\sum_{k=0}^{\infty} \frac{(-As)^k}{k!} \right) u(s)\,ds \\
\leq \quad & max_{u:[0,t]\to U} \int_0^t \vec{n}^T \left(\sum_{k=0}^{m} \frac{(-As)^k}{k!} \right) u(s)\,ds \\
& + max_{u:[0,t]\to U} \int_0^t \vec{n}^T \left(\sum_{k=m+1}^{\infty} \frac{(-As)^k}{k!} \right) u(s)\,ds
\end{aligned} \qquad (4.20)$$

where m is chosen to set the degree of the approximation of the matrix exponential. In practice, $\|At\|$, is small, and $m = 1$ or $m = 2$ will produce a very small overapproximation. Let

$$\begin{aligned}
G \quad = \quad & max_{u:[0,t]\to U} \int_0^t \vec{n}^T \left(\sum_{k=0}^{m} \frac{(-As)^k}{k!} \right) u(s)\,ds \\
H \quad = \quad & max_{u:[0,t]\to U} \int_0^t \vec{n}^T \left(\sum_{k=m+1}^{\infty} \frac{(-As)^k}{k!} \right) u(s)\,ds
\end{aligned} \qquad (4.21)$$

For small m, G can be computed exactly by considering one component of u at a time. To bound H, observe

$$\begin{aligned}
H \quad = \quad & max_{u:[0,t]\to U} \int_0^t \vec{n}^T \left(\sum_{k=m+1}^{\infty} \frac{(-As)^k}{k!} \right) u(s)\,ds \\
\leq \quad & max_{u:[0,t]\to U} \int_0^t \|\vec{n}^T\| \left\| \sum_{k=0}^{m} \frac{(-As)^k}{k!} \right\| \|u(s)\|\,ds \\
= \quad & t\|\vec{n}\| \left(e^{\|At\|} - \sum_{k=0}^{m} \frac{\|At\|^k}{k!} \right) \|U\|
\end{aligned} \qquad (4.22)$$

This provides an overapproximation of the solution to Equation 4.19 providing the bound we want for the most outward foward face $\delta_t(f_i)$. Thus, the forward reachable region is bounded by

$$\delta_t(S) \subseteq \bigcap_{f_i \in S} \delta_t(f_i). \tag{4.23}$$

The actual implementation of COHO is based on an earlier approximation that we derived but that can be underapproximate. The COHO code is based on the assumption that the critical value does not change during the time step and approximating the forward region $\delta_t(S)$. With this assumption, the integral is underapproximated slightly as:

$$\begin{aligned} \int_0^t \vec{n}^T(t)u^*(s)\,ds &\approx \max_{u\in U} \int_0^t \vec{n}^T(s)u\,ds \\ &= |P_i(I-e^{-At})A^{-1}|u. \end{aligned}$$

Therefore, the forward reachable region is bounded by a COHO *advanced constraint* of the form:

$$\begin{aligned} PEx &\leq \hat{q} \\ E &= e^{-At} \\ \hat{q} &= q + P(I-E)A^{-1}b + |P(I-E)A^{-1}|u, \end{aligned} \tag{4.24}$$

where the matrix E is a linear operator for moving a point at the end of a time step back to the original point at the beginning of the time step. We ran some experiments to quantify the effect of this under-approximation for the examples described in Chapter 5. In all cases, any under-approximation was *very* small, and we are convinced that the over-approximations in other parts of COHO ensure that, in practice, the overall computation is an overapproximation as desired. Both approximation methods have the advantage that they are relatively efficient because only matrix operations rather than numerical integration are involved during the computation.

Finally, we add redundant faces to the initial region S to reduce approximation error. As illustrated in Figure 4.5(c), a vertex p lies on two faces f_1 and f_2, how-

ever, the critical values u_1^* and u_2^* of these two faces are not necessarily the same. Therefore, the intersection point $p^{\Delta t}$ of the two forward faces is not reachable from the point p. The redundant face f_3 can trim unreachable regions and thus reduce overapproximation error.

4.3.2 COHO Linear Program Solver and Projection Algorithm

COHO makes extensive use of linear programs (LPs) which have either COHO constraint systems from Equation 4.9 or COHO advanced constraint systems from Equation 4.24. This section presents an overview of our linear program solver and projection algorithm based on my Master's work [208]. The linear programs used on COHO can be written as

$$\begin{aligned} \min_x d^T \cdot x, \text{ s.t.} \\ P \cdot E \cdot x \quad \leq \quad q, \end{aligned} \tag{4.25}$$

where $P \cdot x \leq q$ is a COHO constraint system whose feasible region is a convex projectagon, d is the cost vector, and E is an optimal backward time step operator for LDI models. Because time steps are relatively small, E is well-conditioned, and E^{-1} can be easily computed. We refer to a linear program in the form of Equation 4.25 as a COHO *linear program*.

There are two main approaches for solving linear programs: *interior point algorithms* [156, 164] and the *Simplex* algorithm [80]. However, the inherent ill-conditioning of interior point methods compounded the problem of badly conditioned linear programs and prevented their successful application. On the other hand, the Simplex algorithm is typically formulated to operate on standard form linear programs. A *standard form* linear program has the form [165]:

$$\min\ c^T \cdot x,\ s.t.\ Ax \leq b. \tag{4.26}$$

To retain the special structure of COHO matrices, we convert a COHO LP to its dual

form [165] as

$$\begin{array}{rcl} \min\limits_{y} & -q^T \cdot y, \text{ s.t.} & \\ P^T \cdot y & = & E^{-T} d \\ y & \geq & 0. \end{array} \tag{4.27}$$

The COHO *dual LP* is a standard form linear program. Therefore, it can be solved by the Simplex algorithm. By the *duality theorem* [196, Chapter 5], the primal and dual linear programs have the same optimal value and optimal basis; thus, the COHO LP can also be solved.

Simplex is a greedy algorithm. It repeatedly selects a subset of the columns of P^T called the *basis*. Let $P^T_{\mathscr{B}}$ denote P^T restricted to a basis $\mathscr{B}$. Simplex solves for $y_{\mathscr{B}} = P^{-T}_{\mathscr{B}} \cdot d$ and determines if the basis can be modified so as to improve the cost. This step is called *pivoting* which replaces a column in the basis with a new column to obtain a more favorable basis. Simplex pivots to reduce the cost until the optimal basis is found.

Algorithm 2: COHO LP Solver (lp_solve(LP))

Input: $\mathsf{LP}(P,q,d,E)$: a COHO LP of the form: $\min d^T x, s.t. PEx \leq q$
Output: B: optimal basis, pt: optimal point, v: optimal value

```
1  begin
2      A = −P^T; b = −E^{−T} d; c = −q;
3      dLP = lp_standard(A,b,c);
4      B = BigM(dLP);
5      repeat
6          c_b = A_B^{−T} c_B;
7          j = arg min_{i∉B} (c_i − A_{:,i}^T c_b < 0);
8          t_0 = A_B^{−1} b; t_j = A_B^{−1} A_{:,j};
9          k = arg min_{i∈B} t_{0,i}/t_{j,i};
10         B = replace(B, k, j);
11     until B = null ;
12     pt = E^{−1} P_B^{−1} q_B; v = d^T pt;
13 end
```

COHO uses a modified Simplex algorithm as shown in Algorithm 2. It first converts COHO LPs to the dual forms (lines 2,3) and computes an initial feasible basis

based on the *Big M method* [196, Chapter 2.3] (line 4). It uses a standard pivoting algorithm (lines 6-10). However, COHO computes the Simplex tableau from the input data for each pivot which is different from most implementations where the tabulea is updated based on its predecessor [80]. This modification avoids accumulated error because only a vector of integers specifying the basis is passed through each pivot. The algorithm is as efficient as the traditional implementation based on rank-1 updates of the tableau, because COHO's solver exploits the structure of the COHO matrix P to obtain a linear time algorithm for solving linear systems on lines 6, 8 and 12. More details of the linear system solver are described in [208, 214].

To maintain the projectagon structure, advanced faces in Equation 4.24 need to be projected onto two-dimensional planes to compute projections for constructing a new projectagon. Likewise, projectagons are re-projected to remove infeasible regions which will be discussed in Section 4.3.3. Therefore, an important operation in COHO is finding the *projected polygon* of a convex region described by a COHO constraint system $Px \leq q$ or COHO advanced constraint system $PEx \leq q$ onto a plane defined by two orthogonal axes $(\vec{x}, \vec{y})$. We call this problem the *projection problem* and present an algorithm based on linear programming for projection problems with COHO constraint systems and extend it to problems with COHO advanced constraint systems.

The idea behind our projection algorithm in Algorithm 3 is to solve COHO LPs

$$\max_{x \in \mathbb{R}^n} (\vec{x}\cos\theta + \vec{y}\sin\theta) \cdot x \quad \text{s.t. } Px \leq q, \tag{4.28}$$

for all θ from 0 to 2π and use the optimal points to construct the projected polygon. It is unnecessary and impossible to solve Equation 4.28 for every possible value of θ. Instead, COHO only solves one LP for each edge of the projected polygon, where the optimal direction $\vec{x}\cos\theta + \vec{y}\sin\theta$ is also the normal vector of the edge.

The normal vector of a polygon edge is computed based on the optimal basis of the COHO LP in Equation 4.28. Our Simplex based solver works on its dual problem:

$$\min_{y \in \mathbb{R}^{+m}} q \cdot y \quad \text{s.t. } P^T y = \vec{x}\cos\theta + \vec{y}\sin\theta \ , y \geq 0. \tag{4.29}$$

When the solver finds a solution to Equation 4.29, it also finds an optimal basis

$\mathscr{B}$ and an optimal point $pt = P_{\mathscr{B}}^{-T}(\vec{x}\cos\theta + \vec{y}\sin\theta)$ whose elements are all non-negative. By increasing the value of θ to some critical value, the basis $\mathscr{B}$ will no longer be optimal for the optimization direction $\vec{x}\cos\theta + \vec{y}\sin\theta$. The critical value of θ is the one θ_{next} (line 11) at which pt acquires a negative element, and the corresponding direction is orthogonal to the polygon edge that is from the current optimal vertex to the new optimum. Each successive value of θ can be determined by a single linear system solve in linear time. Similarly, the normal vector of the edge from the previous optimal vertex to the current one can be computed by decreasing the value of θ to θ_{prev} (line 10) as shown in Figure 4.6(a).

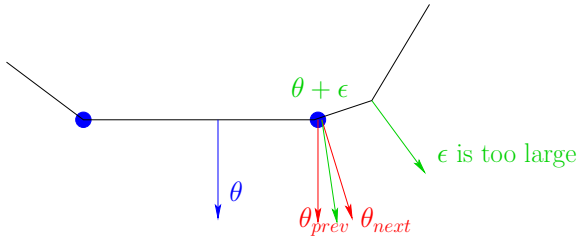


Figure 4.6: Projection Algorithm: a) finding θ_{next} and θ_{prev} from the optimal basis; b) both blue vertices are optimal for the normal vector; c) an edge is skipped when ε is too large.

However, using the normal of a polygon edge as the optimal direction may not find the expected optimal basis, because several projectagon vertices may correspond to the same projection polygon vertex as illustrated in Figure 4.6(b). Therefore, we increase the value of θ by a small amount ε (line 5) to force the LP solver to find the correct optimal basis and optimal point. However, this may skip some edges and produce under-approximated results as shown in Figure 4.6(c). We detect this scenario by comparing the value of θ and θ_{prev}. If there is no edge has been skipped in the step, θ_{prev} must equal to θ. Otherwise, the mean value of θ and θ_{prev} is used as the new optimization direction (line 12) to repeat the computation.

The algorithm is generalized to projection problems with COHO advanced constraint systems for projecting a time advanced projectagon $P \cdot E \cdot x \leq q$. Although the feasible region of a COHO advanced constraint system is not a projectagon in the standard coordinate $\mathscr{C}$, it can be represented as a projectagon of the form $P \cdot x \leq q$ in the new coordinate $E^{-1}\mathscr{C}$. Therefore, the problem is converted to

project $P \cdot x \leq q$ on to a subspace with basis vectors $E^{-1}\vec{x}$, and $E^{-1}\vec{y}$ (lines 2,3,16). The algebraic and geometric explanations of this coordinate transformation can be found in [208].

Algorithm 3: COHO LP Projection Algorithm (lp_project($PEx \leq q$,$\vec{x}$,$\vec{y}$))

Input: COHO advanced constraints: $PEx \leq q$
Input: projection plane $(\vec{x},\vec{y})$
Output: p: projection onto $(\vec{x},\vec{y})$ plane

1 **begin**
2 $\quad \vec{x} = E^{-1}\vec{x};\ \vec{y} = E^{-1}\vec{y};$
3 $\quad \vec{y} = \vec{y} - \frac{\vec{x}\cdot\vec{y}}{||\vec{x}||};\ \vec{x} = \frac{\vec{x}}{||\vec{x}||};\ \vec{y} = \frac{\vec{y}}{||\vec{y}||};$
4 $\quad$ **for** $\theta = 0, \theta \leq 2\pi$ **do**
5 $\quad\quad \theta_{opt} = \theta + \varepsilon;$
6 $\quad\quad$ **repeat**
7 $\quad\quad\quad d = \vec{x}\cos(\theta_{opt}) + \vec{y}\sin(\theta_{opt})$; lp = lp_standard($P, q, d$);
8 $\quad\quad\quad$ [B, pt, v] = lp_solve(lp);
9 $\quad\quad\quad \zeta = P_{\mathsf{B}}^{-T}\vec{x};\ \eta = P_{\mathsf{B}}^{-T}\vec{y};$
10 $\quad\quad\quad \theta_{prev} = \max_{i\in[1,n]}(\arg_{\theta}\min(\zeta_i\cos\hat{\theta} + \eta_i\sin\hat{\theta} = 0 \wedge \hat{\theta} < \theta_{opt}));$
11 $\quad\quad\quad \theta_{next} = \min_{i\in[1,n]}(\arg_{\theta}\min(\zeta_i\cos\hat{\theta} + \eta_i\sin\hat{\theta} = 0 \wedge \hat{\theta} > \theta_{opt}));$
12 $\quad\quad\quad \theta_{opt} = \frac{\theta + \theta prev}{2};$
13 $\quad\quad$ **until** $\theta = \theta_{prev}$;
14 $\quad\quad \theta = \theta_{next};\ \mathsf{p} = \mathsf{p} + project(\mathsf{pt}, \vec{x}, \vec{y});$
15 $\quad$ **end**
16 $\quad p = convexhull(p);\ p = E^{-1}p;$
17 **end**

As we can see, the correctness of the LP solver depends on correct pivots and the projection algorithm relies on correct computation of θ_{prev} and θ_{next}. However, numerical error of floating point arithmetic may result in an incorrect pivot and fail to find an optimal solution in the Simplex algorithm, especially for ill-conditioned COHO LPs. Therefore, *arbitrary precision rational (APR)* arithmetic is applied in the LP solver and the projection algorithm to make COHO robust. If APR methods were applied to all of COHO's computations, the resulting implementation would be very slow. As described in Section 4.4.4, COHO uses interval-based methods where their accuracy is sufficient and resorts to APR only when necessary.

4.3.3 Computing Forward Reachable Sets

Algorithm 4 shows the procedure of computing a forward reachable set in COHO. Because the system dynamics are bounded, trajectories are continuous and cannot cross. Therefore, trajectories starting on projectagon faces provide bounds for trajectories starting in the interior. Accordingly, the algorithm first finds all projectagon faces using the interval closure approximation in Section 4.2.3 (line 4). Then it calculates a linear differential inclusion (LDI) model for each face and computes a step size Δt (lines 5-7,10). To ensure soundness, it uses a conservative strategy to compute the model and step size based on a constraint that any point in the projectagon can move by at most a user-provided distance Δd along any axis-parallel direction. It bloats each face outward by Δd (line 5) and approximates the system dynamics by a LDI in the bloated face (line 6). Based on the LDI model, it computes the maximum derivative in the bloated face by solving two sets ($\forall i \in \{1, \cdots, n\}$) of linear programs (line 7) as

$$\begin{array}{l} \min / \max \dot{x}_i, \text{ s.t.} \\ \quad \dot{x} \leq Ax + b + u \\ \quad -\dot{x} \leq -(Ax + b - u) \\ \quad Px \leq q, \end{array} \tag{4.30}$$

which can be solved by the LP solver in Section 4.3.2. The step size is calculated by computing the maximal value $\Delta t = \frac{\Delta d}{\max |\dot{x}_i|}$ (line 10) which guarantees that the reachable tube is bounded by the bloated face during this step $[0, \Delta t]$.

Given LDI models and a step size, projectagon faces are advanced by using Equation 4.24 (line 13). To maintain the structure of projectagons, all advanced faces are projected onto two-dimensional planes using the method described in Section 4.3.2. However, each advanced face must be projected onto all planes of the original projectagon (line 15). Intuitively, this is because the projectagon can rotate during this step and any face can become an external face for any plane. Finally, a new projectagon for the forward reachable set is constructed by computing the union of all projected polygons on the same plane and simplifying the result to reduce space complexity (lines 19-22).

During the computation above, projection polygons are computed indepen-

Algorithm 4: Algorithm for Computing Forward Reachable Set

Input: $S_{\mathfrak{L}}$: current reachable set represented by a projectagon
Input: Δd: maximum moving distance allowed
Output: $S_{\mathfrak{L}}^{\Delta t}$: the forward reachable set after time step Δt

begin
  for *each projection* p **do**
    for *each edge* e **do**
      $f_e = prism(e) \cap E_{\mathfrak{L}}$, $f_e = f_e \cap intervalClosure(e, S_{\mathfrak{L}})$ if $f_e \neq \emptyset$;
      $f_e^b = bloat(f_e, \Delta d)$;
      $model_e = model\_create(f_e^b)$;
      $|\dot{x}|_e = lp\_solve(f_e^b, model_e)$;
    end
  end
  $\Delta t = \min_{e \in S_{\mathfrak{L}}} \frac{\Delta d}{\max |\dot{x}|_e}$;
  for *each projection* p **do**
    for *each edge* e **do**
      $f_e^{\Delta t} = lp\_forward(f_e, model_e, \Delta t)$;
      for *each plane* $l \in \mathfrak{L}$ **do**
        $poly_e^l = lp\_project(f_e^{\Delta t}, \vec{x}_l, \vec{y}_l)$;
      end
    end
  end
  for *each plane* $l \in \mathfrak{L}$ **do**
    $p_l^{\Delta t} = poly\_union_{e \in S_{\mathfrak{L}}}(poly^l)$;
    $p_l^{\Delta t} = poly\_simplify(p_l^{\Delta t})$;
  end
  $S_0^{\Delta t} = \bigcap_{l \in \mathfrak{L}} prism(p_l^{\Delta t})$; $E_0^{\Delta t} = \bigcap_{l \in \mathfrak{L}} prism(convexhull(p_l^{\Delta t}))$;
  repeat
    for *each plane* $l \in \mathfrak{L}$ **do**
      $poly_l^i = lp\_project(E_{i-1}^{\Delta t}, \vec{x}_l, \vec{y}_l)$;
      $p_l^i = poly\_intersect(p_l^{i-1}, poly_l^i)$;
      $p_l^i = poly\_simplify(p_l^i)$;
    end
    $S_i^{\Delta t} = \bigcap_{l \in \mathfrak{L}} prism(p_l^i)$; $E_i^{\Delta t} = \bigcap_{l \in \mathfrak{L}} prism(convexhull(p_l^i))$;
  until $||E_{i-1}^{\Delta t} - E_i^{\Delta t}|| \leq \varepsilon$;
  $S_{\mathfrak{L}}^{\Delta t} = S_i^{\Delta t}$; $E_{\mathfrak{L}}^{\Delta t} = E_{i-1}^{\Delta t}$;
end

dently for each plane. Therefore, it is possible that there are *infeasible regions* of one projection to other projections, *i.e.*, the prism from this region does not intersect with other prisms of other polygons. The infeasible regions of the new projectagon may lead to incomplete boundaries in the next step when working on an infeasible face. To solve the problem, all projection polygons must be clipped to make them feasible to each other. Section A.1 shows that the problem of determining whether a projectagon constructed from a set of non-convex polygons is non-empty is NP-complete. Thus, we accept that a practical algorithm must be based on heuristics and/or approximations.

We relax the problem of removing infeasible regions as clipping a projectagon such that each of its projections is feasible to the inequality representation of the projectagon, *i.e.*, convex hulls of other projections. The algorithm is based on the projection algorithm in Section 4.3.2. As shown in line 23 of Algorithm 4, the inequality representation of the new projectagon is constructed from the convex hulls of projection polygons. This representation can include non-tight or redundant constraints from infeasible edges. To obtain a canonical representation, we project the COHO constraint system onto all planes (line 26) and construct a new set of inequalities from the projected polygons. It is easy to show that all these projected polygons are feasible to each other. Therefore, the intersection of an original projection and its corresponding updated projected polygon (line 27) is still feasible to the new inequality representation. This algorithm clips a projectagon and makes it feasible to its inequality representation.

The inequality representation must be refined because the intersection makes a projectagon polygon smaller but also changes its convex hull. The convex hulls of intersected polygons are computed and the inequality representation is reconstructed at the end of each step (line 30). This procedure is repeated until all convex hulls are fixed. However, it may take infinite number of steps to converge to the stable projectagon[2]. To solve the problem, we stop the computation after a certain number of iterations or the progress is smaller than a threshold (line 31). We do not update the inequality representation in the last step (line 32). Therefore, the inequality representation at the end is an over-approximation rather than an exact

[2] An example is provided in Section A.2.

representation of the convex hull of the projectagon.

This method has been applied in Algorithm 4 to make the boundary complete. It ensures that projections of any new projectagon are feasible to its inequality representation. As show in line 4, we use Equation 4.14 to obtain a tighter approximation of a projectagon face if the face is feasible; otherwise, we use Equation 4.13 to approximate the projectagon face to ensure the approximation is not empty. To control space complexity, intersected polygons are also simplified (line 28). However, only concave vertices can be removed because replacing two convex vertices with one can enlarge the convex hull and make it infeasible to the inequality representation.

Noting that we can add arbitrary COHO constraint systems to the inequality representation during the projecting step (line 26), the algorithm is extended to compute the intersection of a projectagon and a COHO constraint system. Actually, the intersection of a projectagon and a hyper-plane required in Algorithm 1 is over-approximated by the intersection of the projectagon and a bloated hyper-plane. This is because the intersection of a projectagon and hyper-plane is not a full-dimensional projectagon. By bloating the hyperplane slightly, we avoid the need to handle special cases for objects that are not full-dimensional, and this makes the implementation much simpler.

4.4 Improvements

The algorithm described above is numerically stable; however, the reachability computation can be very expensive especially for high-dimensional systems. Furthermore, the over-approximated result may be too large to verify a correct system successfully. To improve performance and accuracy, several new algorithms have been developed as shown in Algorithm 5. Section 4.4.1 presents a method which projects advanced faces onto one plane to reduce projection error and improve performance (line 7). Section 4.4.2 proposes a guess-verify strategy for increasing the step size and reducing error (lines 2-17). More techniques to reduce model error are discussed in Section 4.4.3 (lines 9-12,18-21). To improve performance, hybrid computation is applied as described in Section 4.4.4 and approximate algorithms are applied as shown in Section 4.4.5 (line 11).

Algorithm 5: Improved Algorithm for Computing Forward Reachable Set

Input: $S_{\mathfrak{L}}$: current reachable set represented by a projectagon
Input: Δd, Δt: bloat amount and step size of the previous step
Output: $S_{\mathfrak{L}}^{\Delta t}$: the forward reachable set after time step Δt

1 **begin**
2 **repeat**
3 update Δd and Δt;
4 **for** *each projection p* **do**
5 **for** *each edge e* **do**
6 $f_e = prism(e) \cap E_{\mathfrak{L}} \cap intervalC(e, S_{\mathfrak{L}}) \;||\; prism(e) \cap E_{\mathfrak{L}}$;
7 $f_e = (bloat(f_e, \Delta d) \cap E_{\mathfrak{L}}) \;||\; heighten(f_e, \Delta d)$;
8 $models_e = model\_create(bloat(f_e, \Delta d))$;
9 **for** *each model* $m_e \in models_e$ **do**
10 $f_e^{\Delta t} = lp\_forward(f_e, m_e, \Delta t)$;
11 $polys_e(i) = lp\_projA(f_e^{\Delta t}, \vec{x}_p, \vec{y}_p)$;
12 **end**
13 $poly_e = poly\_intersect(polys_e)$;
14 $\Delta d' = \max(ph\_bloatAmt(f_e, poly_e), \Delta d')$;
15 **end**
16 **end**
17 **until** $\Delta d' \leq \Delta d$;
18 **repeat**
19 $\Delta d = \Delta d'$;
20 repeat lines $8-14$;
21 **until** $||\Delta d - \Delta d'|| \leq \varepsilon$;
22 **for** *each plane* $l \in \mathfrak{L}$ **do**
23 $p_l^{\Delta t} = poly\_simplify(poly\_union_{e \in S_{\mathfrak{L}}}(poly^l))$;
24 **end**
25 $S_0^{\Delta t} = \bigcap_{l \in \mathfrak{L}} prism(p_l^{\Delta t})$; $E_0^{\Delta t} = \bigcap_{l \in \mathfrak{L}} prism(convexhull(p_l^{\Delta t}))$;
26 **repeat**
27 **for** *each plane* $l \in \mathfrak{L}$ **do**
28 $poly_l^i = lp\_project(E_{i-1}^{\Delta t}, \vec{x}_l, \vec{y}_l)$;
29 $p_l^i = poly\_simplify(poly\_intersect(p_l^{i-1}, poly_l^i))$;
30 **end**
31 $S_i^{\Delta t} = \bigcap_{l \in \mathfrak{L}} prism(p_l^i)$; $E_i^{\Delta t} = \bigcap_{l \in \mathfrak{L}} prism(convexhull(p_l^i))$;
32 **until** $||E_{i-1}^{\Delta t} - E_i^{\Delta t}|| \leq \varepsilon$;
33 $S_{\mathfrak{L}}^{\Delta t} = S_i^{\Delta t}$; $E_{\mathfrak{L}}^{\Delta t} = E_{i-1}^{\Delta t}$;
34 **end**

4.4.1 Reducing Projection Error

In Algorithm 4, each advanced face must be projected onto all planes in order to ensure soundness. This approach is very expensive with large projection error. Projecting an advanced face back to the plane for the original edge is usually much more accurate than projecting onto other planes. This is because the face has much tighter constraints of variables X_l and Y_l, *i.e.*, an edge, than other variables, and the linear differential inclusion model has a smaller error term for these two variables. Although the value of X_l and Y_l variables can be affected by other variables as the face may rotate during a step, the correlation is usually much smaller especially when the step size is very small. Therefore, it is attractive to only project an advanced face back onto its own plane.

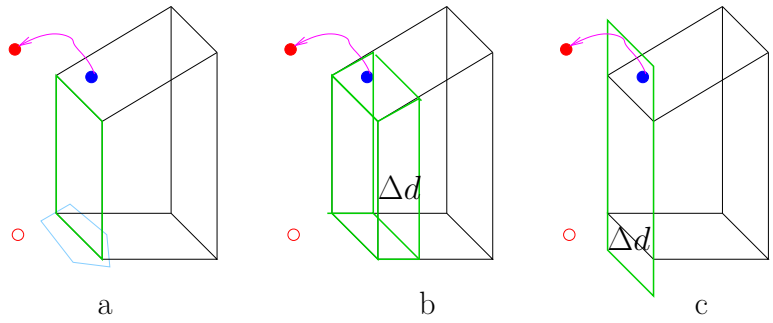


Figure 4.7: Projectagon Faces to be Advanced: a) Extreme trajectory from a face which does not correspond to an edge of the plane; b) Bloat the face inward; c) Increase the height of the face.

However, this method does not consider all possible trajectories as illustrated in Figure 4.7(a). To ensure soundness, there are two approaches as shown in Figure 4.7(b) and Figure 4.7(c) [3]. The first approach bloats a face inward by a distance Δd which captures all boundaries that can lead to extreme points of projected polygons, because any point in the projectagon cannot move more than Δd distance along any direction. The second approach increases the height of the face by Δd to block any extreme trajectory from other faces because the extreme trajectory must cross the heightened face. The bloated face or heightened face are used to create linear differential models and to be advanced in each step. Both meth-

[3]The correctness of our approach is proved in Appendix B.

ods significantly improve the performance. The second technique has relatively smaller error than the first one because the bloated face enlarges the region of X_l and Y_l variables directly whereas the heightened face only increases the range of other variables that have smaller impact on X_l and Y_l.

4.4.2 Guess-Verify Strategy

Choosing a good pair of step size Δt and bloat amount Δd is important to obtain good performance and small error. If the bloat amount is too small, COHO will take very small time steps resulting in long execution times and reachable regions that are overly conservative because of the error from the projection phase. Conversely, if the bloat amount is too large, then the non-linearity error (the u term in Equation 4.15), will be large, causing another kind of over approximation and small time steps. In Algorithm 4, the step size is computed using the ℓ_∞ norm of the derivatives. Therefore, it is usually very pessimistic and much smaller than what would actually be safe for the given bloat amount. When a face is advanced, the successor of the face at the end of a time step of Algorithm 4 often lies well inside the bloated face.

Noting the fact that the pair of step size and bloat amount are valid as long as the advanced face lies inside the bloat region used to create the LDI model[4]. A *guess-verify* strategy was developed which tries to guess a pair of step size Δt and bloat amount Δd based on the data from previous steps. At the end of each step, the method checks that the estimated bloat is sufficient for the estimated step size. If not, it updates the bloat amount and/or step size and repeats the computation. In addition to enabling larger time steps, the guess-verify strategy speeds up the computation of each step by eliminating the step-size calculation phase of Algorithm 4 (lines 7,10).

The verify step is based on computing the *real bloat amount* $\Delta d'$ and comparing it with the estimated bloat amount Δd. The basic operation of computing a forward reachable set is to move forward a face that corresponds to a projection edge e and

[4]Soundness requires that the reachable tube from the face throughout the time step $[0, \Delta t]$ must remain in the bloated region. In order to obtain a simple implementation, we only consider reachable sets at time 0 and Δt for efficiency in the implementation. There is a potential unsoundness if a trajectory goes outside the bloat region and re-enters the bloat by the end of the time step.However, this happens rarely because the step size is generally tiny and our inclusion dynamics are linear.

project back the advanced face onto its own plane, resulting in a projected polygon p. To check if the assumption that every point can move along any axis direction by at most Δd, the maximum moving distances along X_e and Y_e must be computed based on the edge e and the projected polygon p. This is computed by solving a linear program.

$$\begin{aligned} &\min \Delta d', \text{ s.t.} \\ Px &\leq b + |P| \cdot \Delta d' \\ Ax &\leq b, \end{aligned} \tag{4.31}$$

where $Px \leq q$ is the COHO constraint system that describes the face, and $Ax \leq b$ is the constraint from the convex polygon p. By assuming each point on the face can move by at most $\Delta d'$ along each axis direction, the region reachable during the time step $[0, \Delta t]$ is bounded by $Px \leq b + |P| \cdot \Delta d'$. Therefore, solving this linear program finds the minimum bloating amount that ensures the polygon p is contained in the bloated face.

4.4.3 Reducing Model Error

A large fraction of the approximation error of the reachability computation is from the linearization error during generating linear differential inclusion models, *i.e.*, the u term in Equation 4.15. Although the guess-verify strategy reduces the accumulated error by decreasing the number of steps, bloated faces or heightened faces make the error term larger. In the current research, I have devised, implemented, and evaluated several techniques to reduce the magnitude of the over-approximations used by COHO.

In Algorithm 4, all variables are bloated equally. However, in a dynamic system, it is common that some variables change much faster than others. For example, in digital circuits, a few signals will be in transition at any given time and the others will be relatively stable. This results in excessive bloating for the stable signals. To achieve an acceptable step size, the bloat for fast changing signals must be relatively large. When the same bloat is used for all variables, the bloat for slow changing signals is excessive, leading to much larger error terms in the differential inclusion than necessary. Likewise, when a signal is changing, it is generally either clearly rising or clearly falling. Thus, a large bloat is only needed in one direction,

allowing the total bloat for these variables to be reduced by nearly a factor of two. We implemented *asymmetric and anisotropic bloating*. Asymmetric bloating allows the positive and negative bloats for a variable to be different. Thus, bloating can adapt to the direction in which a signal is making a transition. Anisotropic bloating allows each variable to have its own bloat amount. Thus, bloating can adapt according to which variables are changing and which are stable. This approach allowed a significant increase (*e.g.,* 4x) in the typical step size. As an added benefit, the smaller total bloat reduced the error terms in the differential inclusion, allowing COHO to compute tighter bounds on the reachable regions.

At the end of each step, a *real bloat amount* $\Delta d'$, which is the smallest valid bloat for the step size, can be calculated as used in the guess-verify strategy. It is smaller than the bloat amount Δd hence it can be used to refine models. These refined models also make it easier to guess an accurate bloat amount for the next step and consequently reduce the number of guesses.

Another approach to obtain more accurate models is to use *multiple models* at the same time. The intersection of several linear differential inclusions provides a tighter bound of the non-linear dynamics. As a result, each face must be advanced and projected several times. The intersection of projected polygons from different models is used to generate a more accurate forward reachable set. Of course, this method also increases the total running time. It is appealing especially for circuit verification because it can prevent non-physical behaviors. For example, if the drain voltage is greater than the source voltage, the current of NMOS transistor is always positive, but negative current may be introduced during the modeling phase. An extra constant differential inclusion can be added to eliminate projectagon regions corresponding to these non-physical, negative currents.

4.4.4 Hybrid Computation

To make the algorithm robust, we applied APR numbers in the LP solver as described in Section 4.3.2. We also use APR in the geometric computations and the projection function. Although all rational numbers are rounded to floating point numbers at the end of each step, it is still very expensive to use APR numbers for all computations. In practice, APR numbers are only necessary for ill-conditioned

problems which happen rarely. A *hybrid computation* strategy is used to solve this problem by applying both interval computation and APR arithmetic. Most computations are performed using interval arithmetic which computes upper and lower bounds of the result and detects ill-conditioned problems. If the interval computation fails, the APR package is used to recomputed an exact solution. This hybrid computation method improves the performance significantly. Prior to using APR, COHO runs would often fail due to ill-conditioned computations, and these errors occurred in many places in the COHO code. The hybrid interval/APR approach provided a simple implementation that has eliminated these errors from COHO. Therefore, it has been applied to the LP solver, and the geometry package. The projection function is implemented entirely with APR as most of time are spent on solving LPs.

The method was generalized to support floating point numbers, interval numbers and APR numbers. The Simplex algorithm with floating point numbers can solve most of the LPs and find the exact optimal basis. To speed up the computation, the LP solver uses ordinary double-precision arithmetic for each pivot. It then verifies that each pivot succeeded in reducing the cost function first by using interval arithmetic, and in the infrequent event that this fails, COHO uses APR. If the pivot failed to reduce the cost, it repeats the pivot step with interval arithmetic or APR. Likewise, at the end of the algorithm, it tests the optimality of the solution by verifying that it is feasible in both the primal and dual LPs, again using interval arithmetic first and APR if the result from the interval calculation is inconclusive. In this way, we obtain the certainty of APR while performing nearly all calculations using ordinary, double-precision arithmetic. The most efficient algorithm can often be obtained by the strategy which uses floating point numbers for ordinary computation, uses interval numbers to validate the result and uses APR for ill-condition problems. Of course, other combinations of floating point, interval and APR numbers can also be applied.

4.4.5 Approximation Algorithms

Even with the hybrid computation method, solving LPs and projecting advanced faces is still one of the most expensive computations in COHO. To improve perfor-

mance further, we developed more efficient algorithms to compute approximated results.

Noting that most of the LPs to solve occur in the projection algorithm, the LP solver is improved by taking advantage of the special properties of these LPs. As shown in Section 4.3.2, when θ in Equation 4.28 and Equation 4.29 is increased to force a pivot to the next edge of a projected polygon, the standard form LP becomes infeasible. Traditional formulations of Simplex assume a feasible basis; thus, the original algorithm restarted the LP solver to establish feasibility for each edge of the projection of each face. However, only a single pivot is required to re-establish feasibility in the absence of degeneracies. Accordingly, we modified our LP solver to try each column of P^T to determine if its introduction into the optimal basis of the previous LP achieves optimality. This requires a single linear-system solve for each column tried which can be performed in $O(n)$ time due to the special structure of COHO's LPs. We found that this optimization works for about 80% of the projection polygon edges which resulted in a significant improvement in performance. The rather high failure rate is because the prisms whose intersection forms the projectagon are orthogonal to each other, leading to a higher rate of degeneracy than for typical LPs. A more efficient method was also developed to find the initial feasible basis for the Simplex algorithm. It tries to find constraints with only one non-zero coefficient for each variable, which is easy to find as all constraints of COHO LPs have only one or two non-zero coefficients, and constructs a feasible basis from these candidates.

The projection of an advanced face at the end of a time step can have clusters of very closely spaced vertices separated by much larger gaps. These clusters arise from near degeneracies in the COHO LPs. To avoid a rapid growth in the number of vertices in the projection polygon, COHO performs a simplification step where the projection polygon is replaced by an enclosing polygon of smaller degree. Consequently, every vertex but one in a cluster will be discarded by the simplification process, but the projection algorithm expended a significant amount of computation time to determine these vertices. In the new implementation as shown in Algorithm 6, we avoid this extra work by enforcing a lower bound on the change of θ at each step of the projection algorithm as shown in Algorithm 3. The approximate algorithm skips over vertices if the normals of the consecutive polygon edges

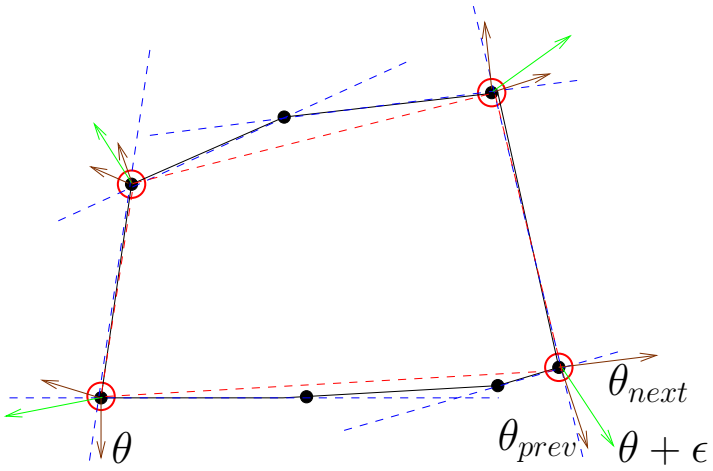


Figure 4.8: Approximated Projection Algorithm. The red polygon is an under-approximation and the blue polygon is an over-approximation of the projection.

are nearly parallel. Thus, the polygon obtained from the revised projection algorithm could be an under-approximation *e.g.,* the red polygon in Figure 4.8, which would violate the soundness requirement for COHO. Conversely, we can use each vertex from the projection algorithm to define a half plane, and construct the polygon defined by the intersection of these half-planes. The resulting polygon is an over-approximation, illustrated as the blue polygon in Figure 4.8. COHO computes both polygons. If their areas differ by more than a preset tolerance, COHO reverts to computing the exact projection polygon. Otherwise it uses the over-approximation.

4.5 Implementation

The COHO tool is implemented in MATLAB and JAVA with a layered architecture as shown in Figure 4.9. It consists of four layers. The top layer converts circuit verification problems into reachability analysis problems as described in Chapter 3. It constructs one or more hybrid automata based on the circuit structure described in MSPICE and extracts circuit dynamics in order to compute LDI models. The second layer, *i.e.,* the *Hybrid Automata* package, implements the reachability analysis algorithm described in Section 4.1. The projectagon representation described in

Algorithm 6: Approximated Projection Algorithm(lp_projA($PEx \leq q, \vec{x}, \vec{y}$))

Input: COHO advanced constraints: $PEx \leq q$
Input: projection plane $(\vec{x}, \vec{y})$
Input: error tolerance τ
Output: p: projection onto $(\vec{x}, \vec{y})$ plane

begin
  $\vec{x} = E^{-1}\vec{x}$; $\vec{y} = E^{-1}\vec{y}$;
  $\vec{y} = \vec{y} - \frac{\vec{x}\cdot\vec{y}}{||\vec{x}||}$; $\vec{x} = \frac{\vec{x}}{||\vec{x}||}$; $\vec{y} = \frac{\vec{y}}{||\vec{y}||}$;
  for $\theta = 0, \theta \leq 2\pi$ **do**
    $d = \vec{x}\cos(\theta) + \vec{y}\sin(\theta)$; lp = lp_standard($P, q, d$);
    [B, pt, v] = lp_solve(lp);
    $p_l = p_l + project(\mathsf{pt}, \vec{x}, \vec{y})$;
    $planes = planes + plane\_create(\theta, \mathsf{v})$;
    $\theta_{opt} = \theta + \varepsilon(\tau)$; $d = \vec{x}\cos(\theta_{opt}) + \vec{y}\sin(\theta_{opt})$; lp = lp_standard($P, q, d$);
    [B, pt, v] = lp_solve(lp);
    $p_l = p_l + project(\mathsf{pt}, \vec{x}, \vec{y})$;
    $\zeta = P_{\mathsf{B}}^{-T}\vec{x}$; $\eta = P_{\mathsf{B}}^{-T}\vec{y}$;
    $\theta_{prev} = \max_{i\in[1,n]}(\arg_\theta \min(\zeta_i \cos\hat{\theta} + \eta_i \sin\hat{\theta} = 0 \wedge \hat{\theta} < \theta_{opt}))$;
    $\theta_{next} = \min_{i\in[1,n]}(\arg_\theta \min(\zeta_i \cos\hat{\theta} + \eta_i \sin\hat{\theta} = 0 \wedge \hat{\theta} > \theta_{opt}))$;
    if $\theta_{prev} > \theta$ **then**
      $planes = planes + plane\_create(\theta_{prev}, \mathsf{v})$;
    end
    $\theta = \theta_{next}$;
  end
  $p_l = convexhull(p_l)$; $p_u = plane\_intersect(planes)$;
  if $\frac{area(p_u) - area(p_l)}{area(p_l)} \leq \tau$ **then**
    $p = E^{-1}p_u$;
  else
    $p = lp\_project(PEx \leq q, \vec{x}, \vec{y})$;
  end
end

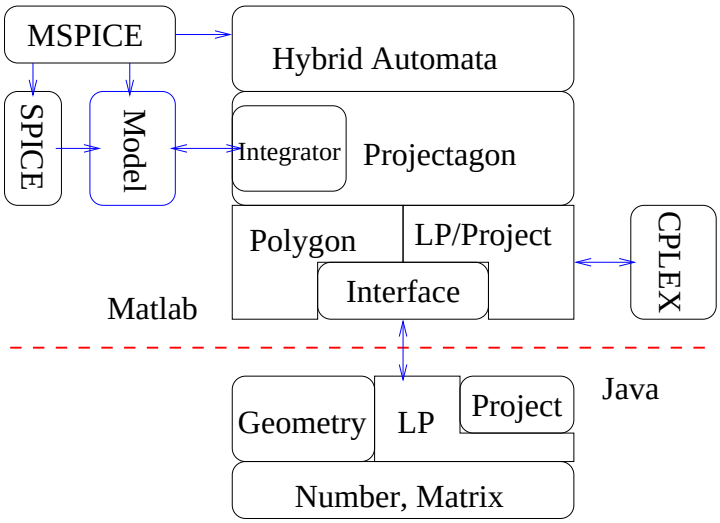


Figure 4.9: Architecture of COHO

Section 4.2 is implemented in the third layer. An *Integrator* package is also implemented in this layer to compute forward reachable projectagons as described in Section 4.3 and Section 4.4. The bottom layer provides basic functions including a COHO LP solver, projection algorithms and a geometric engine. COHO has two main components: a component written MATLAB and a component written in JAVA. The MATLAB part provides the interface for higher layers and also implements simple functions which do not require exact solution. The JAVA part provides robust LP solvers, projection functions and geometry computation functions based on the *Number* and *Matrix* packages. The MATLAB and JAVA components communicate through a pair of pipes created by a simple C program. The layer based architecture has the benefit that it is very easy to implement new algorithms to replace old ones without affecting other layers. For example, instead of our JAVA based LP solver, commercial LP solvers, *e.g.*, CPLEX [5], can also be adopted easily. The separation of MATLAB and JAVA components also provides a convenient place to create a log-file that allows COHO runs to be restarted from just before an error. This has been especially helpful for debugging COHO when implementing new algorithms.

For complex hybrid systems, reachability computation is usually very expen-

sive and the approximation error is large. Therefore, it is crucial to achieve a good trade-off between performance and accuracy in order to successfully verify a system. COHO offers several implementations of every function, optimized for performance or precision. First, it supports two kinds of representation methods: *non-convex projectagon* and *convex projectagon*. The general projectagon representation has the smallest representation error especially for nonlinear systems. However, the convex projectagon method is much faster because many operations can be implement more efficiently, such as the union of convex polygons. Therefore, it is often used in the first time of reachability computation to obtain an estimated result quickly. The approximation error is usually acceptable, otherwise, the computation is refined using the non-convex projectagons. Second, COHO supports three methods for finding a bloat amount and step size pair: *fixed bloat amount*, *fixed time step*, and *guess-verify*. The first and second options use a user provided bloat amount or step size to compute a valid step size or bloat amount, and the guess-verify method is described in Section 4.4.2. Third, it supports several methods for finding projectagon faces which are used to build LDI models and to be advanced. COHO can either advance each face and project back onto all planes represented in Section 4.2.3, or advance bloated faces or heightened face and project onto its own plane only as described in Section 4.4.1. In addition to these methods, it also supports a much faster technique which creates only one LDI model for a projectagon and advances the whole projectagon rather than working on each face. However, this method has larger approximation error and does not work for non-convex projectagons. Another option is to advance each projectagon face and only project back onto its own plane. This method can generate the most accurate result. Although it is not sound in theory, it works well in practice because approximation errors from other steps, such as modeling and projecting steps, make the result conservative enough to contain all possible trajectories. Finally, there are some other parameters provided to control the computation precision, such as tolerance values used for simplifying a polygon or computing an approximated projected polygon. These options make it possible to use different strategies in states of a hybrid automaton. For example, we can optimize for running time in states where the system converges quickly and optimize for accuracy otherwise. Furthermore, it makes COHO more robust by switching to another set

of options and repeating the computation when an error or exception is found in a step.

4.6 Summary and Discussion

We have devised techniques for reachability analysis of hybrid systems and implemented them in the COHO tool. It supports moderate-dimensional systems based on the projectagon representation which reduces the number of dimensions by projecting high-dimensional polyhedra onto two-dimensional planes. It also supports nonlinear systems by approximating continuous dynamics by linear differential inclusions. The COHO tool is robust and has been used to verify properties of several circuits as will be described in the next chapter. The main improvements that we developed to provide this robustness were the use of a combination of interval arithmetic and arbitrary precision rational arithmetic throughout the linear program solver and geometric engine. We also developed new techniques to remove infeasible regions from projection polygons, which is necessary for the robustness of the implementation. Reachable sets computed by COHO are accurate for several reasons. First, non-convexpolyhedra can be represented by projectagons directly, and all faces can be computed accurately by the interval closure technique. Second, all computations are performed on projectagon faces rather the whole projectagon. Therefore, small approximation error is achieved during the modeling and projecting steps. Furthermore, several techniques have been developed to reduce modeling error including model refinement, multiple models, and a guess-verify strategy. COHO is also efficient because it employs many techniques including hybrid computation, guess-verify, and an approximated LP solver as well as the projection function. COHO also supports trade-offs between performance and accuracy and handles exceptions automatically.

There are several related techniques and tools, including D/DT [48], CHECKMATE [40], and PHAVER [70]. Compared with other representation techniques, the projectagon representation is more accurate because it supports non-convex regions and works on individual faces. Similarly, orthogonal polyhedra [33] works on each face in the face lifting technique and represents non-convex regions by the union of fixed-grid hyper-rectangles. It can be viewed that intervals or hyper-

rectangles are the results of projecting polyhedra onto one dimensional lines and projectagons are the results of projecting polyhedra onto two-dimensional planes. The projecting idea can be generalized to three or higher dimensional subspaces. It might be helpful to represent circuit states more precisely because the dynamics of one circuit node usually depends on more than two variables. However, manipulating moderate-dimensional, non-convex polyhedra is much more expensive, therefore, this technique has not been studied. Hybridization strategy has been widely used to solve nonlinear dynamics. We adopted this approach and optimized it to reduce approximation errors by computing LDI models for each face. This on-the-fly approach is more accurate but also more expensive than computing only one model for a fixed-grid region as used in other tools, *e.g.*, D/DT [24]. The LDI model is much more precise than the constant differential inclusion model used in many tools, including LEMA [148] and PHAVER [70]. On the other hand, solving constant differential inclusion is much easier. Furthermore, we found it is helpful to use both linear DI models and constant DI models to reduce modeling error. APR numbers are also used in PHAVER [70] for exact computation. However, we have not found that the hybrid computation technique is applied in PHAVER or other similar tools.

There are some research directions to improve COHO. First, the normal of each advanced face can be estimated based on the maximum principle in Section 4.3.1. It can help to simplify the new projectagon at the end of each step. As a result, the number of projectagon faces is roughly constant. A similar approach is used by template polyhedra [181] in order to control the space complexity. However, the template is fixed in the template polyhedra representation whereas it is updated automatically in the projectagon representation. Furthermore, parallel computation is a promising technique to speedup COHO significantly as there are many parallelisms in the projectagon based computation.

5

Examples

With methods described in Chapter 3 and Chapter 4, we can construct mathematical models for circuits and compute reachable regions using the tool COHO. This chapter applies these techniques to the circuits presented in Section 3.2 to demonstrate correctness and effectiveness of our methods and tools. We first present a general framework for defining verification problems, computing reachable regions and checking properties in Section 5.1. Then we show experimental examples with greater details and verification results in the following sections.

5.1 Verification of AMS Circuits

To formally verify a property of a given circuit, we propose a framework which consists of three steps.

1. Simulate the circuit and the property
2. Compute reachable regions of the circuit
3. Check the property based on the result of reachability analysis.

The remainder of this section describes each step.

5.1.1 Simulation and Verification

Generally, we simulate the circuit to be verified before attempting verification. This has several advantages. First, simulation can find obvious errors with less

effort and computation than formal verification. Second, the simulation result can be used to approximate the initial condition of formal verification. For example, we use simulation traces to estimate Brockett's annuli for input signals of a circuit such that its outputs satisfy properties of interest. During the process of verifying the circuit, we often compare simulation traces and reachable regions to find the most critical computation steps such that we can obtain a good trade-off between performance and approximation error during reachability analysis.

We have developed a MATLAB package, MSPICE, to describe the netlist of a circuit and simulate the circuit. While the simulation speed is an order-of-magnitude slower than a dedicated simulator such as HSPICE, MSPICE gives the user much greater flexibility and access to the numerical computations of the simulation. This allows, for example, inputs to be generated as random trajectories that satisfy their given Brockett's annuli. With this framework, we use the same models (see Section 3.3.2) for simulation and verification. This makes it possible to compare the results of MSPICE and COHO, where the simulation result is an under-approximation of possible circuit states and the reachable region computed by COHO is an over-approximation. This guides us when choosing which methods to use to reduce the over-approximations computed by COHO. Furthermore, when developing new reachability techniques in COHO, the comparison with simulation results is helpful for identifying and correcting errors in the algorithms or their implementations. However, the integrator of MSPICE requires that the functions to be integrated be continuous. To make the models in Section 3.3.2 continuous, we smooth out interpolated values by a weighted cosine window. For example, let g_0, $g_1, \ldots$ be a set of continuous functions where g_i is the one for $i-1 \leq x \leq i+1$, the interpolated function $f(x)$ is

$$\begin{array}{rcl} f(x) & \approx & w_m(x-\lfloor x \rfloor) g_{\lfloor x \rfloor}(x-\lfloor x \rfloor) + w_m(x-\lceil x \rceil) g_{\lceil x \rceil}(x-\lceil x \rceil) \\ w_m(x) & = & \frac{1}{2} + \frac{1}{16}\left(9\cos(\pi x) - \cos(3\pi x)\right). \end{array} \tag{5.1}$$

This 3-term cosine window method makes $f(x)$ a C^3 function because $\frac{d^k}{dx^k} w(1) = 0, \forall k \in [0,3]$. Higher orders of smoothness can be obtained by applying more cosine terms in the cosine window function.

5.1.2 Reachability Computations

Reachability computations are often quite slow, *e.g.*, it may take several days to complete the computation for a 6-dimensional system. Therefore, it is crucial to partition the reachability computation into several phases and perform the computations in parallel. To make the computation in each phase independent of the other phases, we apply an *assume-guarantee* strategy. For each phase, we *assume* that the circuit state is bounded by a region R_i and compute forward reachable regions from R_i. At the end, we *show* that all assumptions are correct based on reachable regions computed in all phases to establish an invariant set. Usually, we partition the reachability computation based on input transitions as shown in Section 3.6.2. As described in Section 3.6.2, the reachability computation for a circuit with one input can be partitioned into two phases which correspond to the rising and falling stages of the input signal. For a circuit with two inputs, the reachability computation as shown in Figure 3.24 is usually partitioned into four phases: the first phase starts from the state $B_{<1,1>}$ and ends in the state $B_{<3,3>}$; the second phase is from the state $B_{<1,3>}$ to the state $B_{<3,1>}$; the third phase is from the state $B_{<3,1>}$ to the state $B_{<1,3>}$; and the last phase is from the state $B_{<3,3>}$ to the state $B_{<1,1>}$. This strategy enables us to complete the verification, debug the model, and address issues of over approximation in a reasonably timely manner. It also makes it easier to find phases with large approximation errors and consequently optimize COHO for accuracy to avoid false-negatives. Similarly, we can speed up reachability computation for phases in which circuit states converge quickly.

We can further partition one state of the hybrid automaton used in COHO into several states by *slicing* signals. For example, instead of computing the reachable region for a signal x in the region of $[0,1]$ in one state, we can slice the signal by the face where $x = 0.5$ and then solve reachability problems for states where x is in the range of $[0,0.5]$ and $[0.5,1.0]$. The slicing strategy avoids computing LDI models (see Equation 4.15) for large regions and thus reduces approximation error. It could also improve the performance of reachability computation which depends on the size of the initial region. By increasing the number of states of the hybrid automaton used in COHO, slicing enables more parallelism. However, it also introduces extra computation and approximation, *e.g.*, intersection of reachable regions

and slicing faces. From the experimental results, we found it is a good choice to slice a signal in 0.2 volt intervals for several designs in a 180nm, 1.8 volt CMOS process.

5.1.3 Checking Properties

Based on the reachable regions computed above, we can check properties of interest. For example, we can compute the Brockett's annulus for a given signal based on the reachable region and its ODE model as shown in Equation 3.2. Finding an invariant set of the reachable region is very helpful to show safety properties and liveness properties. In the current implementation, most properties specified by the logic in Section 3.4 are checked by manually inspecting the reachable regions computed by COHO. We will talk about details for each verification example in the following sections.

By default, we use circuit models for the TSMC 1.8V 180nm bulk CMOS process. For a circuit, the Brockett's annulus for each input signals s is specified by $\mathscr{B}_{(0,0.15,1.65,2.0,1.5e10,2e10,1e-9)}$. However, when computing the linear differential inclusion model as described in Section 3.6.1, we increase the maximum low value from 0.15V to 0.2V and lower the minimum high value from 1.65V to 1.6V to ensure the derivative is strictly positive in region 2 and negative in region 4. For example, we compute a linearized model in Brockett region B_1 for $0 \leq s \leq 0.2$ as if V_{0h} were 0.2V, and use the original Brockett's annulus for $0.2 \leq s \leq 1.6$ which ensures that COHO sees a clearly positive value for $\dot{s}$ in region B_2. It is similar for regions B_3 and B_4. This method over-approximates the input and is thus sound.

It is possible that COHO is unable to verify that the outputs of some circuits satisfy a desired Brockett's annulus due to COHO's approximation error. Buffering the output with an inverter provide a very simple model without changing the circuit behavior. An inverter functions as a Brockett's annulus transformer. The inverter can take a slow input transition and turn it into a faster output transition, because of the inverter's gain. Conversely, it can take a very fast input transition and produce a slower output transition because the inverter has a maximum slew rate. The filtering effect removes reachable regions that COHO computed due to approximation errors. This trick usually does not increase the number of dimen-

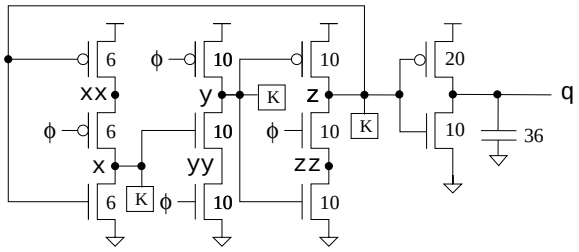


Figure 5.1: Verified Toggle Circuit

sions of the reachability computations, because the decomposition strategy can be applied to partition the circuit and perform two separate reachability computations for the original circuit and the inverter.

5.2 The Yuan-Svensson Toggle

Figure 5.1 shows the toggle circuit that we verified. Transistors are labeled with their shape factors and the capacitor on the *q* output represents a load equivalent to the gate capacitance of transistors with a total shape factor of 36; this is the load that the toggle places on its clock input. We use this load to verify that the output of one toggle can drive the clock input of another to implement a ripple counter.

The verification that we present resembles an earlier verification result by Greenstreet [83]. There are several significant differences between our approach and the earlier work. Most significantly, Greenstreet ignored internal nodes, *i.e.,* xx, yy, and zz, and reduced the system further to three dimensions by changing variables. We include these internal nodes in our model, which results in a seven-dimensional state space. Second, we model the drain-source currents of the transistors based on tabulated data obtained from HSPICE as described in Section 3.3.2, and thus our results are based on the BSIM-3 models implemented by HSPICE for the TSMC 180nm process. In contrast, [83] used a simple, first-order, long-channel MOSFET model that neglected leakage currents and other important properties of transistors in a deep-submicron technology. Using a realistic model forced us to address

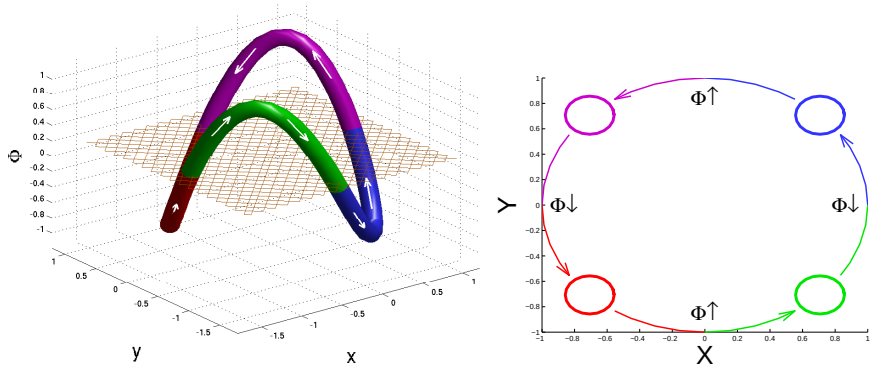


Figure 5.2: Behavior of a Toggle

other real-world issues, most notable of which was the leakage currents of the transistors. Thus, we added "keepers" to nodes x, y and z (illustrated by "k" in Figure 5.1). While such practicalities can seem like a nuisance from a formal verification perspective, they show that our approach is solidly connected to the issues that challenge circuit designers for deep submicron processes. Like COHO, [83] used projection polygons to represent reachable regions. However, these polygons were restricted to have axis-parallel edges. In contrast, we use projectagons to represent reachable regions and the algorithms from Chapter 4 that are more efficient and more accurate for reachability analysis. We also verified that the output signal satisfies the same Brockett's annulus specification as we use to specify the input clock, ϕ.

We specify the behavior of the toggle circuit using the safety properties given in Figure 3.20. We use COHO to find an invariant subset of $\mathbb{R}^d$ such that all trajectories in this set have a period twice that of the clock signal. This notion can be visualized using a Poincaré section [166] as illustrated in Figure 5.2. Let ϕ be the continuous signal corresponding to Φ, and let c be some constant with $V_{0h} < c < V_{1l}$. Consider the intersection of the invariant set with the $\phi = c$ hyperplane. These intersections form a Poincaré map [166]. We verify that these intersections form four disjoint regions (two for rising ϕ crossing c, and two falling crossings), and all trajectories must visit these four regions in the same order. Thus, the $\phi = c$ plane

partitions the invariant set of the continuous model into four disjoint regions that map to the four discrete states of the discrete model.

5.2.1 The Reachability Computation

Our specification for the toggle requires it to have an invariant set that has twice the period of the input clock, ϕ. Accordingly, our reachability calculation is carried out for two periods of ϕ. Applying the assume-guarantee strategy, we break each of these periods into two phases: one for the rising transition of ϕ and the time that ϕ is high; and the other for the falling transition and the low time. We estimate a bounding hyper-rectangle for the end of each phase based on the simulation results. With these estimates, we divide the task of verifying the toggle into four separate proof obligations where each obligation is of the form:

> **Assume** that the circuit state is in hyper-rectangle Y_i at the end of phase i.
> **Show** that the circuit state will be in hyper-rectangle Y_{i+1} at the end of phase $i+1$.

Let

$$\mathscr{B}_{toggle} \quad = \quad \mathscr{B}_{(0,0.15,1.65,2.0,2e10,3e10,1e-9)}, \tag{5.2}$$

and set the Brockett's annulus of the clock signal to $\mathscr{B}_{toggle}$. We start each phase with the projectagon for the starting hyper-rectangle and note the bounding hyper-rectangle of the projectagon for the reachable region at the end of each phase as shown in Table 5.1. Table 5.1 also lists the projection polygons that we used for each phase. These were chosen with two considerations. First, we chose projections that correspond to logical dependencies between changing signals. Thus, in the first phase when z changes, we include z vs. zz and z vs. x (because the falling edge of z enables a rising edge of x). Second, we included at least one polygon for each variable to bound the resulting projectagon in all dimensions.

The linear model for $\dot{\phi}$ has large errors if the interval for ϕ is too large. Thus, we "sliced" the space into regions corresponding to 0.1 volt wide intervals for ϕ. It is simple to show that the circuit model has an invariant that all node voltages are between 0 volts (i.e. ground) and 1.8 volts (i.e. V_{dd}) and that $xx \geq x$, $yy \leq y$

Start and end hyper-rectangle for each phase							
Phase	ϕ	x	y	z	xx	yy	zz
1, start	0.2	$[0.000, 0.100]$	$[1.700, 1.800]$	$[1.700, 1.800]$	$[0.000, 1.0]$	$[0.000, 0.100]$	$[0.000, 0.100]$
1, end	1.6	$[0.000, 0.002]$	$[1.790, 1.800]$	$[0.000, 0.014]$	$[1.788, 1.8]$	$[0.000, 0.004]$	$[0.000, 0.001]$
2, start	1.6	$[0.000, 0.100]$	$[1.700, 1.800]$	$[0.000, 0.100]$	$[1.700, 1.8]$	$[0.000, 0.100]$	$[0.000, 0.100]$
2, end	0.2	$[1.795, 1.800]$	$[1.758, 1.800]$	$[0.000, 0.043]$	$[1.795, 1.8]$	$[1.152, 1.736]$	$[0.000, 0.003]$
3, start	0.2	$[1.750, 1.800]$	$[1.750, 1.800]$	$[0.000, 0.050]$	$[1.750, 1.8]$	$[0.800, 1.800]$	$[0.000, 0.040]$
3, end	1.6	$[0.000, 0.001]$	$[0.000, 0.005]$	$[1.703, 1.800]$	$[1.785, 1.8]$	$[0.000, 0.002]$	$[0.843, 1.740]$
4, start	1.6	$[0.000, 0.100]$	$[0.000, 0.100]$	$[1.700, 1.800]$	$[1.700, 1.8]$	$[0.000, 0.100]$	$[0.800, 1.800]$
4, end	0.2	$[0.000, 0.100]$	$[1.700, 1.800]$	$[1.700, 1.800]$	$[0.000, 1.0]$	$[0.000, 0.100]$	$[0.000, 0.100]$

Projection polygons for each phase	
Phase	Polygons
1	x vs. xx, x vs. z, z vs. zz, z vs. xx, ϕ vs. y, ϕ vs. yy
2	x vs. xx, x vs. y, x vs. yy, y vs. yy, ϕ vs. z, ϕ vs. zz
3	x vs. xx, x vs. y, x vs. yy, y vs. yy, y vs. z, y vs. zz, z vs. zz, z vs. z, z vs. xx, ϕ vs. z
4	x vs. xx, y vs. yy, y vs. z, y vs. zz, z vs. zz, ϕ vs. z

Table 5.1: Reachability Summary of Toggle Verification

and $zz \leq z$. We added these extra invariants to COHO to tighten the bounds COHO computes. Phase 3 was the most challenging phase to verify. In this phase, ϕ goes from low-to-high, and all three of x, y, and z change their values, in the order $y\downarrow \rightarrow z\uparrow \rightarrow x\downarrow$. The greatest challenge arose because z can start its rising transition while y is still falling. As seen in Table 5.1, we used ten projection polygons for this phase instead of the six that were used in the other phases to improve accuracy. We found that once ϕ was high (*i.e.*, greater than 1.6 volts), it was helpful to slice the value of y. We used 0.1 volt wide slices for y as it fell from 1.3 volts to 0.1 volts. We sliced z in the same manner but found that it was unnecessary to slice x. Furthermore, the transistor model from Section 3.3 can produce large error bounds that include currents that flow against the drain-to-source voltage. These non-physical behaviours allowed by the model caused COHO to fail to verify the toggle. We solved the negative current problem by adding a transistor model that simply determines the minimum and maximum drain-to-source current (*i.e.*, a constant differential inclusion) for the region around the current face. While this model has a large error-term, it never predicts a current of the wrong sign. The two models produce two different over approximations of the reachable region. At the end of each time step, we compute the intersection of the two projectagons to obtain a tighter bound on the reachable space than either alone. More details are described in [210, 211].

However, we found that the values of all nodes x, y, and z resulted in large intervals during the reachability computation of the toggle circuit without the "keepers" as shown in Figure 5.1. This is caused by leakage currents of transistors. For example, when the input clock ϕ is low, x is low and y is high, node z is floating because both its upper P-channel and lower N-channel transistors are not conducting. The leakage currents of these N-channel and P-channel transistors allow the value of $\dot{z}$ to be a tiny value of arbitrary sign. Because ϕ can be low for an arbitrarily long time, COHO correctly finds that the possible values for z diverge to include any value between the power supply voltage and ground. This failure shows that leakage currents must be considered in deep sub-micron designs even for "simple" circuits such as the toggle circuit. Therefore, we added small "keeper" circuits to nodes x, y and z during the verification.

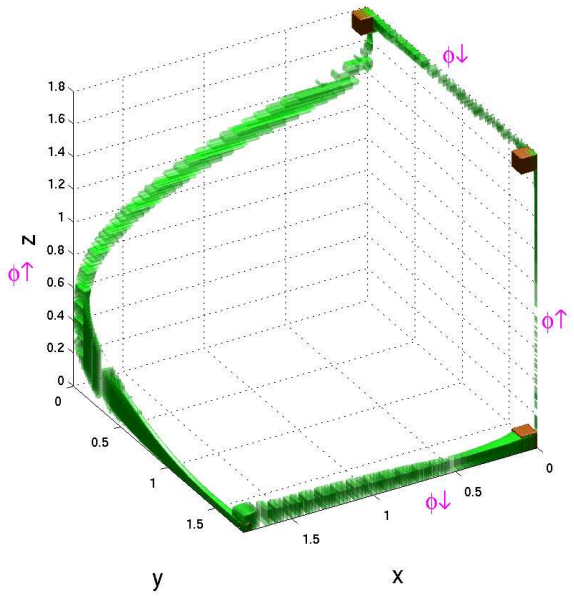


Figure 5.3: The Invariant Set of Toggle Circuit

With these techniques, we computed the reachable regions for each phase. The reachable region is six-dimensional and its projection onto x, y and z space is shown in Figure 5.3. Table 5.1 also lists the ending hyper-rectangle of all phases. Note that the starting hyper-rectangle for each phase contains the ending hyper-rectangle of the previous phase, and the starting hyper-rectangle for phase 1 contains the ending hyper-rectangle for phase 4. Thus, we have established an invariant set. Furthermore, the hyper-rectangles for the four phases are pairwise disjoint. Thus, this invariant set has a period of two with respect to the clock input ϕ.

5.2.2 Verifying the Output Brockett's Annulus

Thus far, we have ignored the q output of the toggle in our analysis – we simply included a load on z equal to the gate capacitance of transistors that drive q. We verified the operation of the inverter separately. To do so, we first constructed the Brockett's annulus for the z output.

At each time step of the verification described above, we determined the reachable combinations of z and $\dot{z}$. We note that $\dot{z}$ is negative monotonic in z and positive monotonic in zz. Thus, the extremal values of $\dot{z}$ *vs*. z occur on the boundary of the z vs. zz projection. For each edge of the z vs. zz projection, COHO computes the linearized circuit model and uses this model to find the reachable combinations of z and $\dot{z}$. From these, we construct a Brockett's annulus that is satisfied by z and $\dot{z}$. Figure 5.4 shows the result; z does not satisfies the same Brockett's annulus as we used to specify ϕ.

We then perform a separate reachability analysis for the output inverter. The input to this circuit is modeled by the Brockett's annulus for z as computed by COHO above. We then use COHO to compute the reachable space for z and q, and use the method above to compute the reachable region for $\dot{q}$ *vs*. q as described above for $\dot{z}$ and z. Figure 5.5 shows the result; q clearly satisfies the constraints that we used for ϕ. Thus, these toggles can be composed to form an arbitrarily large ripple-counter as desired.

5.3 A Flip-Flop Circuit

Figure 5.6 gives the latch circuit that we have verified. Transistors as well as inverters are labeled with their shape factors. The capacitor on output $\overline{q}$ represents a load equivalent to the load of input d. Therefore, we can use $\overline{q}$ as the input of another latch circuit to compose a master-slave flip-flop as shown in Figure 3.6. We use a large inverter (8:4) to generate the $\overline{\phi}$ signal.

Unlike the toggle circuit, the latch circuit has two inputs: the clock signal ϕ and the data input d. To model input transitions of two inputs, we employ the method presented in Section 3.6.2. The input specification requires that d must be stable when ϕ is falling. Therefore, states $B_{<4,2>}$ and $B_{<4,4>}$ are removed from the reachability computation. Noting that the reachable regions also depend on the initial states of internal nodes, two reachability problems with the same input signal and clock are solved for each of the two different latch states. Therefore, there are six independent phases to compute the reachable regions using the assume-guarantee strategy:

1: From $B_{<1,1>}$ to $B_{<3,3>}$ with $i_1 = low$.

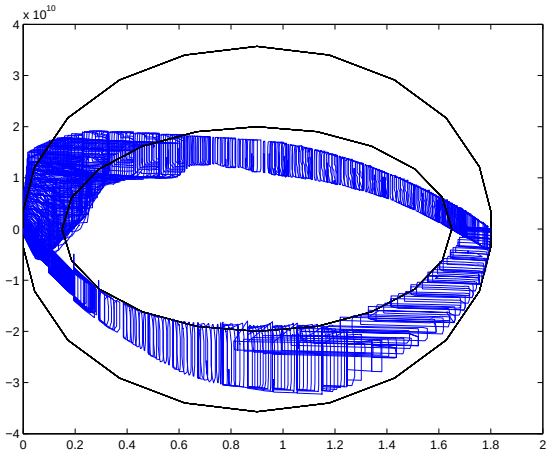


Figure 5.4: Brockett's Annulus of z. The blue polygons show the computed Brockett's annulus of signal z. The black lines show the polygonal approximation of the Brockett's annulus of ϕ.

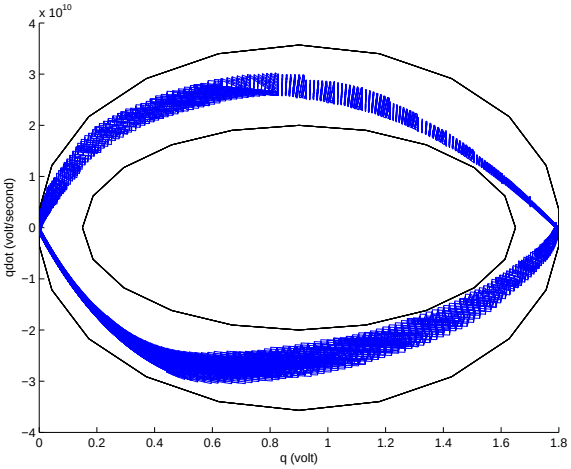


Figure 5.5: Brockett's Annulus of q

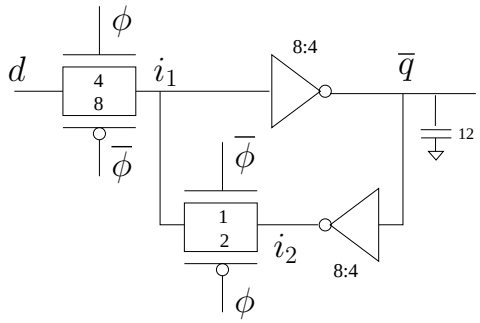


Figure 5.6: Verified Latch Circuit

2: From $B_{<1,1>}$ to $B_{<3,3>}$ with $i_1 = high$.
3: From $B_{<1,3>}$ to $B_{<3,1>}$ with $i_1 = low$.
4: From $B_{<1,3>}$ to $B_{<3,1>}$ with $i_1 = high$.
5: From $B_{<3,1>}$ to $B_{<1,3>}$ with $i_1 = low$.
6: From $B_{<3,3>}$ to $B_{<1,1>}$ with $i_1 = high$.

Table 5.2 lists the projection polygons used for reachability computation. It also provides the initial region of each phase. After the computation of all phases, we combine the results to check our initial estimations. For example, to check the initial region of phase 1, we compute the union of the ending hyper-rectangles of phases 3 and 6 because i_1 is low in these two phases. It is similar for other phases.

Figure 5.7 shows the value of $\overline{q}$ with respect to the Brockett's annulus of clock ϕ. This plot shows that $\overline{q}$ is clearly low or high when ϕ is in region 1. This proves that if input d is stable when clock ϕ is falling, then output $\overline{q}$ is also stable when ϕ is clearly low. Furthermore, it shows that the minimum duration time t_l and t_h can be as low as $0.27ns$ based on the reachable regions. Therefore, the highest frequency is about 1.3GHz considering the rising/falling time as defined by the input specification.

We also verified several properties of the flip-flop circuit as shown in Figure 3.6. Based on the properties of the latch circuit, it is obvious that the output $\overline{q}$ of the master latch of the flip-flop is stable when ϕ is rising. That is, the input $\overline{q}$ of the slave latch does not change when $\overline{\phi}$ is rising. Therefore, we can conclude that the output q of the slave latch is stable when $\overline{\phi}$ is low by applying the property of the

Estimate initial hyper-rectangle and result for each phase						
Phase	ϕ	d	$\overline{\phi}$	i_1	$\overline{q}$	i_2
1, assume	$[0.000, 0.200]$	$[0.000, 0.200]$	$[1.700, 1.800]$	$[0.000, 0.100]$	$[1.700, 1.800]$	$[0.000, 0.100]$
1, result	$[0.000, 0.200]$	$[0.000, 0.200]$	$[1.799, 1.800]$	$[0.000, 0.002]$	$[1.799, 1.800]$	$[0.000, 0.001]$
2, assume	$[0.000, 0.200]$	$[0.000, 0.200]$	$[1.700, 1.800]$	$[1.700, 1.800]$	$[0.000, 0.100]$	$[1.700, 1.800]$
2, result	$[0.000, 0.200]$	$[0.000, 0.200]$	$[1.799, 1.800]$	$[1.791, 1.800]$	$[0.000, 0.001]$	$[1.796, 1.800]$
3, assume	$[0.000, 0.200]$	$[1.600, 1.800]$	$[1.700, 1.800]$	$[0.000, 0.100]$	$[1.700, 1.800]$	$[0.000, 0.100]$
3, result	$[0.000, 0.200]$	$[1.600, 1.800]$	$[1.799, 1.800]$	$[0.000, 0.003]$	$[1.799, 1.800]$	$[0.000, 0.001]$
4, assume	$[0.000, 0.200]$	$[1.600, 1.800]$	$[1.700, 1.800]$	$[1.700, 1.800]$	$[0.000, 0.100]$	$[1.700, 1.800]$
4, result	$[0.000, 0.200]$	$[1.600, 1.800]$	$[1.799, 1.800]$	$[1.793, 1.800]$	$[0.000, 0.001]$	$[1.796, 1.800]$
5, assume	$[1.600, 1.800]$	$[0.000, 0.200]$	$[0.000, 0.100]$	$[0.000, 0.300]$	$[1.700, 1.800]$	$[0.000, 0.100]$
5, result	$[1.600, 1.800]$	$[0.000, 0.200]$	$[0.000, 0.001]$	$[0.000, 0.256]$	$[1.798, 1.800]$	$[0.000, 0.001]$
6, assume	$[1.600, 1.800]$	$[1.600, 1.800]$	$[0.000, 0.100]$	$[1.500, 1.800]$	$[0.000, 0.100]$	$[1.700, 1.800]$
6, result	$[1.600, 1.800]$	$[1.600, 1.800]$	$[0.000, 0.001]$	$[1.536, 1.800]$	$[0.000, 0.001]$	$[1.799, 1.800]$

Projection polygons for each phase
d vs. i_1, ϕ vs. i_1, $\overline{\phi}$ vs. i_1, i_1 vs. i_2, ϕ vs. i_2, $\overline{\phi}$ vs. i_2, ϕ vs. $\overline{\phi}$, i_1 vs. $\overline{q}$, $\overline{q}$ vs. i_2

Table 5.2: Reachability Summary of Latch Verification

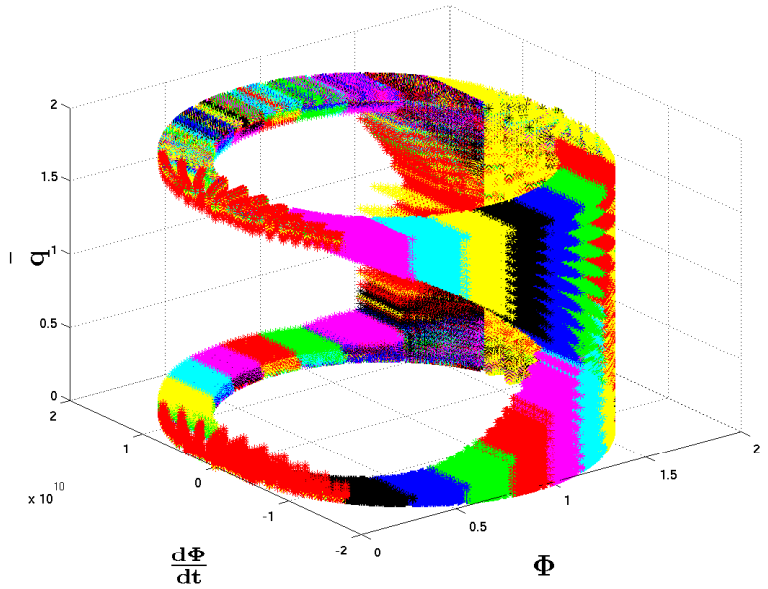


Figure 5.7: The Output Specification of Latch Circuit. This plot shows all reachable sets computed by COHO. Reachable sets are projected onto variables ϕ,$\dot{\phi}$ and $\overline{q}$. Results from different reachability computations are shown in different colors.

latch circuit again. It is proved that the output q of the flip-flop circuit is stable when the clock ϕ is high. In fact, $\overline{q}$ usually becomes stable before the next rising edge of ϕ as shown in Figure 5.8. As measured, the delay from the time when ϕ starts to fall to the time when q is stable is about $200ps$, which is an upper bound of the *clock-to-q* delay of the flip flop.

5.4 An Arbiter Circuit

Figure 5.9 shows the arbiter circuit that we have verified. Kurshan and McMillan [133] studied a similar arbiter circuit from [183] as their main example in proposing a way to verify digital circuits modeled by differential equations. Their arbiter is the nMOS counterpart of our CMOS design illustrated in Figure 5.9. They formulated the verification problem in terms of language containment. To model the continuous behavior of the circuit, they divided the possible values for

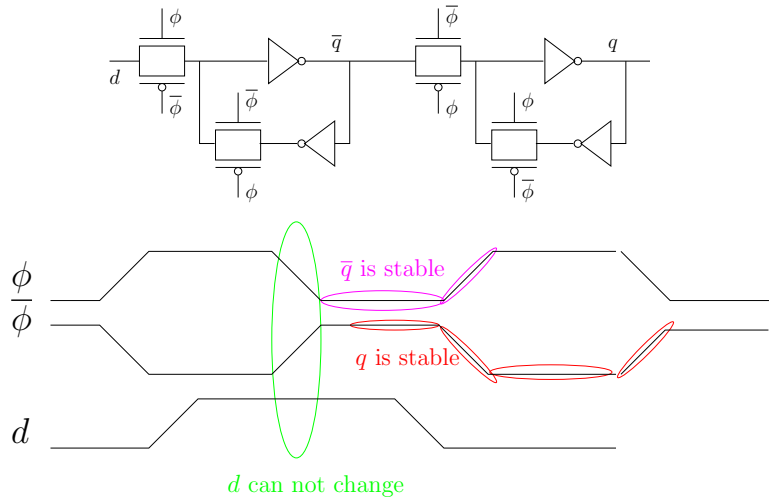


Figure 5.8: The Output Specification of Flip-Flop

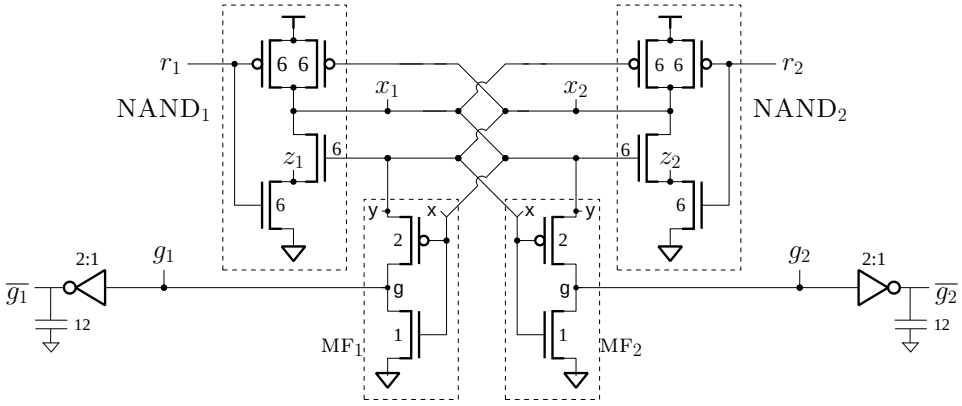


Figure 5.9: Verified Arbiter Circuit

each continuous state variable into 10 to 20 intervals, and computed the set of reachable hyper-rectangles using such a grid. Although the total number of possible hyper-rectangles is large, Kurshan and McMillan used COSPAN to construct the reachable space, and the next hyper-rectangle relation is only computed for reachable hyper-rectangles. Unlike our Brockett's annulus approach for specifying

signal transitions, Kurshan and McMillan model the inputs as making instantaneous transitions. These transitions were allowed at arbitrary times that satisfied the handshake protocol. More details of our approach are presented in [212, 213].

5.4.1 Reachability Computation

Rising transitions of the request signals for the two clients can occur concurrently. These requests can start at different times and have different rise-times. Verifying correct operation of the arbiter requires accounting for all allowed transitions of the inputs, including overlapping ones. We applied the method in Section 3.6.2 to model all input transitions. We excluded the state $B_{<4,4>}$ noting that there must be a failure of the arbiter or its clients if both requests are falling at the same time – this would imply that either the arbiter had violated the mutual-exclusion requirement or that at least one client had violated the handshake protocol. By exploiting the symmetry of the arbiter and its clients, we solve only one reachability problem to compute reachable regions of states $B_{<i,j>}$ and $B_{<j,i>}$. In order to reduce approximation error, we partitioned the rise and fall regions of the Brockett's annulus into seven subregions by employing the slicing technique from Section 5.1. This results in 136 states to perform the complete reachability computation for the arbiter.

Following the guidance in Section 5.1.2, we divided the 136 reachability problems into three phases using the assume-guarantee strategy from Section 5.1

1: From $B_{<1,1>}$ to $B_{<3,3>}$.
2: From $B_{<3,1>}$ to $B_{<3,1>}$.
3: From $B_{<3,3>}$ to $B_{<1,1>}$.

5.4.2 Stiffness

We encountered *stiffness* problems when verifying the arbiter circuit. A system of ordinary differential equations, $\dot{x} = f(x)$ is said to be *stiff* if the Jacobian of f is an ill-conditioned matrix. That is, $\|\lambda_{\max}/\lambda_{\min}\|$ is large where $\lambda_{\max}$ ($\lambda_{\min}$) is the largest (smallest) magnitude eigenvalue of the Jacobian matrix $Jac_f(x)$. For circuits, stiffness occurs when nodes have vastly different time-constants. This

occurs in the arbiter where nodes z_1 and z_2 have much smaller capacitances than the other nodes in the circuit.

The stiffness problems make it difficult for COHO to find a good choice of the time step size in the reachability computation. As described in Chapter 4, COHO has two principle causes of over approximation. First, there is an over approximation when producing a linear differential inclusion for a non-linear ODE as shown in Equation 4.15. Second, over approximations are introduced when projecting the reachable region for a face back down to the basis for the projection polygon. For the arbiter, if COHO chooses a large time step (suitable for the nodes other than z_1 and z_2), then the linearization errors for z_1 and z_2 will be large, creating large over approximations for the voltages of these nodes. As the currents flowing through the n-channel devices driving x_1 and x_2 are quite sensitive to z_1 and z_2, this leads to large over approximations for x_1 and x_2. Conversely, if COHO uses times steps that are small enough to obtain tight bounds for z_1 and z_2, the accumulated projection and simplification errors will be large for the other nodes. Thus, the goals of minimizing the approximation errors due to linearizing the model and minimizing the errors arising from projection and polygon simplification are in tension with each other. For any choice of time step size, we found that COHO produced false-negatives (failure to verify a correct circuit).

Circuit simulators such as HSPICE handle stiffness by using implicit integration algorithms. However, we are unaware of any formulation of an implicit algorithm that is compatible with a forward reachability computation such as used in COHO. As other reachability tools use similar methods, we expect that they will have similar problems if they attempt to verify common CMOS designs. This conjecture is supported by the absence of published results for formal verification of stiff systems. We implemented two methods to solve the stiffness problem. The first solution was to follow the example of typical designers and treat nodes z_1 and z_2 as if they had no capacitance. The other solution was based on a change of variables and constraining the reachability computation with an externally verified invariant. More details are presented in the remainder of this section.

Simplified Model

We first used a simplified model of the arbiter circuit to complete the reachability computation. This method side-stepped the problems of stiffness by treating nodes z_1 and z_2 as if they had no capacitance. With this assumption, the voltage on these nodes is always exactly the value that balances the currents flowing through the upper and lower n-channel transistors of each NAND gate. Thus, we created a model for a nMOS tetrode with source connected to ground, gates connected to r_1 and x_2, and drain connected to z_1, and another such tetrode for the two pull-down transistors for z_2. This simplification reduced the ODE model from six dimensions down to four and eliminated the stiffness issues. This assumption is reasonable as the internal nodes z_1 and z_2 have much smaller capacitances than other nodes of the circuit. In fact, many designers would instinctively ignore the contributions of these tiny capacitors. However, the verification is incomplete. For example, we note that with optical proximity rules, the spacing between series-connected transistors is growing relative to other circuit dimensions for sub-100nm processes. If the capacitances of these nodes are ignored, it is impossible to determine when they have become large enough to cause a circuit failure. Therefore, we implemented another solution to include internal nodes in our model.

Changing Variables and External Invariants

In order to reduce approximation error, we employed two techniques. First, we replaced variables z_1 and z_2 by two new variables which converge to zero rapidly. Second, we applied an externally verified invariant to constrain reachable regions. The remainder of this section presents our modifications to COHO's reachability computation that allow it to compute tight overapproximations.

When either transistor connected to node z_1 is conducting, z_1 tends to converge very quickly to a small neighborhood near its equilibrium value; however, the precise value of the equilibrium varies widely according to the values of r_1, x_1 and x_2. For any choice of values for the voltages of r_1, x_1 and x_2, there is a unique voltage for z_1 such that $\dot{z}_1 = 0$. This is because the current from x_1 to z_1 through the upper transistor is determined by the voltages of nodes x_1, x_2 and z_1 and is negative monotonic in the voltage of node z_1. Likewise, the current from z_1 to ground

through the lower transistor is determined by the voltages of nodes z_1 and r_1 and is positive monotonic in the voltage of node z_1. These properties hold for any realistic transistor model. Thus, given values for the voltages on nodes r_1, x_1 and x_2, there is a unique voltage for node z_1 such that these two currents are equal. This is the voltage at which $\dot{z}_1 = 0$, and we call this voltage the *equilibrium voltage* and denote it by $q_1(r_1, x_1, x_2)$. We define $q_2(r_2, x_2, x_1)$ in the analogous manner.

We replace z_1 and z_2 in the circuit's ODE with $u_1 = z_1 - q_1(r_1, x_1, x_2)$ and $u_2 = z_2 - q_2(r_2, x_2, x_1)$ respectively. Whenever a transistor driving z_1 is conducting, u_1 tends rapidly to zero, and it is much easier to show that u_1 converges to zero than to show that z_1 converges to a moving target. Likewise for u_2 and z_2. This change of variables formalizes the designer's intuition that the capacitance of nodes z_1 and z_2 "usually won't matter." The chain rule yields:

$$\dot{u}_1 \quad = \quad \dot{z}_1 - \left(\frac{\partial q_1}{\partial r_1}\dot{r}_1 + \frac{\partial q_1}{\partial x_1}\dot{x}_1 + \frac{\partial q_1}{\partial x_2}\dot{x}_2 \right). \tag{5.3}$$

Although z_1 is not a state variable of the modified ODE, it can be reconstructed by noting that $z_1 = u_1 + q_1(r_1, x_1, x_2)$, and then $\dot{z}_1$ can be determined based on the values for r_1, x_1, z_1 and x_2. In our implementation, we use a four-dimensional table, indexed by the values of u_1, r_1, x_1 and x_2 to compute values for $\dot{x}_1$ and $\dot{u}_1$. This table accounts for all four transistors of the NAND gate that produces x_1 and the capacitance on nodes x_1 and z_1. By including $\dot{x}_1$ in the table, we eliminate the need to reconstruct z_1 or calculate q_1. By directly computing $\dot{u}_1$ and $\dot{x}_1$, we avoid reconstructing z_1 with large error bound intervals that would then propagate to the other quantities. The same construction applies for computing $\dot{u}_2$ and $\dot{x}_2$.

With the change of variables described above, COHO still encountered a problem at the rising edge of r_1. If r_1 is low and r_2 is high, then x_2 will be low, and both transistors connected to node z_1 will be in cut-off. In this case, the equilibrium voltage for z_1 is determined by balancing the small leakage currents of the two transistors. Thus, u_1 can have a large value when r_1 is low. Following a rising edge of r_1, u_1 should go quickly to zero. However, COHO's over approximations from when r_1 was low led it to a region from which it could not establish this contraction. In fact, the range for u_1 blew up to cover the entire interval from 0 to V_{dd}, and led to continuous states that violated the specification from Figure 3.19.

We solved this problem by manually establishing a simple invariant. The intuition behind this invariant is that based on the leakage currents for our implementation in the TSMC 180nm, 1.8V CMOS process, we can determine that node z_1 eventually settles to 1.45V if $r_1 = 0$ and $x_2 = V_{dd}$, and this is an upper bound for z_1. By symmetry the same bound applies to z_2, from which we postulated the invariants $-1.45 \leq u_1 \leq 1.45V$ and $0 \leq z_1 \leq 1.45V$. It is straightforward to establish this invariant by computing the values of $\dot{u}_1$, $\dot{u}_2$, $\dot{z}_1$, and $\dot{z}_2$ on the boundary faces of this region to show that trajectories on these faces flow inward. We constrained the projectagons computed by COHO to satisfy these simple, externally verified invariants.

5.4.3 Results

Using the methods described above, COHO computed an invariant region for the arbiter. This set allows us to establish the correct operation of the arbiter as described below.

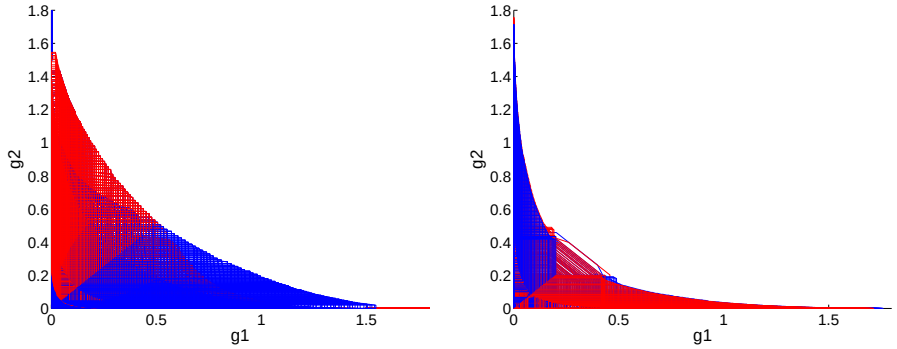
Safety Properties

Mutual Exclusion:

Figure 5.10 shows the verification of mutual exclusion. Part (a) of the figure shows all reachable states; clearly g_1 and g_2 are never both high. In fact, the region where they both reach values near 0.5V only occurs when one grant is falling and the other is rising as the arbiter transfers a grant that one client released and the other has requested. Figure 5.10.b shows the reachable space for g_1 and g_2 when falling transitions of the grant signals are excluded. This shows that the separation of grants is very distinct.

Handshake Protocol:

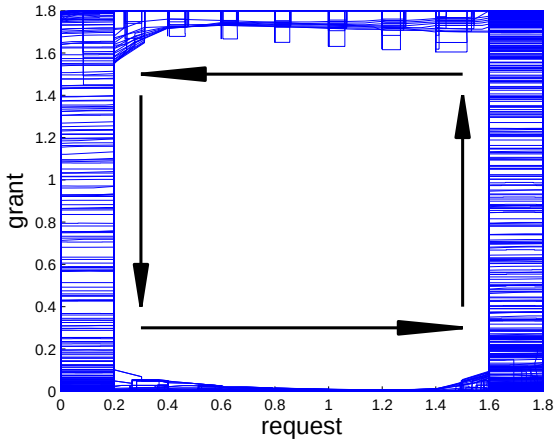
In a similar fashion, projecting the reachable space onto the signals g_1 and r_1 as shown in Figure 5.11 demonstrates that g_1 enters B_2 only when r_1 is in B_3. Likewise g_1 enters B_4 when $r_1 \in [0.0, 0.22]$. When $r_1 \in [0.2, 1.6]$, $\dot{r_1} < 0$, thus $r_1 \in B_4$. This shows that g_1 starts to fall only when the discrete abstraction of r_1 is a logically low signal. Thus, the grants both rise and fall in accordance with the handshake protocol.



a. g_1 vs. g_2 (including B_4)

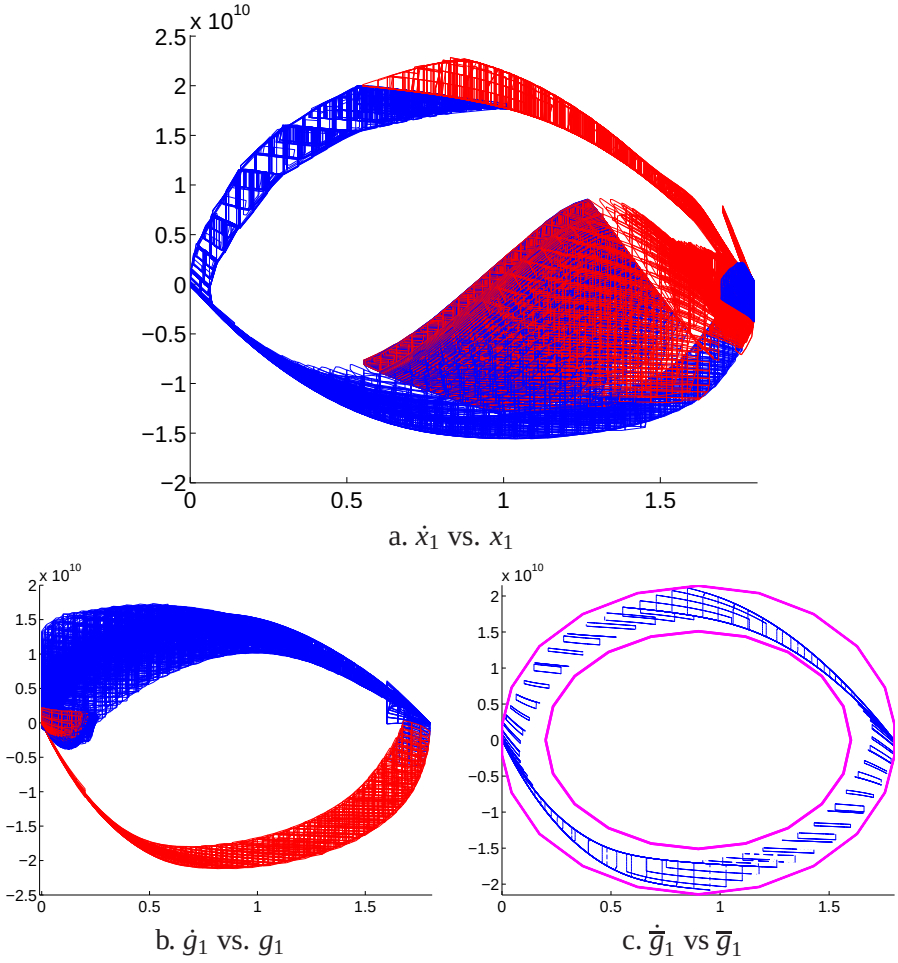
b. g_1 vs. g_2 (excluding B_4).

Figure 5.10: Verification of Arbiter: Mutual Exclusion. The left plot shows all reachable sets projected onto variables g_1 and g_2, where the blue regions are reachable sets computed by COHO and the red regions are computed by circuit symmetry. The right plot shows reachable space for g_1 and g_2 when falling transitions of the grant signals are excluded.



g_1 vs. r_1

Figure 5.11: Verification of Arbiter: Handshake Protocol. The blue polygons show the reachable sets projected onto variables g_1 and r_1, and the arrows illustrate the handshake protocol.



a. $\dot{x}_1$ vs. x_1

b. $\dot{g}_1$ vs. g_1

c. $\dot{\overline{g}}_1$ vs $\overline{g}_1$

Figure 5.12: Verification of Arbiter: Brockett's Annuli

Brockett's Annuli:

Figure 5.12 shows the derivatives of the x, g and $\overline{g}$ signals versus their voltages. The grant signal, g_1 satisfies a Brockett's annulus, but it is less restrictive than the one that we used for the request signals. In contrast, the time-derivative of signal x_1 has a much different shape. The lobe in the lower right shows the metastable behaviours: x_1 can start to fall, and then return to its nominal high value if x_2 wins the contest. The contrast between the plots for x_1 and g_1 shows the effectiveness of the metastability filter as a Brockett's annulus transformer. The output inverters

that produce $\overline{g}_1$ and $\overline{g}_2$ further improve the transitions to produce the plot shown in part c where the reachable space computed by Coho is indicated by the blue polygons, and the pink ellipses show the Brockett's annulus used for the request signals. With this output buffering, the output signals satisfy the original input annulus.

Liveness Properties

Initialization:

We used COHO to compute the reachable space when r_1 and r_2 were both low (i.e. in region B_1) starting from a state where x_1, x_2, g_1 and g_2 could be anywhere in $[0, 1.8]$. COHO establishes that within 200ps, x_1 and x_2 enter $[1.6V, 1.8V]$ (i.e. B_3), and g_1 and g_2 enter $[0.0V, 0.2V]$ (i.e. B_1). Thus, the arbiter can be initialized simply by not asserting any requests for a short time – no additional reset signal is needed.

Uncontested Requests:

We consider the reachable space with the additional restriction that r_2 remains within region B_1 (i.e. a logically low value). COHO shows that g_1 is asserted within 343 ps of r_1 rising (i.e. entering B_2). This shows that the arbiter is guaranteed to respond to uncontested requests within a bounded amount of time.

Contested Requests:

If r_1 and r_2 are asserted at nearly the same time, the arbiter may exhibit metastable behavior and may remain in a metastable state for an arbitrarily long period of time. COHO can show that metastability can only occur in the hyper-rectangle where:

$$\begin{array}{lll} r_1 \in B_3 & x_1 \in [0.55, 1.3] & g_1 \in B_1 \\ r_2 \in B_3 & x_2 \in [0.55, 1.3] & g_2 \in B_1. \end{array} \tag{5.4}$$

Outside of this region, the dot product of the derivative vector with the final stable state of granting client 1 or granting client 2 is unambiguous. Section 5.4.4 explores liveness under metastable conditions in greater detail.

Reset:

COHO shows that if client *i* has a grant and lowers its request signal then the arbiter lowers the grant for client *i* within 270 ps. This shows that the arbiter satisfies the liveness requirement for withdrawing grants.

Fairness:
If client *i* wins a grant while the other client is making a request and subsequently client *i* drops its request, then Coho shows that the other client receives a grant within 420 ps. This shows that the other client receives at most one grant while the current client has a pending request; therefore, this simple arbiter is fair.

Using COHO, we have verified all properties of the arbiter from the specification given in Figure 3.19 except for the clause

$$\alpha\text{-ins} \Rightarrow (\Box_Z(B_3(r_i) \overset{\mathbf{U}}{\Rightarrow} B_{2,3}(g_i))).$$

This property concerns the behaviour of the arbiter under metastable conditions, and we describe our technique for verifying this property in the next section.

5.4.4 Metastability and Liveness

Metastable behavior in digital circuits has been studied since Chaney and Molnar's original paper on synchronizer failures [38]. Hurtado [124] analyzed metastability from a dynamical systems perspective. Seitz [183] gave a nice introduction to metastability issues, and Marino [154] provided a fairly comprehensive treatment.

When both requests are asserted concurrently, the arbiter may enter a metastable condition that can persist for an arbitrarily long time. Thus, it is not possible to prove that all behaviours when both requests are simultaneously asserted will eventually lead to granting a client. On the other hand, with a properly designed arbiter, the probability of no grant being issued when both requests are asserted should decrease exponentially with time. This implies that the probability of a liveness failure should go to zero as the settling time goes to infinity. If the arbiter works for all situations except for some set with a probability measure of zero, then we say that the circuit works *almost surely* [169, Chapter 2.6]. Our approach is to use COHO to bound the reachable space when both requests are asserted. Most of this space can be shown to quickly resolve to granting one client or the other. For a small region near the metastable point, such progress can't be demonstrated, and we use the method from [160] to show that this metastable region is exited with probability one. The remainder of this section describes these steps in greater detail.

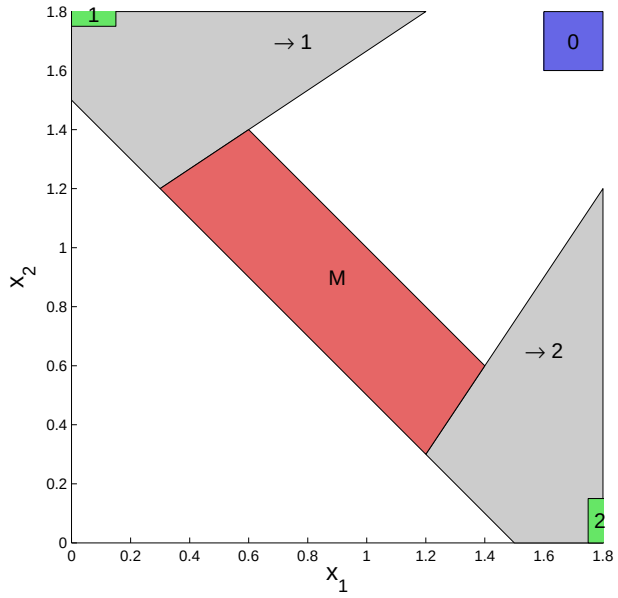


Figure 5.13: Reachable Regions When r_1 and r_2 are High

To verify liveness when both requests are asserted, we first use COHO to compute the reachable state after both requests are asserted. Figure 5.13 shows the outcome of this analysis. The square marked **0** indicates the initial region for x_1 and x_2 when r_1 and r_2 are both in region B_1. COHO then determines the reachable space for all low-to-high transitions of r_1 and r_2 allowed by their Brockett's rings. After both r_1 and r_2 have been in region B_3 for a while, COHO shows that the state is in the union of the regions labeled **1**, **2**, $\rightarrow$**1**, $\rightarrow$**2**, and **M**. Region **1** is where x_1 has gone low, and grant g_1 will be asserted. COHO shows that all trajectories in region $\rightarrow$**1** converge to region **1** in bounded time and thus lead to asserting grant g_1. Likewise, from regions **2** and $\rightarrow$**2** lead to asserting g_2. Region **M** contains the metastable point. Because trajectories on the stable manifold for the metastable point remain in region **M** indefinitely, COHO cannot verify that all such trajectories eventually lead to issuing a grant.

We now show that the arbiter is live in the almost-sure sense. We first make a

change of variables. Let

$$w^- = x_1 - x_2, \qquad w^+ = x_1 + x_2. \tag{5.5}$$

We will write $\mathbf{M}_w$ to denote the region M in (w^-, w^+) coordinates. As shown in [160], almost-all trajectories diverge from region $\mathbf{M}$ if we can find constants $c, d > 0$ such that for all points (w_1^-, w_1^+) and (w_2^-, w_2^+) with $w_1^- < w_2^-$,

$$\begin{array}{rcl} ((w_2^+ - w_1^+) \leq c(w_2^- - w_1^-)) & \Rightarrow & \dot{w}_2^- - \dot{w}_1^- \geq \quad d(w_2^- - w_1^-) \\ (w_2^+ > w_1^+) \wedge ((w_2^+ - w_1^+) = c(w_2^- - w_1^-)) & \Rightarrow & \dot{w}_2^+ - \dot{w}_1^+ \leq -d(w_2^- - w_1^-) \\ (w_2^+ < w_1^+) \wedge ((w_2^+ - w_1^+) = c(w_2^- - w_1^-)) & \Rightarrow & \dot{w}_1^+ - \dot{w}_2^+ \leq -d(w_1^- - w_2^-). \end{array} \tag{5.6}$$

Basically, this condition ensures that if at some time, t_0 two trajectories differ only by their w^- components, then they must exponentially diverge; this ensures that one of them must leave region $\mathbf{M}$. This implies that the set of trajectories that remain in $\mathbf{M}$ indefinitely must have a Lesbesgue measure of zero, which yields the desired almost-surely result.

We now present a simple test that ensures that the conditions from Equation 5.6 hold. For any point $w \in \mathbf{M}_w$, let $J(w)$ be the Jacobian operator for the ODE model projected onto the (w^-, w^+) space:

$$J(w) = \begin{bmatrix} \frac{\partial \dot{w}^-}{\partial w^-} & \frac{\partial \dot{w}^-}{\partial w^+} \\ \frac{\partial \dot{w}^+}{\partial w^-} & \frac{\partial \dot{w}^+}{\partial w^+} \end{bmatrix}. \tag{5.7}$$

Now, define:

$$\begin{array}{rcl} \mu & = & \min_{w \in M}(J(w)(1,1), -J(w)(2,2)) \\ h_1 & = & \left(\max_{w \in M} |J(w)(1,2)|/J(w)(1,1) \right)^{-1} \\ h_2 & = & \max_{w \in M} -|J(w)(2,1)|/J(w)(2,2). \end{array} \tag{5.8}$$

The conditions of Equation 5.6 are satisfied if

$$(\mu > 0) \quad \wedge \quad (h_1 > h_2), \tag{5.9}$$

where μ, h_1 and h_2 are defined as in Equation 5.8. This is straightforward to show by integration along the line segment from point (w_1^-, w_1^+) to point (w_2^-, w_2^+) from Equation 5.6. We applied these tests to show that trajectories will leave region M

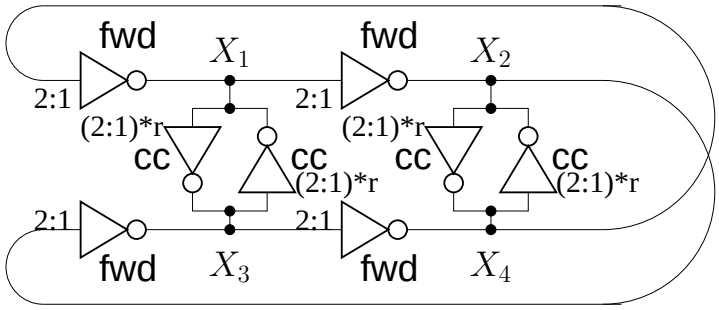


Figure 5.14: Verified Two-Stage Rambus Ring Oscillator. It consists of forward inverters (labeled as fwd) and cross-couple inverters (labeled as *cc*). The ratio of cross-couple inverter size and forward inverter size is denoted as *r*.

almost-surely.

In summary, we have verified both safety and liveness properties of the arbiter circuit. We also showed that the metastability filters function as a Brockett's annulus transformer. During the verification, we found the challenging stiffness problem and proposed two solutions. We applied a method based on dynamical systems theory to prove that the arbiter circuit satisfies its liveness properties in the almost-surely sense, which cannot be demonstrated by reachability analysis alone.

5.5 The Rambus Ring Oscillator

Figure 5.14 shows the two-stage ring oscillator that we verified. The circuit was proposed by researchers at Rambus [129] as a verification challenge, and they noted that some implementations of the circuit had failed in real, fabricated chips. Therefore, they posed the problem of showing that the oscillator starts from all initial conditions for a particular choice of transistor sizes. Although [88] establishes a condition to ensure that the oscillator is free from lock-up, there is still the possibility that the circuit could oscillate at a harmonic of the intended frequency, display chaotic behavior, or have some other steady-state behavior other than the intended oscillation. We solved these problems for the two-stage Rambus oscillator shown in Figure 5.14. In the circuit, all inverters have the same size and signal nodes

are denoted as x_1, x_2, x_3, x_4. With the circuit states computed by COHO, we have shown that the two stage oscillator with any possible initial condition oscillates in the specified mode with probability one.

5.5.1 Static Analysis and Reachability Computation

Our verification proceeds in three main phases:

1. The oscillator shown in Figure 5.14 is a differential design: nodes X_1 and X_3 form a "differential pair" and likewise for nodes X_2 and X_4. The first phase of the verification shows that each of these differential pairs can be treated as a single signal. This symmetry reduction of the state space simplifies the subsequent analysis.

2. Any oscillator must have at least one equilibrium point. Using the methods from [88], we can show that any such equilibrium points are unstable; however, we note that any trajectory that starts on the stable manifold for such an equilibrium point will lead to a non-oscillating behaviour. In other words, for *any* oscillator circuit, there must exist an infinite set of initial conditions for which the circuit will fail to oscillate. Fortunately, this failure set can have a lower dimensionality that the full state space. Thus, the second phase of the verification shows that this occurs with probability zero.

3. The first two phases show that most initial conditions lead to a fairly small subset of the full phase space. In the final phase, we divide the remaining space into small regions, and use existing reachability methods to show that the oscillator starts up properly from each such region.

The remainder of this section describes the dynamical systems issues associated with each of these phases. Section 5.5.2 describes our verification method based on these observations.

We model the oscillator circuit from Figure 5.14 using non-linear ordinary differential equations (ODEs) obtained by standard, modified nodal-analysis methods as described in Section 3.3. This gives us an equation of the form:

$$\dot{x} = f(x), \tag{5.10}$$

where x is a vector of node voltages. Let d be the dimensionality of x. We assume that f is C^2 which guarantees that Equation 5.10 has a unique solution for any initial state, $x(0)$.

Differential Behaviour

Nodes X_1 and X_3 in the oscillator from Figure 5.14 form a "differential pair" and likewise for nodes X_2 and X_4. Let x_i denote the voltage on node X_i. The *differential component* of the differential pair is $x_1 - x_3$, and the *common mode* component is $x_1 + x_3$. When the oscillator is operating properly, the common mode components are roughly constant and the oscillation is manifested in the differential components. Let V_0^+ be the nominal value for the common mode components. We show that for a relatively small V_{err}, if $|x_1 + x_3 - V_0^+| > V_{err}$, then $\frac{d}{dt}(x_1 + x_3)$ and $(x_1 + x_3)$ have opposite signs. This shows that that the common mode component for X_1 and X_3 converges to within V_{err} of the nominal value. Likewise for X_2 and X_4 by circuit symmetry.

Equilibrium Points and Their Manifolds

If the circuit is an oscillator, then the dynamical system described by Equation 5.10 must have a periodic attractor. This is a periodic orbit in $\mathbb{R}^d$ such that any trajectory that starts in some open neighbourhood of this orbit must asymptotically converge to the orbit. For any periodic attractor, there must be an associated equilibrium-point [121, Chapter 13], i.e. a point x_{ep} for which $f(x_{ep}) = 0$. If this equilibrium point is an attractor, then its basin of attraction is a set of initial conditions for which the circuit will not oscillate. Otherwise, the equilibrium point may be a saddle point, in which case it has an associated stable manifold. This manifold is a set of points that form a surface with a dimensionality $< d$ such that trajectories starting anywhere on this manifold converge to x_{ep}. In this case, the set of initial conditions that lead to x_{ep} has zero volume in the full-dimensional space, and the probability of starting at one of these points is zero. More technically, the failure set associated with this equilibrium point is *negligible* (with respect to the Lesbegue measure) and trajectories diverge from the neighbourhood of x_{ep} *almost surely* [169, Chapter 2.6]. We use the terminology *almost surely* to indicate

something that has probability one, and *almost all* to indicate the entire state space minus a negligible set.

This dynamical systems perspective provides a critical observation about oscillators: *every oscillator circuit has a set of initial conditions for which it fails to oscillate.* Direct application of continuous state-space model checkers (e.g. [69, 97]) to the oscillator start-up problem will identify regions where trajectories might stay forever (or the reachability computation used was unsound). Because we cannot show that the set of failure states is empty, we must settle for showing that it is negligible. This is sufficient in practice, as designers are not worried about a design that fails with probability zero. To perform this verification, we need reachability-modulo-measure-theory. We describe such a method below.

To verify the oscillator, we extend the technique described in Section 5.4.4. The basic idea is straightforward. Let x_{eq} be an equilibrium point of Equation 5.10, $f(x_{eq}) = 0$. Let $y_1 \ldots y_k$ and $z_{k+1} \ldots z_d$ be orthonormal vectors and let B be a simple region. If there is some constant, $\mu > 0$, such that for every point, b, in B,

$$\begin{array}{rcll} y_i \cdot \frac{\partial f}{\partial y_i}(b) & > & \mu, & \forall i \in 1 \ldots k \\ z_j \cdot \frac{\partial f}{\partial z_j}(b) & > & -\mu, & \forall j \in k+1 \ldots d, \end{array} \tag{5.11}$$

then it can be proved [160] that almost all trajectories in B leave B. The intuition is that trajectories that start from points that differ only in their $y_{1\ldots k}$ components must diverge from each other.

Reachability Computation

The verification problem that we consider is to show that a Rambus ring oscillator with a particular choice of transistor sizes will oscillate in its fundamental mode from nearly all initial conditions. We do this by first showing differential operation and then showing that almost all trajectories diverge from the stable manifold of the unstable equilibrium point. These first two phases show that trajectories from almost all initial conditions lead to a relatively small part of the state space. Furthermore, this small part of the state space has the common mode components of both differential signal pairs within V_{err} of V_0^+. This allows us to rewrite the

differential equation model from Equation 5.10 as a differential inclusion [105]:

$$\dot{u} \quad \in \quad F(u), \tag{5.12}$$

where u is the vector $[x_1 - x_3, x_2 - x_4]$. By using an inclusion, F accounts for *all* values of the common mode components in $V_0^+ \pm V_{err}$. Reducing the four-dimensional state space of the original problem to a two-dimensional space makes the exploration of trajectories from all remaining start conditions straightforward.

By showing that all such trajectories lead to an oscillation in the fundamental mode, we solve the first part of the challenge problem from [129]: we show that for a particular choice of transistor sizes, the circuit will start oscillation from almost all initial conditions. Extending the approach to handle the second part of the challenge is straightforward. To ensure sound verification, all of the steps of the verification use over approximations of the circuit model. We can use these uncertainty terms to model a range of size ratios between the forward and cross-coupled inverters. Thus, we will show that the oscillator starts up properly for any ratio of transistor sizes in a relatively wide interval.

5.5.2 Implementation

As with the previous examples, our verification is for a design in the TSMC 180μ 1.8V CMOS process.

Differential Operation

This verification phase starts by changing the coordinate system to one based on the differential and common mode representation of signals. Then, a static analysis of the trajectory flows allows most of the common-mode subspace to be eliminated from further consideration.

Let u be the circuit state in "differential" coordinates:

$$\begin{aligned} u &= M^{-1}x \\ M &= \frac{\sqrt{2}}{2}\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}. \end{aligned} \tag{5.13}$$

We assume each of nodes X_1, X_2, X_3 and X_4 can independently have initial voltages anywhere in $[0,1.8]$V. Thus, the differential components, u_1 and u_2, are initially in $\pm 0.9\sqrt{2}$, and the common mode components, u_3 and u_4, are initially in $[0,1.8\sqrt{2}]$.

To establish differential operation, we divide the range of each of the u variables into n intervals, creating n^4 cubes. We construct a graph, $G = (V,E)$ to represent the reachability relationship between these cubes. Let $v_{i,j,k,\ell}$ be a vertex corresponding to the i^{th} interval for u_1, the j^{th} interval for u_2 and so on. There is an edge from v to w if f allows a flow out of the cube for v directly into the cube for w, and there is a self-loop for v if each component of f is zero somewhere in v. The key idea is that if vertex G has no incoming edges, then any trajectory that starts in the corresponding cube will eventually leave that cube, and no trajectories will ever enter the cube. Such a cube can be eliminated from further consideration. Thus, we only need to consider cubes whose vertices are members of cycles. These vertices can be identified in $O(V+E) = O(n^4)$ time. With a direct implementation of this computation, constructing G dominates the entire time for verifying the oscillator.

To obtain a more efficient computation, we first note that the goal is to establish differential operation. It is sufficient to project the vertices of V onto the common-mode components of the differential signals and show that most vertices can be eliminated from further consideration. Let $G' = (V',E')$ where $v'_{k,\ell}$ corresponds to the k^{th} interval of u_3 and the ℓ^{th} interval of u_4. There is an edge in E' from v'_{k_1,ℓ_1} to v'_{k_2,ℓ_2} iff there is an edge in E from v_{i,j,k_1,ℓ_1} to v_{i,j,k_2,ℓ_2} for some i and j. Clearly, G' overapproximates reachability. Thus, if a vertex of G' has no incoming edges, then all of the corresponding vertices in G must have no incoming edges as well. Computing the edges in E' requires examining all of the edges of E, but subsequent operations on the graph G' are much faster than those on G.

To reduce the time required to find edges of E, we start with a small value of n and thus a coarse grid. Many large blocks can be eliminated from G' even with a coarse grid. We then double n (i.e. divide each vertex of G' into four) and recompute reachability using the finer grid for finding edges in E as well. In practice this adaptive gridding approach eliminates blocks quickly while achieving enough precision to allow the rest of the verification to proceed without difficulties.

Negligible Failure Set

As described in Section 5.5.1, we can eliminate cubes from further consideration if we can find a constant $\mu > 0$ and orthonormal vectors y_1, y_2, z_1 and z_2 such that for every point x in the cube,

$$\begin{array}{rcl} y_1 \cdot \frac{\partial f}{\partial y_1}(x) & > & \mu, \\ y_2 \cdot \frac{\partial f}{\partial y_2}(x) & > & \mu, \\ z_1 \cdot \frac{\partial f}{\partial z_1}(x) & < & -\mu, \\ z_2 \cdot \frac{\partial f}{\partial z_2}(x) & < & -\mu. \end{array} \tag{5.14}$$

We note that $f(x) = 0$ when $x_1 = x_2 = x_3 = x_4 = v$ with v near $V_{dd}/2$. Let x_{eq} denote this point. The stable manifold for this attractor is the plane defined by $(x_1 = x_3) \wedge (x_2 = x_4)$. The Jacobian matrix for f at x_{eq} has two eigenvalues with positive real parts (divergent trajectories), and two with negative real parts (convergence in a subspace). The corresponding eigenvectors are the columns of M as defined in Equation 5.13. This suggests choosing the y and z vectors as:

$$y_1 = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \end{bmatrix}, \quad y_2 = \frac{\sqrt{2}}{2}\begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix},$$

$$y_3 = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad y_4 = \frac{\sqrt{2}}{2}\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}.$$

While we chose the y and z vectors based on the manual analysis of the dynamics of the oscillator sketched above, we believe that this process could be automated. Equilibrium points can be found by standard root-finding methods, and the eigenvalues and eigenvectors can be computed using standard numerical linear algebra routines.

Having chosen y and z vectors, we now check each remaining cube in the state space to determine if it satisfies the conditions of Equation 5.14[1]. This requires computing bounds on the partial derivatives of f. The convexity (and concavity) of transistor currents with respect to node voltages allows the range of possible partial derivatives of transistor currents with respect to node voltages to be determined from the end-points of the intervals for these voltages. By our simplification of using constant capacitances, f is a linear function of transistor drain-to-source currents. Thus, we can combine the intervals for the partial derivatives of transistor currents to obtain over-approximations of the intervals for partial derivatives of f.

At the end of this phase, the number of cubes to consider for the final reachability analysis has been reduced to a small fraction of the original. More importantly, cubes that contain or are near the unstable equilibrium point of the oscillator have been safely eliminated. This allows a reachability computation from the remaining cubes to complete the verification.

Proper Oscillation

Noting that the common mode voltages u_3 and u_4 are restricted to a small region as shown in Figure 5.15, we eliminate these two variables by replacing the differential equation model for the circuit with a differential inclusion. This reduces the state space from four dimensions to two which enables efficient reachability computation. Figure 5.16 shows the region that remains to be verified. We divide this region into its inner and outer boundaries, and a collection of "spokes" as shown in Figure 5.17. The computation has three parts:

1. Starting from each "spoke", show that all trajectories starting at that spoke eventually cross the next spoke.

[1]The details of our method for establishing the inequalities from Equation 5.14 are described in [209]

2. Show that all trajectories starting from the inner or outer boundary eventually cross the next spoke.

3. Starting from one spoke, compute the reachable set until it converges to a limit set.

The first two show that all trajectories converge to the same attractor. This means that all initial conditions lead to a unique mode of oscillation. The final step tightens the bound on this unique mode.

5.5.3 Results

As described earlier, our verification is for designs in the TSMC 0.18μ process. All transistors in the designs that we considered have gate lengths of 0.18μ. For all inverters, we use pMOS devices that are twice as wide as the nMOS devices. All forward inverters have transistors of the same size, and likewise for the cross-coupled inverters. In the following, r denotes the ratio of the cross-coupled inverter size to the forward inverter size. This section first presents the verification of an oscillator with $r = 1$. Then, the oscillator is verified for $0.875 \leq r \leq 2.0$.

The verification routines were implemented using Matlab with COHO used for the final reachability computation. All times were obtained running on a dual Xeon E5520 (quad core) 2.27GHz machine with 32GB of memory; however, the computations described here are all performed using a single core.

Verification with Equal-Size Inverters

The first phase of the verification establishes differential operation. Initially, the computation partitions the space for each of the u_i variables into 8 regions, creating a total of $8^4 = 4096$ cubes to explore. After eliminating cubes that have no incoming or self-circulating flows, the remaining cubes are subdivided and rechecked until there are 64 intervals for each variable. Figure 5.15 shows the remaining cubes projected onto the common-mode variables, u_3 and u_4 at the end of this phase.

With 8 intervals per region, there are 752 cubes under consideration (18% of the total space). With each subdivision, the number of cubes remaining increases by a factor of roughly 4.6, and thus the volume of the space under consideration

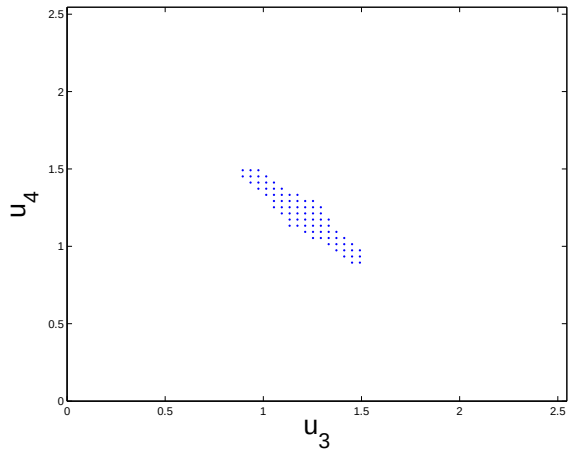


Figure 5.15: Common-Mode Convergence to $V_{dd}/\sqrt{2}$. The first phase of verification shows that the common mode components are roughly constant. It eliminates cubes that have no incoming or self-circulating flows. This plot shows the remaining cubes projected onto the common-mode variables u_3 and u_4.

drops by about a factor of roughly 0.29. With 64 intervals per region, 74676 cubes remain (0.45% of the total space). The decrease in the volume is steady, suggesting that further reductions would be possible with more iterations. However, the time per iteration increases with the number of cubes under consideration, and the time for this phase dominates the total verification time. Thus, for verifying this circuit, there is no incentive to further refine the region bounding the common-mode signal.

The second phase of the verification eliminates the unstable equilibrium and cubes near this equilibrium's stable manifold. It starts with the 74676 cubes from the previous phase and performs the computation steps described in Section 5.5.2. Figure 5.16 shows the remaining cubes projected onto the differential variables, u_1 and u_2 at the end of this phase. This phase eliminates roughly half of the remaining cubes, leaving 38384 cubes for analysis by the final phase.

The final phase starts with the 38384 cubes from the second phase. As described in Section 5.5.2, we divide these cubes into 16 wedges divided by "spokes" in the $u_1 \times u_2$ projection. For each such wedge, it is sufficient to show that all trajectories starting on the boundary of the wedge lead to points inside the next wedge

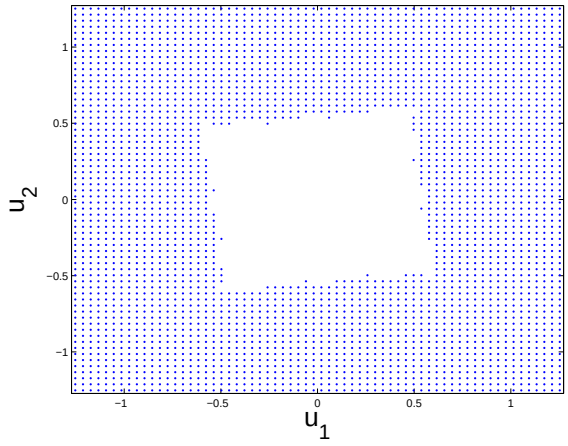


Figure 5.16: Eliminating the Unstable Equilibrium. The second phase eliminates the unstable equilibrium and cubes near the stable manifold. This plot shows the remaining cubes projected onto the differential variables u_1 and u_2.

in the clockwise direction. This is shown with three reachability computations per wedge:

1. Show that all trajectories starting from cubes on the leading spoke (counterclockwise boundary of the wedge) cross the trailing spoke (clockwise boundary) in the interior of the wedge. These spokes are drawn in magenta in Figure 5.17.

2. Show that all trajectories starting from cubes on the inner boundary cross the trailing spoke. These cubes are drawn in green in Figure 5.17.

3. Show that all trajectories starting from cubes on the outer boundary cross the trailing spoke. These cubes are also drawn in green in Figure 5.17.

With 16 wedges, we perform 48 reachability computation runs. At this point, the oscillator is verified.

We also ran a longer reachability computation starting from a spoke and completing two complete cycles of the oscillation. The second cycle starts from a smaller region that the first and establishes tighter bounds on the limit cycle. The

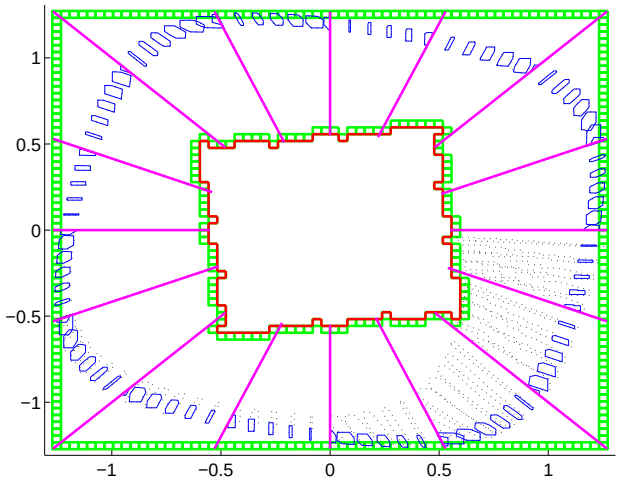


Figure 5.17: Computing the Invariant Set. The third phase computes reachable regions starting from all remaining cubes. It projects all cubes onto the two-dimensional plane by differential variables u_1 and u_2. The u_1, u_2 plane is partitioned by 16 spokes (magenta lines). All trajectories starting from the inner or outer boundary (green cubes) are shown to cross the next spoke. The dotted polygons show the reachable sets from a spoke and the blue polygons show the final invariant set of the oscillator.

blue polygons in Figure 5.17 indicate this limit cycle. The remaining width of the limit cycle is mainly due to approximating the four-dimensional differential equation with a differential inclusion.

Verification for a Range of Inverter Sizes

Each phase of our verification method uses conservative over-approximations to guarantee soundness of the results. These approximations make it straightforward to model r as being in an interval rather than having a precise value. With this change, we verified that the oscillator starts up properly for any value $0.875 \leq r \leq 2$. For $r < 0.875$, the second phase of the verification fails to show that the oscillator escapes from the region near the stable equilibrium. For $0.72 < r < 0.9$, DC-equilibrium analysis along the lines described in [88] shows that the oscillator

Table 5.3: Verification Times (seconds)

r	phase 1	phase 2	phase 3	Total	verified?
[0.7,0.9]	1122	464	—	1586	NO
[0.85,0.9]	705	237	—	942	NO
[0.875,0.9]	652	209	659	1520	YES
[0.9,1.1]	724	257	468	1449	YES
[1.1,1.3]	533	171	382	1086	YES
[1.3,1.5]	429	132	402	963	YES
[1.5,1.7]	378	112	512	1002	YES
[1.7,1.9]	335	99	624	1058	YES
[1.9,2.0]	308	91	688	1087	YES
[2.0,2.1]	308	91	1150	1549	NO
[2.1,2.3]	293	80	3879	4252	NO

has three unstable equilibria. We conjecture that the second phase is failing because it cannot distinguish the stable manifolds for the multiple equilibrium points. The DC analysis method shows that the oscillator is susceptible to lock-up for $r < 0.72$. For $r > 2$, the third phase of the verification fails to show that trajectories leave the "corners" of the $u_1 \times u_2$ space. These correspond to lock-up of the cross-coupled inverters. The DC analysis method shows that these lock-up states become stable for $r > 2.25$. The gap between the reachability computation and the DC analysis is presumably due to conservative approximations used in the reachability method.

Table 5.3 shows the run times for each phase of the verification. In general, the time for the first phase decreases with larger values of r because the stronger cross-coupled inverters eliminate the common-mode component of the signals faster. For the same reason, the number of cubes at the end of phase 1 decreases with larger values of r resulting in less time for phase 2. If the second phase shows a failure, we don't run phase 3. Generally, the run-time for phase three increases with larger r because the oscillator period increases, and it takes the reachability computation more steps to show that trajectories exit regions where the cross-coupled inverters are close to locking up. For the smallest values of r, the oscillator converges to its limit set more slowly, and we ran the reachability computation longer to establish a small limit set.

6

Conclusion and Future Work

This dissertation has addressed formal verification of analog and mixed signal circuits. Our approach is based on translating verification problems to reachability analysis problems by modeling circuit dynamics as ODEs and specifying analog properties based on Brockett's annuli and a LTL-based logic with continuous time and values (see Chapter 3). Reachable circuit states are computed by COHO, which is a reachability analysis platform for nonlinear, moderate-dimensional hybrid systems. COHO employs projectagons to represent and manipulate moderate-dimensional, non-convex objects and bounds the solution of nonlinear ODEs by approximating ODEs by differential inclusions (see Chapter 4). The correctness and efficiency of our solution have been demonstrated by the success of verifying several synchronous, asynchronous, and analog circuits (see Chapter 5). Our work has extended the application of formal methods to practical analog circuits and also motivates many future research topics.

6.1 Contributions

We proposed a reachability based solution to circuit verification. It represents circuit states by moderate-dimensional regions and computes all circuit states using COHO. COHO is a sound, efficient and robust reachability analysis tool for nonlinear, moderate-dimensional systems. Our approach has successfully verified several real circuits.

We employed a MNA (modified nodal analysis) based technique to model a circuit by a system of nonlinear ODEs. The drain-to-source current functions for transistors are obtained by HSPICE simulations. This simple and general method can generate accurate models for any process and any device. We developed a polynomial model which applies quadratic polynomials to approximate current functions in order to support larger devices with more terminals. We also developed a cosine-window interpolation method to ensure the smoothness of these models, and so enable us to use the same model for both simulation and verification.

To formally specify analog properties, we introduced simple extensions to LTL to support dense time and continuous variables. We also introduced probability into the logic to express nondeterministic properties such as metastability behaviours. We applied the Brockett's annulus construction to specify a family of signals and map between continuous trajectories and discrete behaviors.

With the mathematical model and specification method, we applied reachability analysis to compute forward reachable regions from initial states. We developed new algorithms to provide robust and efficient manipulation of the projectagon data structure originally proposed in [86] to represent and manipulate high-dimensional, non-convex regions. A projectagon maintains two structures: the geometric representation which projects a high-dimensional object onto two-dimensional subspaces, and the inequality representation which applies linear inequalities to bound the convex hull of the object. The geometric representation reduces the number of dimensions of the object and implements operations on high-dimensional objects based on efficient polygon operations, such as unions and intersections of two projectagons. On the other hand, the inequality representation can implement some operations more efficiently based on linear programming and a new interval closure operation; these operations are used to find bounds on variables and project high-dimensional faces onto the projection planes of the projectagon.

We developed an efficient and robust algorithm to compute the successor of a projectagon with dynamics described by ODEs based on the algorithm from [87]. Our implementation advances projectagon faces rather than the entire projectagon because trajectories starting on these faces establish bounds for trajectories starting anywhere in the projectagon. We made several improvements to the algorithm from [87]. First, we use the maximum principle to bound solutions of linear dif-

ferential inclusions; this ensures the soundness of our version. Second, we developed a completely new algorithm for projecting advanced faces onto the projection planes; thereby solving many of the robustness issues of the version from [87]. This algorithm is based on linear programming and requires exact solution of linear programs. We implemented an exact linear program solver which uses arbitrary precision rational numbers to guarantee the optimal solution and employs an efficient ($O(n)$-time) linear system solver to improve performance. Third, the early version of COHO from [87] would fail in the presence of "infeasible vertices" which arose when over approximations in projecting a face produced a vertex in a projection polygon that fails to satisfy the constraints implied by other projections. We solved this problem of infeasible vertices by iteratively trimming projection polygons produced by the projection algorithm.

We developed several algorithms to improve performance and reduce approximation errors. The multiple-model and asymmetric bloating methods reduce modeling error and exclude non-physical trajectories. The guess-verify strategy decreases the number of steps by adaptively guessing a larger, valid step size. We developed approximate algorithms for our LP solver and projection operation which improved performance significantly. We developed a hybrid implementation of the LP solver that combines floating-point interval arithmetic and arbitrary-precision-rational (APR) computations. The LP solver performs most of its computation using floating point arithmetic, but can detect when critical round-off errors occur. In the latter case, the computation is repeated with APR arithmetic. In practice, this provides the speed of hardware-supported floating point computations with the robustness and soundness of APR.

With the techniques and algorithms described above, we implemented COHO, a sound, robust, efficient and accurate reachability analysis tool for nonlinear, moderate-dimensional hybrid systems. Furthermore, we developed a framework to verify analog and mixed signal circuits using continuous models. First, we provided a standard, easy-to-use interface for the tool. With this interface, reachability computation for a circuit can be formally described by a hybrid automaton. We also provided a technique to model input transitions for circuits with multiple input signals. The method models all possible trajectories using a finite number of automata states for either independent or related input signals.

We applied our methods to verify synchronous, asynchronous and analog circuits. We found an invariant set for a toggle circuit and verified that its output and input satisfy the same specification, thus a ripple counter can be constructed using this toggle circuit. We verified that the output of a flip-flop circuit satisfies a Brockett's annulus if its input specification is satisfied. We also measured the clock-to-q delay and the maximum frequency of the flip-flop. We verified both safety and liveness properties of an asynchronous arbiter circuit. However, a challenging stiffness problem was encountered during the reachability computation. We presented two techniques that can addressed the stiffness problem. First, we simplified the circuit by a model-reduction technique proposed in [83] that eliminates nodes with small capacitances[1]. Our second technique involved a change of variables of the ODEs in a way that made the stiffness more manageable along with introducing a simple invariant to reduce approximation error. This invariant was established by static analysis techniques. We also developed a method based on dynamical system theory to show the probability of staying in the metastable region is zero. The Rambus ring oscillator is a real circuit from industry. We combined static analysis and reachability computation to show that the circuit always oscillates as expected from all initial conditions except for a set of measure zero. The success of these verifications demonstrated the robustness and efficacy of our algorithms and the COHO tool.

From these verification experiences, we learned several lessons. First, it is possible and necessary to apply circuit-level models to formal verifications. We could not find the potential flaw of the toggle circuit caused by leakage currents if we only used digital models. Second, stiffness is a problem for reachability analysis. We believe that stiffness will arise in many circuit verification problems because it is common for nodes to have capacitances and associated time constants that differ by several orders of magnitude. Although stiffness has been thoroughly addressed in the context of numerical integration and simulation[47], the difficulties caused by stiffness for reachability computations do not appear to have been previously studied. Furthermore, we found that formal verification techniques can be made more powerful by combining static analysis with reachability analysis. For ex-

[1] In [83], the reduction technique was motivated by a need to reduce the dimensionality of the state space rather than our use of the reduction to avoid problems of stiffness.

ample, metastable behaviours cannot be analyzed solely by reachability analysis. Finally, the application of interval computation and APR numbers are essential for the robustness of COHO.

In summary, we have developed methods for systematically modeling circuits based on non-linear models in a way that captures the phenomena of state-of-the-art device models and is suitable for formal verification. We gave an extension of LTL for specifying circuit properties. We have made numerous improvements to COHO, introduced new analysis techniques, and proven the soundness of these techniques. We have demonstrated the efficacy of these techniques by verifying significant properties of real circuits from the literature. These results demonstrate the feasibility of formally verifying digital and analog circuit behaviors using projectagon based reachability analysis.

6.2 Future Research

While our research has demonstrated that properties of realistic circuits can be formally verified, this work also raises many questions and motivates many future research topics. To fully realize the potential of formal methods for analog models, we expect to make improvements to the reachability methods pioneered in COHO including supporting a wider range of device models, using parallel computation to speed-up the verification, and working on formal specification techniques that in turn should help to automate many aspects of the verification process. To verify larger circuits and a wider range of properties, we believe that reachability techniques should be complemented by other methods including small-signal analysis, static analysis, and parameterized verification. The reachability analysis techniques that we have developed could also be applied to a wide range of problems including control theory, biological systems, and hybrid systems as well. We describe each of these in greater detail below.

6.2.1 AMS Verification

To verify an AMS circuit, designers and verifiers need an expressive specification language that will allow them to communicate the essential properties of the circuit. Specifications formalize the correct circuit behaviors and provide a uniform

interface for CAD tools such that they can be compared or integrated with other tools. Unambiguous specifications are also a key prerequisite for design re-use. It is attractive to extend the specification method in this thesis to include properties that are commonly used by designers to describe analog circuits such as gain, frequency, and bandwidth. A key challenge here is that these are naturally described as frequency-domain properties, whereas formal verification tools have generally focused on time-domain based analysis. What are sound semantics for frequency-domain properties when the underlying circuit models are non-linear?

With clearly defined specifications, we should be able to automate much of the verification process for analog circuits. Currently, we check most of the properties to be verified by manually inspecting the reachable regions computed by COHO. If we have specifications with clear, mathematical interpretations, then it should be possible to automatically generate the reachability problems that COHO or another tool must solve, and then check the results using interval computation and linear programming techniques. Standard specifications are also helpful to tightly integrate simulation and verification. In particular, we could use optimal control methods to try to construct counter-example trajectories that correspond to verification failures. Conversely, if such a trajectory cannot be found, we could use the "gap" between the simulation trajectories and the computed reachable space to guide where additional computational effort should be invested to reduce the over-approximations of the reachability computation.

Our verification examples described in Chapter 5 show that non-linear reachability computations, static invariant computations, and small-signal linear analysis can be used as complementary techniques to build a verification framework that is much more powerful that the sum of its parts. For example, tools such as HYSAT [116] and HSOLVER [177] can be used to derive static constraints on the feasible regions of non-linear dynamical systems that could then be used by tools such as COHO to compute tighter bound on the dynamically reachable space. We envision producing a "satisfiability modulo non-linear dynamics" that could be applied to AMS circuits and other hybrid systems. Furthermore, the integration of these algorithms enable users to obtain a good trade-off between performance and accuracy during the verification. For example, reachability computation by COHO can solve nonlinear dynamics accurately but the computation is expensive. On

the other hand, static analysis by HYSAT or HSOLVER offers greater efficiency. We tried this idea in the verification of the Rambus oscillator and arbiter circuits. However, there remain many opportunities to further integrate these approaches.

Most analog circuits are examples of a small number (≤ 20) of basic types of cells, such as A/D (D/A) converters, amplifiers, oscillators, and phase-comparators. Therefore, it may be practical to develop *point verification* tools and specification techniques for the most commonly used types of analog components and structures. For example, oscillators, with two or three typical structures, are a good starting point.

A promising approach for verifying large AMS circuits is to take advantage of common circuit structures. It is common to have multiple stages or a large number of identical elements in an analog circuit. For example, ladder structures that implement a unary encoding of a control value are common in analog and mixed signal designs. The PLL design from [205] constructs the capacitors that set the resonant frequency by using hundreds of replicas of the same structure to ensure monotonic response to the feedback and control paths under PVT variations. *Parameterized verification* is a promising technique to verify large circuits with such structure. It has the potential to simplify the circuit model and make it practical to verify analog circuits with hundreds of nodes.

Another promising approach to verifying AMS circuits is to apply small signal analysis techniques to characterize (nearly) linear behaviors at the intended operating point and use reachability analysis to show large-scale convergence to this linear behavior. Although the semiconductor devices exhibit highly non-linear large-signal behaviors, nearly all analog circuits are designed to operate with nearly linear transfer functions when viewed from the appropriate domain [131]. In fact, designers usually describe the behavior of their circuits in terms of these linear, small-signal responses. However, analog designs can fail when unforeseen combinations of large-signal, non-linear behaviors prevent the circuit from reaching the intended operating point. Such failures can occur at start-up or during mode transitions. Reachability analysis can be applied to identify stable operating regions and show that a circuit has the intended global convergence properties. One example of this approach is the Rambus ring-oscillator described above. For the Rambus ring-oscillator, we noted that convergence can only be shown *almost surely*. We expect

that this will be the case for most analog circuits. Thus, key topics for future research include developing systematic ways of showing almost-surely convergence for a wide range of analog circuits, and developing model-reduction techniques – such as showing differential operation of the oscillator – that will make the reachability computations tractable for realistic circuits.

Designers frequently employ digital circuits to tune out the non-idealities of analog circuits, such as PVT variations. This motivates integrating digital formal verification with circuit-level verification. Techniques such as assume-guarantee [114] could be applied to first check individual analog and digital blocks and then prove properties of the whole circuit.

6.2.2 Improve Performance of COHO

COHO is quite slow, *e.g.*, it may take several days to complete the reachability computation for an analog circuit with more than six nodes. Roughly speaking, COHO has three main performance bottlenecks: 1) determining the error terms for linear differential inclusions, 2) projecting feasible regions of linear constraint systems onto projection planes, and 3) computing bounds on node voltages for device model evaluations. For example, approximately 30% of total runtime is spent on computing the bounding box of a projectagon (bottleneck 3). Each of these tasks offers abundant parallelism, thus the performance can be improved by parallel computation, such as GPGPUs and multi-threaded programming described below.

COHO has computations that are highly-data parallel, such as the problem of finding worst-case errors for linear differential inclusions (bottleneck 1). These computations are natural candidates for GPGPUs (general purpose GPUs). A GPU usually has hundreds of simple processors which are specialized for compute-intensive applications. Therefore, it can process data-parallel computation more efficiently than a CPU with high frequency. In the past, exploiting the parallelism offered by GPUs was complicated because a programmer needed to know the architecture and programming details of the particular GPU that they were using. Many GPUs now provide a general programming interface, *e.g.*, the CUDA [6] programming model supported by many Nvidia GPUs.

COHO could also take advantage of multi-threaded programming. The COHO implementation is partitioned into two components: the MATLAB process implements the basic reachability algorithms and the JAVA process implements computationally intensive geometric and linear programming operations. The MATLAB process can generate a large number of independent problems for all of the projectagon faces and put these problems into a pool. These problems include projecting full-dimensional polyhedra onto two-dimensional planes (bottleneck 2), computing the bounding box of a projectagon (bottleneck 3), and reachability computations on projectagon faces. Asynchronously, the JAVA process can create multiple threads and assign an idle thread to each problem. As the current COHO only uses one process, this approach can speed up computations significantly without vastly modifying the current implementation.

Currently, the computation of node voltage bounds (bottleneck 3) is done with by solving linear programs using CPLEX [5]. For this arrangement, the parallelism is limited by the number of CPLEX licenses available. Alternatively, we could solve these linear programs using our JAVA methods. While our JAVA code is not as fast as CPLEX, we can increase the throughput by multi-threaded programming as described above. At today's prices, processor nodes are sufficiently less expensive than CPLEX licenses that the parallel JAVA approach is more cost-effective. This gap will almost certainly continue to widen for the foreseeable future.

In addition to these three bottlenecks, COHO has many other operations where task and data level parallelism is readily available. For example, reachability computations in many automata states can be performed independently. We believe that COHO and verification problems are excellent candidates for parallel computation.

The computation in COHO could also be improved by developing more efficient algorithms. For example, the bounding box of a projectagon is currently computed by linear programming. It could also be obtained by applying the interval closure method (see Section 4.2.3) on projection polygons. An interval tree [46, Chapter 14.3] is an efficient data structure to implement the algorithm. The new algorithm could be implemented in the JAVA process and thus be parallelized. As another example, the current implementation generates a complete polygon for the projection of each time-advanced face. Most edges of this polygon are discarded when com-

puting the union of these "face polygons" to produce a projection polygon and in the simplification operations. It may be possible to modify the projection operation to only compute the edges that will be used by the next time-step.

6.2.3 Hybrid Systems and Others

Our reachability analysis algorithms and tools can be applied to other hybrid systems or biological systems that are modeled by ODEs. Unlike traditional simulation methods, the formal approach accounts for all system behaviors and can thereby guarantee correctness. This is important for security-critical systems, such as public transport systems. For example, the airplane collision problem has been studied in [195] and a helicopter control system has been studied in [71]. As COHO supports moderate-dimensional, nonlinear systems and computes accurate results, we believe our methods can be applied to these and more complicated systems.

Bibliography

[1] Spectre simulator from Cadence. → pages 11, 59

[2] Ultrasim simulator from Cadence. → pages 11

[3] *Formal Methods in Computer-Aided Design, 7th International Conference, FMCAD 2007, Austin, Texas, USA, November 11-14, 2007, Proceedings*. IEEE Computer Society, 2007. → pages 193

[4] *Proceedings of the 13th Asia South Pacific Design Automation Conference, ASP-DAC 2008, Seoul, Korea, January 21-24, 2008*. IEEE, 2008. → pages 184, 193

[5] CPLEX 12.1 user's manual, 2009. → pages 118, 170

[6] NVIDIA CUDA programming guide, 2010. Version 3.2. → pages 169

[7] The source of Intel's Cougar Point SATA bug. AnandTech, January 2011. → pages 2

[8] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425. IEEE Computer Society, 1990. → pages 21, 29

[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(1):3 – 34, 1995. Hybrid Systems. → pages 18, 24

[10] Rajeev Alur, Thao Dang, and Franjo Ivancic. Counterexample-guided predicate abstraction of hybrid systems. *Theor. Comput. Sci.*, 354(2):250–271, 2006. → pages 33, 35, 36

[11] Rajeev Alur, Thao Dang, and Franjo Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. Embedded Comput. Syst.*, 5(1):152–199, 2006. → pages 32, 33, 36

[12] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994. → pages 12, 18, 19

[13] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996. → pages 21

[14] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Logic in Computer Science, Symposium on*, page 207, 1996. → pages 35

[15] Rajeev Alur and Thomas A. Henzinger. Modularity for timed and hybrid systems. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 1997. → pages 33

[16] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22:181–201, 1996. → pages 5, 21, 34, 35, 61

[17] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002. → pages 35

[18] Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. MOCHA: Modularity in model checking. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 521–525, London, UK, 1998. Springer-Verlag. → pages 15, 34, 35

[19] Rajeev Alur and George J. Pappas, editors. *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings*, volume 2993 of *Lecture Notes in Computer Science*. Springer, 2004. → pages 184, 189, 191

[20] Tobias Amnell, Gerd Behrmann, Johan Bengtsson, Pedro R. D'Argenio, Alexandre David, Ansgar Fehnker, Thomas Hune, Bertrand Jeannet, Kim Guldstrand Larsen, M. Oliver Möller, Paul Pettersson, Carsten Weise, and Wang Yi. UPPAAL - now, next, and future. In Franck Cassez, Claude Jard, Brigitte Rozoy, and Mark Dermot Ryan, editors, *MOVEP*, volume 2067 of *Lecture Notes in Computer Science*, pages 99–124. Springer, 2000. → pages 15, 27, 34, 35

[21] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002. → pages 22

[22] Eugene Asarin, Thao Dang, Goran Frehse, Antoine Girard, Colas Le Guernic, and Oded Maler. Recent progress in continuous and hybrid reachability analysis. In *In Proc. IEEE International Symposium on Computer-Aided Control Systems Design. IEEE Computer*. Society Press, 2006. → pages 18

[23] Eugene Asarin, Thao Dang, and Antoine Girard. Reachability analysis of nonlinear systems using conservative approximation. In Oded Maler and Amir Pnueli, editors, *HSCC*, volume 2623 of *Lecture Notes in Computer Science*, pages 20–35. Springer-Verlag, 2003. → pages 29, 31, 35, 36

[24] Eugene Asarin, Thao Dang, and Antoine Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Inf.*, 43(7):451–476, 2007. → pages 30, 31, 32, 35, 36, 44, 121

[25] Eugene Asarin, Thao Dang, and Oded Maler. d/dt: A tool for reachability analysis of continuous and hybrid systems. In *5th IFAC Symposium Nonlinear Control Systems (NOLCOS) , 2001. [ACH + 95*, pages 3–34, 2001. → pages 19, 35

[26] P. M. Aziz, H. V. Sorensen, and J. vn der Spiegel. An overview of sigma-delta converters. *Signal Processing Magazine, IEEE*, 13(1):61 –84, January 1996. → pages 37

[27] Roberto Bagnara, Elisa Ricci, Enea Zaffanella, and Patricia M. Hill. Possibly not closed convex polyhedra and the parma polyhedra library. In *SAS '02: Proceedings of the 9th International Symposium on Static Analysis*, pages 213–229, London, UK, 2002. Springer-Verlag. → pages 24

[28] Erich Barke, Darius Grabowski, Helmut Graeb, Lars Hedrich, Stefan Heinen, Ralf Popp, Sebastian Steinhorst, and Yifan Wang. Formal approaches to analog circuit verification. In *DATE*, pages 724–729. IEEE, 2009. → pages 18

[29] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on Uppaal. In Marco Bernardo and Flavio Corradini, editors, *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004. → pages 34, 35

[30] V. Bertin, Etienne Closse, M. Poize, Jacques Pulou, Joseph Sifakis, P. Venier, Daniel Weil, and Sergio Yovine. TAXYS=esterel+kronos. a tool for verifying real-time properties of embedded systems. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, volume 3, pages 2875–2880 vol.3, 2001. → pages 15

[31] Oleg Botchkarev and Stavros Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *HSCC '00: Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control*, pages 73–88, London, UK, 2000. Springer-Verlag. → pages 16, 27, 30, 35, 36

[32] Ahmed Bouajjani, Stavros Tripakis, and Sergio Yovine. On-the-fly symbolic model checking for real-time systems. In *IEEE Real-Time Systems Symposium*, pages 25–. IEEE Computer Society, 1997. → pages 29, 35

[33] Olivier Bournez, Oded Maler, and Amir Pnueli. Orthogonal polyhedra: Representation and computation. In *Schuppen (Eds.), Hybrid Systems: Computation and Control, LNCS 1569*, pages 46–60. Springer, 1999. → pages 25, 35, 120

[34] Marius Bozga, Hou Jianmin, Oded Maler, and Sergio Yovine. Verification of asynchronous circuits using timed automata. *Electronic Notes in Theoretical Computer Science*, 65(6):47 – 59, 2002. Theory and Practice of Timed Systems (Satellite Event of ETAPS 2002). → pages 38

[35] Marius Bozga, Oded Maler, Amir Pnueli, and Sergio Yovine. Some progress in the symbolic verification of timed automata. In Orna Grumberg, editor, *CAV*, volume 1254 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 1997. → pages 38

[36] M. S. Branicky. Multiple lyapunov functions and other analysis tools for switched and hybrid systems. *Automatic Control, IEEE Transactions on*, 43(4):475 –482, apr 1998. → pages 32

[37] R. W. Brockett. Smooth dynamical systems which realize arithmetical and logical operations. In Hendrik Nijmeijer and Johannes M. Schumacher, editors, *Three Decades of Mathematical Systems Theory: A Collection of Surveys at the Occasion of the 50th Birthday of J. C. Willems*, volume 135 of *Lecture Notes in Control and Information Sciences*, pages 19–30. sv, 1989. → pages 66

[38] T. J. Chaney and Charles E. Molnar. Anomalous behavior of synchronizer and arbiter circuits. *IEEETC*, C-22(4):421–422, April 1973. → pages 54, 146

[39] N. V. Chernikov. Algorithms for discovering the set of all solutions of a linear programming problem. *Computational Mathematics and Mathematical Physics*, pages 283–293, 1968. → pages 24

[40] Alongkrit Chutinan. *Hybrid System Verification Using Discrete Model Approximations*. PhD thesis, Carnegie Mellon University, 1999. → pages 15, 21, 32, 35, 36, 120

[41] Alongkrit Chutinan and Bruce H. Krogh. Verification of infinite-state dynamic systems using approximate quotient transition systems. *Automatic Control, IEEE Transactions on*, 46(9):1401–1410, September 2001. → pages 19, 25, 33, 35

[42] Alongkrit Chutinan and Bruce H. Krogh. Computational techniques for hybrid system verification. *Automatic Control, IEEE Transactions on*, 48(1):64–75, January 2003. → pages 32, 35

[43] Edmund M. Clarke, Alexandre Donzé, and Axel Legay. Statistical model checking of mixed-analog circuits with an application to a third order delta-sigma modulator. In Hana Chockler and Alan J. Hu, editors, *Haifa Verification Conference*, volume 5394 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2008. → pages 12, 38, 39

[44] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems, 2003. → pages 33

[45] Etienne Closse, Michel Poize, Jacques Pulou, Joseph Sifakis, Patrick Venter, Daniel Weil, and Sergio Yovine. TAXYS: A tool for the development and verification of real-time embedded systems. In *CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification*, pages 391–395, London, UK, 2001. Springer-Verlag. → pages 15

[46] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. → pages 170

[47] Germund G. Dahlquist. A special stability problem for linear multistep methods. *BIT Numerical Mathematics*, 3:27–43, 1963. 10.1007/BF01963532. → pages 165

[48] Thao Dang. *Verification and Synthesis of Hybrid Systems*. PhD thesis, Institut National Polytechnique de Grenoble, 2000. → pages 15, 36, 120

[49] Thao Dang. Approximate reachability computation for polynomial systems. In Hespanha and Tiwari [117], pages 138–152. → pages 32

[50] Thao Dang, Alexandre Donzé, and Oded Maler. Verification of analog and mixed-signal circuits using hybrid system techniques. In Alan J. Hu and Andrew K. Martin, editors, *FMCAD*, volume 3312 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2004. → pages 5, 28, 38, 39, 44

[51] Thao Dang, Colas Le Guernic, and Oded Maler. Computing reachable states for nonlinear biological models. In Pierpaolo Degano and Roberto Gorrieri, editors, *CMSB*, volume 5688 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 2009. → pages 31

[52] Thao Dang and Oded Maler. Reachability analysis via face lifting. In Thomas A. Henzinger and Shankar Sastry, editors, *HSCC*, volume 1386 of *Lecture Notes in Computer Science*, pages 96–109, London, UK, 1998. Springer-Verlag. → pages 31, 35

[53] Thao Dang, Oded Maler, and Romain Testylier. Accurate hybridization of nonlinear systems. In Karl Henrik Johansson and Wang Yi, editors, *HSCC*, pages 11–20. ACM ACM, 2010. → pages 31

[54] Thao Dang and Tarik Nahhal. Randomized simulation of hybrid systems for circuit validation. In *FDL*, pages 9–15. ECSI, 2006. → pages 11

[55] Thao Dang and Romain Testylier. Hybridization domain construction using curvature estimation. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, HSCC '11, pages 123–132, New York, NY, USA, 2011. ACM. → pages 31

[56] Tathagato Rai Dastidar and P. P. Chakrabarti. A verification system for transient response of analog circuits using model checking. In *VLSI Design*, pages 195–200. IEEE Computer Society, 2005. → pages 13, 21, 38, 42

[57] C. Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool KRONOS. In *Proceedings of the DIMACS/SYCON workshop on Hybrid*

systems III : verification and control, pages 208–219, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc. → pages 19, 34, 35

[58] William Denman, Har'El Z., and Ivan Sutherland. A bond graph approach for the constraint based verification of analog circuits. *Proc. Workshop on Formal Verification of Analog Circuit (FAC08)*, pages pp. 1–28., July 2008. → pages 20

[59] Zhi Jie Dong, Mohamed H. Zaki, Ghiath Al Sammane, Sofiène Tahar, and Guy Bois. Checking properties of pll designs using run-time verification. *Proc. IEEE International Conference on Microelectronics (ICM'07)*, 2007. → pages 12

[60] Zhi Jie Dong, Mohamed H. Zaki, Ghiath Al Sammane, Sofiène Tahar, and Guy Bois. Run-time verification using the VHDL-AMS simulation environment. *Proc. IEEE Northeast Workshop on Circuits and Systems (NEWCAS'07)*, 2007. → pages 12, 38

[61] Magnus Egerstedt and Bud Mishra, editors. *Hybrid Systems: Computation and Control, 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings*, volume 4981 of *Lecture Notes in Computer Science*. Springer, 2008. → pages 180, 189

[62] Andreas Eggers, Martin Fränzle, and Christian Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In Sung Deok Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *ATVA*, volume 5311 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2008. → pages 31, 35, 36

[63] E. Allen Emerson, Aloysius K. Mok, A. Prasad Sistla, and Jai Srinivasan. Quantitative temporal reasoning. *Real-Time Systems*, 4(4):331–352, 1992. → pages 21

[64] Harry Foster, Erich Marschner, and Yaron Wolfsthal. IEEE 1850 PSL: The next generation, 2005. → pages 20

[65] Martin Fränzle and Christian Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007. → pages 31, 35

[66] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007. → pages 25, 31, 35, 36

[67] Goran Frehse. Compositional verification of hybrid systems with discrete interaction using simulation relations. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 59–64, September 2004. → pages 33, 35

[68] Goran Frehse. *Compositional Verification of Hybrid Systems Using Simulation Relations*. PhD thesis, Radboud Universiteit Nijmegen, October 2005. → pages 34

[69] Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. pages 258–273. Springer, 2005. → pages 34, 152

[70] Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. *Int. J. Softw. Tools Technol. Transf.*, 10(3):263–279, 2008. → pages 15, 19, 21, 31, 34, 35, 120, 121

[71] Goran Frehse, Scott Cotton, Rajarshi Ray, Alexandre Donzé, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Colas Le Guernic, and Oded Maler. Spaceex: Scalable verification of hybrid systems. 2011. submitted. → pages 28, 171

[72] Goran Frehse, Zhi Han, and Bruce H. Krogh. Assume-guarantee reasoning for hybrid i/o-automata by over-approximation of continuous interaction. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, pages 479–484 Vol.1, December 2004. → pages 33, 34, 35

[73] Goran Frehse, Bruce H. Krogh, and Rob A. Rutenbar. Verifying analog oscillator circuits using forward/backward abstraction refinement. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 257–262, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association. → pages 5, 29, 33, 35, 38, 40, 41, 42

[74] Goran Frehse, Bruce H. Krogh, Rob A. Rutenbar, and Oded Maler. Time domain verification of oscillator circuit properties. *Electr. Notes Theor. Comput. Sci.*, 153(3):9–22, 2006. → pages 19, 22, 35

[75] Goran Frehse and Rajarshi Ray. Design principles for an extendable verification tool for hybrid systems. In *ADHS'09*, volume 3, part 1, 2009. → pages 16, 35, 36

[76] Abhijit Ghosh and Ranga Vemuri. Formal verification of synthesized analog designs. *Computer Design, International Conference on*, 0:40, 1999. → pages 16, 38

[77] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In Morari and Thiele [162], pages 291–305. → pages 16, 26, 30, 35

[78] Antoine Girard and Colas Le Guernic. Zonotope/hyperplane intersection for hybrid systems reachability analysis. In Egerstedt and Mishra [61], pages 215–228. → pages 26, 27

[79] Antoine Girard, Colas Le Guernic, and Oded Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In Hespanha and Tiwari [117], pages 257–271. → pages 16, 28, 30, 35

[80] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. → pages 100, 102

[81] Darius Grabowski, Daniel Platte, Lars Hedrich, and Erich Barke. Time constrained verification of analog circuits using model-checking algorithms. *Electr. Notes Theor. Comput. Sci.*, 153(3):37–52, 2006. → pages 14, 21

[82] Mark R. Greenstreet. *STARI: A Technique for High-Bandwidth Communication*. PhD thesis, Princeton University, 1993. → pages 3

[83] Mark R. Greenstreet. Verifying safety properties of differential equations. In *Proceedings of the 1996 Conference on Computer Aided Verification*, pages 277–287, New Brunswick, NJ, July 1996. → pages 45, 126, 127, 165

[84] Mark R. Greenstreet and Peter Cahoon. How fast will the flip flop? In *Proceedings of the First International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 77–86, Salt Lake City, November 1994. IEEE Computer Society Press. → pages

[85] Mark R. Greenstreet and Xuemei Huang. A smooth dynamical system that counts in binary. In *Proceedings of the 1997 International Conference on Circuits and Systems*, volume II, pages 977–980, Hong Kong, June 1997. IEEE. → pages 45

[86] Mark R. Greenstreet and Ian Mitchell. Integrating projections. In *HSCC '98: Proceedings of the First International Workshop on Hybrid Systems*, pages 159–174. Springer Verlag, 1998. → pages 45, 163

[87] Mark R. Greenstreet and Ian Mitchell. Reachability analysis using polygonal projections. In *HSCC '99: Proceedings of the Second*

International Workshop on Hybrid Systems, pages 103–116, London, UK, 1999. Springer-Verlag. → pages 45, 163, 164

[88] Mark R. Greenstreet and Suwen Yang. Verifying start-up conditions for a ring oscillator. In *GLSVLSI '08: Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pages 201–206, New York, NY, USA, 2008. ACM. → pages 149, 150, 160

[89] Orna Grumberg and David E. Long. Model checking and modular verification. In Jos C. M. Baeten and Jan Friso Groote, editors, *CONCUR*, volume 527 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 1991. → pages 21

[90] Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 540–554. Springer, 2009. → pages 27

[91] Smriti Gupta, Bruce H. Krogh, and Rob A. Rutenbar. Towards formal verification of analog designs. In *Proceedings of 2004 IEEE/ACM International Conference on Computer Aided Design*, pages 210–217, November 2004. → pages 38, 39, 40

[92] Nicolas Halbwachs. Delay analysis in synchronous programs. In *CAV '93: Proceedings of the 5th International Conference on Computer Aided Verification*, pages 333–346, London, UK, 1993. Springer-Verlag. → pages 24

[93] Nicolas Halbwachs, Pascal Raymond, and Yann-eric Proy. Verification of linear hybrid systems by means of convex approximations. In *SAS*, pages 223–237. Springer-Verlag, 1994. → pages 24

[94] Keith Hanna. Automatic verification of mixed-level logic circuits. In Ganesh Gopalakrishnan and Phillip Windley, editors, *Formal Methods in Computer-Aided Design*, volume 1522 of *Lecture Notes in Computer Science*, pages 530–530. Springer Berlin / Heidelberg, 1998. → pages 16

[95] Keith Hanna. Reasoning about analog-level implementations of digital systems. In *Formal Methods in System Design*, volume 16, pages 127–158. Springer Netherlands, 2000. → pages 16, 37, 38

[96] R. H. Hardin, Z. Har'EI, and Robert P. Kurshan. COSPAN. *Lecture Notes in Computer Science, Computer Aided Verification*, 1102/1996:423–427, 1996. → pages 13

[97] Walter Hartong, Lars Hedrich, and Erich Barke. Model checking algorithms for analog verification. In *DAC '02: Proceedings of the 39th annual Design Automation Conference*, pages 542–547, New York, NY, USA, 2002. ACM. → pages 5, 13, 38, 40, 43, 152

[98] Walter Hartong, Lars Hedrich, and Erich Barke. On discrete modeling and model checking for nonlinear analog systems. In *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pages 401–413, London, UK, 2002. Springer-Verlag. → pages 13, 21, 43

[99] Thomas A. Henzinger. Hybrid automata with finite bisimulatioins. In *ICALP '95: Proceedings of the 22nd International Colloquium on Automata, Languages and Programming*, pages 324–335, London, UK, 1995. Springer-Verlag. → pages 33

[100] Thomas A. Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, volume Vol.170, pages 265–292. Springer, 2000. → pages 19, 21, 35

[101] Thomas A. Henzinger and Pei-Hsin Ho. HyTech: The Cornell HYbrid TECHnology tool. In *Hybrid Systems II, LNCS 999*, pages 265–293. Springer-Verlag, 1995. → pages 34

[102] Thomas A. Henzinger and Pei-Hsin Ho. A note on abstract-interpretation strategies for hybrid automata. In *Hybrid Systems II, volume 999 of LNCS*, pages 252–264. Springer-Verlag, 1995. → pages 24, 35

[103] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-toi. HyTech: The next generation. In *In Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 56–65. IEEE Computer Society press, 1995. → pages 34, 35

[104] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-toi. A user guide to HyTech, 1995. → pages 15

[105] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:225–238, 1996. → pages 18, 19, 31, 35, 153

[106] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463, 1997. → pages 34

[107] Thomas A. Henzinger, Benjamin Horowitz, Rupak Majumdar, and Howard Wong-toi. Beyond HyTech: Hybrid systems analysis using interval numerical methods. In *in HSCC*, pages 130–144. Springer, 2000. → pages 15, 25, 31, 34, 35

[108] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Lazy abstraction. In *POPL*, pages 58–70, 2002. → pages 33

[109] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Journal of Computer and System Sciences*, pages 373–382. ACM Press, 1995. → pages 12, 15, 19

[110] Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 514–529. Springer, 1996. → pages 29

[111] Thomas A. Henzinger, Marius Minea, and Vinayak S. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *HSCC*, volume 2034 of *Lecture Notes in Computer Science*, pages 275–290. Springer, 2001. → pages 33

[112] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994. → pages 29

[113] Thomas A. Henzinger, Jorg Preu, and Howard Wong-toi. Some lessons from the HyTech experience. In *In Proceedings of the 40th Annual Conference on Decision and Control*, pages 2887–2892. IEEE Press, 2001. → pages 34

[114] Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. You assume, we guarantee: Methodology and case studies. In Alan J. Hu and Moshe Y. Vardi, editors, *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 1998. → pages 33, 35, 70, 169

[115] Thomas A. Henzinger, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. An assume-guarantee rule for checking simulation. *ACM Trans. Program. Lang. Syst.*, 24(1):51–64, 2002. → pages 33

[116] Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige. Analysis of hybrid systems using HySAT. In *ICONS '08: Proceedings of the Third International Conference on Systems*, pages 196–201, Washington, DC, USA, 2008. IEEE Computer Society. → pages 16, 35, 36, 167

[117] João P. Hespanha and Ashish Tiwari, editors. *Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, March 29-31, 2006, Proceedings*, volume 3927 of *Lecture Notes in Computer Science*. Springer, 2006. → pages 177, 180

[118] Timothy J. Hickey. Analytic constraint solving and interval arithmetic. In *POPL*, pages 338–351, 2000. → pages 31

[119] Timothy J. Hickey and David K. Wittenberg. Rigorous modeling of hybrid systems using interval arithmetic constraints. In Alur and Pappas [19], pages 402–416. → pages 31

[120] G. Hinton, M. Upton, D. J. Sager, D. Boggs, D. M. Carmean, P. Roussel, T. I. Chappell, T. D. Fletcher, M. S. Milshtein, M. Sprague, S. Samaan, and R. Murray. A $0.18 - \mu$ CMOS IA-32 processor with a 4-GHz integer execution unit. *Solid-State Circuits, IEEE Journal of*, 36(11):1617 –1627, November 2001. → pages 5

[121] Morris W. Hirsch and Stephen Smale. *Differential Equations, Dynamical Systems, and Linear Algebra*. Academic Press, 1974. → pages 151

[122] Pei-Hsin Ho. *Automatic Analysis of Hybrid Systems*. PhD thesis, Cornell University, Ithaca, NY, USA, August 1995. → pages 29

[123] David A. Hodges, Horace G. Jackson, and Resve A. Saleh. *Analysis and Design of Digital Integrated Circuits in Deep Submicron Technology*. McGraw Hill, 2004. → pages 59

[124] Marco Hurtado. *Structure and Performance of Asymptotically Bistable Dynamical Systems*. PhD thesis, Sever Institute, Washington University, Saint Louis, MO, 1975. → pages 146

[125] Alexander Jesser. *Mixed-Signal Circuit Verification Using Symbolic Model Checking Techniques*. PhD thesis, University of Frankfurt a.M., Germany, ISBN 978-3-89963-841-7, October 2008. → pages 21, 38

[126] Alexander Jesser and Lars Hedrich. A symbolic approach for mixed-signal model checking. In *ASP-DAC* [4], pages 404–409. → pages 14, 27

[127] Er Jesser, Stefan Lämmermann, Er Pacholik, Lars Hedrich, Jürgen Ruf, Thomas Kropf, Wolfgang Fengler, and Wolfgang Rosenstiel. Analog simulation meets digital verification a formal assertion approach for mixed-signal verification, 2008. → pages 38

[128] M. Johansson and A. Rantzer. Computation of piecewise quadratic lyapunov functions for hybrid systems. *Automatic Control, IEEE Transactions on*, 43(4):555 –559, apr 1998. → pages 32

[129] Kevin D. Jones, Jaeha Kim, and Victor Konrad. Some "real world" problems in the analog and mixed-signal domains. In *Proc. Workshop on Designing Correct Circuits*, April 2008. → pages 3, 9, 149, 153

[130] Kevin D. Jones, Victor Konrad, and Dejan Nickovic. Analog property checkers: a DDR2 case study. *Form. Methods Syst. Des.*, 36(2):114–130, 2010. → pages 37, 38

[131] Jaeha Kim, M. Jeeradit, Byongchan Lim, and M. A. Horowitz. Leveraging designer's intent: A path toward simpler analog CAD tools. In *Custom Integrated Circuits Conference, 2009. CICC '09. IEEE*, pages 613 –620, September 2009. → pages 168

[132] A. Korshak. Noise-rejection model based on charge-transfer equation for digital CMOS circuits. *IEEE Transactions on Computer Aided Design*, 23(10):1460–1465, October 2004. → pages 65

[133] Robert P. Kurshan and Kenneth L. McMillan. Analysis of digital circuits through symbolic reduction. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 10(11):1356–1371, November 1991. → pages 13, 20, 38, 136

[134] Alexander B. Kurzhanski and Istvan Valyi. *Ellipsoidal Calculus for Estimation and Control*. Birkhäuser Boston, 1 edition edition, September 1996. → pages 26, 35

[135] Alex A. Kurzhanskiy and Pravin Varaiya. Ellipsoidal toolbox. Technical Report UCB/EECS-2006-46, EECS Department, University of California, Berkeley, May 2006. → pages 27

[136] Gerardo Lafferriere, George J. Pappas, and Shankar Sastry. Hybrid systems with finite bisimulations. In *Hybrid Systems V*, pages 186–203, London, UK, 1999. Springer-Verlag. → pages 33

[137] Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. Decidable hybrid systems. Technical report, Department of Mathematical Science, Portland State University, Portland, OR, 1998. → pages 12, 19

[138] Stefan Lämmermann, Jürgen Ruf, Thomas Kropf, Wolfgang Rosenstiel, Alexander Viehl, Alexander Jesser, and Lars Hedrich. Towards assertion-based verification of heterogeneous system designs. In *DATE*, pages 1171–1176. IEEE, 2010. → pages 12, 22

[139] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Compositional and symbolic model-checking of real-time systems. In *RTSS '95: Proceedings of the 16th IEEE Real-Time Systems Symposium*, page 76, Washington, DC, USA, 1995. IEEE Computer Society. → pages 29

[140] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997. → pages 19, 35

[141] P. Larsson and C. Svensson. Noise in dynamic CMOS circuits. *IEEE Journal of Solid State Circuits*, 29(6):655–662, June 1994. → pages 65

[142] Marius Laza. A robust linear program solver for projectahedra. Master's thesis, Univerisity of British Columbia, 2001. → pages 46

[143] H. Le Verge. A note on chernikov's algorithm. Technical report, IRISA, 1992. → pages 24

[144] Scott Little. *Efficient Modeling and Verification of Analog/Mixed-Signal Circuits Using Labeled Hybrid Petri Nets*. PhD thesis, University of Utah, 2008. → pages 15, 34, 35, 38, 40

[145] Scott Little and Chris J. Myers. Abstract modeling and simulation aided verification of analog/mixed-signal circuits. *Formal Verification of Analog Circuits (FAC) '08*, 2008. → pages 20

[146] Scott Little, Nicholas Seegmiller, David Walter, Chris J. Myers, and Tomohiro Yoneda. Verification of analog/mixed-signal circuits using labeled hybrid Petri nets. In *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 275–282, New York, NY, USA, 2006. ACM. → pages 19, 27, 35

[147] Scott Little and David Walter. Verification of analog and mixed-signal circuits using timed hybrid Petri nets. *Automated Technology for Verification and Analysis*, 3299 of LNCS:426–440, November 2004. → pages 19, 35

[148] Scott Little, David Walter, Kevin D. Jones, and Chris J. Myers. Analog/mixed-signal circuit verification using models generated from simulation traces. *Automated Technology for Verification and Analysis, Lecture Notes in Computer Science*, 4762:114–128, 2007. Springer, Berlin. → pages 31, 35, 60, 121

[149] John Lygeros. Lecture notes on hybrid systems, January 2003. Department of Engineering, University of Cambridge. → pages 19

[150] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine, editors, *FORMATS/FTRTFT*, volume 3253 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2004. → pages 12, 21, 61

[151] Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 475–505. Springer, 2008. → pages 12

[152] Oded Maler and Amir Pnueli. Extending PSL for analog circuits. Technical report, PROSYD: Property-Based System Design, 2005. → pages 21

[153] Oded Maler and Sergio Yovine. Hardware timing verification using KRONOS. *Israeli Conference on Computer-Based Systems and Software Engineering*, 0:23, 1996. → pages 38

[154] L. R. Marino. General theory of metastable operation. *IEEETC*, C-30(2):107–115, February 1981. → pages 54, 71, 146

[155] Alain J. Martin. Programming in VLSI: From communicating processes to delay insensitive circuits. In C. A. R. Hoare, editor, *University of Texas Year of Programming Institute on Concurrent Programming*. Addison-Wesley, 1989. → pages 53

[156] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992. → pages 100

[157] Michael Mendler and Terry Stroup. Newtonian arbiters cannot be proven correct. In *Proceedings of the 1992 Workshop on Designing Correct Circuits*, January 1992. → pages 54, 71

[158] Ian Mitchell. Comparing forward and backward reachability as tools for safety analysis. In Alberto Bemporad, Antonio Bicchi, and Giorgio C. Buttazzo, editors, *HSCC*, volume 4416 of *Lecture Notes in Computer Science*, pages 428–443. Springer, 2007. → pages 87

[159] Ian Mitchell. The flexible, extensible and efficient toolbox of level set methods. *J. Sci. Comput.*, 35(2-3):300–329, 2008. → pages 16, 36

[160] Ian Mitchell and Mark R. Greenstreet. Proving Newtonian arbiters correct, almost surely. In *Proceedings of the Third Workshop on Designing Correct Circuits*, Båstad, Sweden, September 1996. → pages 9, 71, 146, 148, 152

[161] Ian Mitchell and Claire J. Tomlin. Level set methods for computation in hybrid systems. In *HSCC '00: Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control*, pages 310–323, London, UK, 2000. Springer-Verlag. → pages 27, 31, 35, 36

[162] Manfred Morari and Lothar Thiele, editors. *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, volume 3414 of *Lecture Notes in Computer Science*. Springer, 2005. → pages 180, 189

[163] Dejan Nickovic and Oded Maler. AMT: A property-based monitoring tool for analog systems. In Jean-françois Raskin and P. S. Thiagarajan, editors, *FORMATS*, volume 4763 of *Lecture Notes in Computer Science*, pages 304–319. Springer, 2007. → pages 12, 22, 38

[164] J. Nocedal and S. Wright. *Numerical Optimization*, pages 395–417. Springer Series in Operations Research, Springer Press, 1999. → pages 100

[165] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982. → pages 100, 101

[166] Thomas S. Parker and Leon O. Chua. *Practical Numerical Algorithms for Chaotic Systems*. sv, New York, 1989. → pages 127

[167] W. Kopke Peter. *The Theory of Rectangular Hybrid Automata*. PhD thesis, Cornell University, August 1996. → pages 26

[168] Amir Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981. → pages 61

[169] David Pollard. *A User's Guide to Measure Theoretic Probability*. Cambridge University Press, 2001. → pages 146, 151

[170] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In Alur and Pappas [19], pages 477–492. → pages 32

[171] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Texts and Monographs in Computer Science. Springer, 1985. → pages 93, 198

[172] Joerg Preussig, Stephan Kowalewski, Howard Wong-toi, and Thomas A. Henzinger. An algorithm for the approximative analysis of rectangular automata. In Anders P. Ravn and Hans Rischel, editors, *FTRTFT*, volume 1486 of *Lecture Notes in Computer Science 1486*, pages 228–240. Springer, 1998. → pages 26

[173] Anuj Puri and Pravin Varaiya. Decidability of hybrid systems with rectangular differential inclusion. In *CAV '94: Proceedings of the 6th International Conference on Computer Aided Verification*, pages 95–104, London, UK, 1994. Springer-Verlag. → pages 12, 19

[174] Nacim Ramdani, Nacim Meslem, and Yves Candau. Reachability of uncertain nonlinear systems using a nonlinear hybridization. In Egerstedt and Mishra [61], pages 415–428. → pages 31

[175] Stefan Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Trans. Comput. Log.*, 7(4):723–748, 2006. → pages 36

[176] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In Morari and Thiele [162], pages 573–589. → pages 35, 36

[177] Stefan Ratschan and Zhikun She. Constraints for continuous reachability in the verification of hybrid systems. In Jacques Calmet, Tetsuo Ida, and Dongming Wang, editors, *AISC*, volume 4120 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2006. → pages 16, 31, 32, 35, 36, 167

[178] A. Salem. Semi-formal verification of VHDL-AMS descriptions. In *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, volume 5, pages V–333–V–336 vol.5, 2002. → pages 12

[179] Ghiath Al Sammane, Mohamed H. Zaki, Zhi Jie Dong, and Sofiène Tahar. Towards assertion based verification of analog and mixed signal designs using PSL. In *FDL*, pages 293–298. ECSI, 2007. → pages 12, 21, 38, 39

[180] Ghiath Al Sammane, Mohamed H. Zaki, and Sofiène Tahar. A symbolic methodology for the verification of analog and mixed signal designs. In Rudy Lauwereins and Jan Madsen, editors, *DATE*, pages 249–254. ACM, 2007. → pages 16, 38, 39

[181] Sriram Sankaranarayanan, Thao Dang, and Franjo Ivancic. Symbolic model checking of hybrid systems using template polyhedra. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2008. → pages 25, 121

[182] Carl-Johan H. Seger, Robert B. Jones, John W. O'Leary, Thomas F. Melham, Mark Aagaard, Clark Barrett, and Don Syme. An industrially effective environment for formal hardware verification. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(9):1381–1405, 2005. → pages 4

[183] Charles L. Seitz. System timing. In *Introduction to VLSI Systems (Carver Mead and Lynn Conway)*, chapter 7, pages 218–262. Addison Wesley, 1979. → pages 54, 136, 146

[184] B. Izaias Silva, K. Richeson, Bruce H. Krogh, and Alongkrit Chutinan. Modeling and verifying hybrid dynamical systems using CheckMate. In *Proceedings of the* 4^{th} *International Conference on Automation of Mixed Processes (ADPM 2000)*, pages 323–328, September 2000. → pages 36

[185] B. Izaias Silva, Olaf Stursberg, Bruce H. Krogh, and S. Engell. An assessment of the current status of algorithmic approaches to the verification of hybrid systems. *40th Conference on Decision and Control*, December 2001. → pages 18

[186] O. Stauning. *Automatic Validation of Numerical Solutions*. PhD thesis, Danmarks Tekniske Universitet, Kgs., Lyngby, Denmark, 1997. → pages 31

[187] Sebastian Steinhorst and Lars Hedrich. Model checking of analog systems using an analog specification language. In *DATE*, pages 324–329, New York, NY, USA, 2008. ACM. → pages 14, 22, 38, 42, 43

[188] Sebastian Steinhorst and Lars Hedrich. Advanced methods for equivalence checking of analog circuits with strong nonlinearities. *Form. Methods Syst. Des.*, 36(2):131–147, 2010. → pages 12, 36, 38

[189] Sebastian Steinhorst, Er Jesser, and Lars Hedrich. Advanced property specification for model checking of analog systems. In *ANALOG06*, 2006. → pages 22, 38

[190] Olaf Stursberg, Ansgar Fehnker, Zhi Han, and Bruce H. Krogh. Specification-guided analysis of hybrid systems using a hierarchy of validation methods. In *In Proc. IFAC Conference ADHS*. Elsevier, 2003. → pages 33, 35

[191] Olaf Stursberg and Bruce H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In *In HSCC2003, LNCS 2289*, pages 482–497. Springer, 2003. → pages 26

[192] Ashish Tiwari and Gaurav Khanna. Nonlinear systems: Approximating reach sets. In Alur and Pappas [19], pages 600–614. → pages 32

[193] Saurabh K. Tiwary, Anubhav Gupta, Joel R. Phillips, Claudio Pinello, and Radu Zlatanovici. First steps towards SAT-based formal analog verification. In *ICCAD '09: Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 1–8, New York, NY, USA, 2009. ACM. → pages 12, 20, 37, 38, 42

[194] Claire J. Tomlin, Ian Mitchell, Alexandre M. Bayen, Re M. Bayen, and Meeko Oishi. Computational techniques for the verification and control of hybrid systems. In *Proceedings of the IEEE*, pages 986–1001, 2003. → pages 19, 35

[195] Claire J. Tomlin, Ian Mitchell, Alexandre M. Bayen, and Meeko Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, 2003. → pages 171

[196] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, second edition edition, 2001. → pages 101, 102

[197] Pravin Varaiya. Reach set computation using optimal control. In *Proc. KIT Workshop*, pages 377–383, 1998. → pages 29, 30, 35, 96

[198] David Walter, Scott Little, and Chris J. Myers. Bounded model checking of analog and mixed-signal circuits using an SMT solver. *Automated*

Technology for Verification and Analysis, Lecture Notes in Computer Science, 4762:66–81, 2007. Spring, Berlin. → pages 29

[199] David Walter, Scott Little, Nicholas Seegmiller, Chris J. Myers, and Tomohiro Yoneda. Symbolic model checking of analog/mixed-signal circuits. In *ASP-DAC '07: Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pages 316–323, Washington, DC, USA, 2007. IEEE Computer Society. → pages 21, 29, 35

[200] Farn Wang. Efficient verification of timed automata with BDD-like data structures. *STTT*, 6(1):77–97, 2004. → pages 27, 34, 35

[201] Farn Wang. Formal verification of timed systems: A survey and perspective. In *Proceedings of the IEEE*, page 2004, 2004. → pages 18, 29, 35

[202] Farn Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures. *IEEE Trans. Software Eng.*, 31(1):38–51, 2005. → pages 27, 34, 35

[203] Farn Wang. REDLIB for the formal verification of embedded systems. In *ISoLA*, pages 341–346. IEEE, 2006. → pages 15, 34, 35

[204] Farn Wang, Geng-Dian Hwang, and Fang Yu. TCTL inevitability analysis of dense-time systems. In Oscar H. Ibarra and Zhe Dang, editors, *CIAA*, volume 2759 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2003. → pages 35

[205] Ping-Ying Wang, J. H. C. Zhan, Hsiang-Hui Chang, and H. M. S. Chang. A digital intensive fractional-N PLL and all-digital self-calibration schemes. *Solid-State Circuits, IEEE Journal of*, 44(8):2182 –2192, August 2009. → pages 3, 5, 168

[206] David S. Watkins. *Fundamentals of Matrix Computations*. John Wiley & Sons, Inc, New York, NY, USA, 1991. 0-471-61414-9. → pages 197

[207] Neil H. E. Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, 3^{rd} edition, 2004. → pages 52, 59

[208] Chao Yan. Coho: A verification tool for circuit verification by reachability analysis. Master's thesis, The University of British Columbia, August 2006. → pages 46, 94, 100, 102, 104

[209] Chao Yan and Mark Greenstreet. Metastability analysis of Rambus ring oscilator. Technical Report in preparation, Computer Science Department, University of British Columbia, 2011. → pages 156

[210] Chao Yan and Mark R. Greenstreet. Circuit level verification of a high-speed toggle. In *FMCAD* [3], pages 199–206. → pages 9, 130

[211] Chao Yan and Mark R. Greenstreet. Faster projection based methods for circuit level verification. In *ASP-DAC* [4], pages 410–415. → pages 9, 130

[212] Chao Yan and Mark R. Greenstreet. Verifying an arbiter circuit. In Alessandro Cimatti and Robert B. Jones, editors, *FMCAD*, pages 1–9, Piscataway, NJ, USA, November 2008. IEEE Press. → pages 9, 138

[213] Chao Yan, Mark R. Greenstreet, and Jochen Eisinger. Formal verification of arbiters. *The 16th IEEE International Symposium on Asynchronous Circuits and Systems*, May 2010. → pages 9, 138

[214] Chao Yan, Mark R. Greenstreet, and Marius Laza. A robust linear program solver for reachability analysis. In *Proceedings of the First International Conference on Mathematical Aspects of Computer and Information Sciences (MACIS)*, pages pp231–242, Beijing, China, July 2006. → pages 46, 94, 102

[215] Chao Yan and Kevin D. Jones. Efficient simulation based verification by reordering. *DVCon*, February 2010. → pages 3

[216] Sergio Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997. → pages 15, 34, 35

[217] J. Yuan and C. Svensson. High-speed CMOS circuit technique. *Solid-State Circuits, IEEE Journal of*, 24(1):62–70, February 1989. → pages 8, 51

[218] Mohamed H. Zaki. *Techniques for the Formal Verification of Analog and Mixed- Signal Designs*. PhD thesis, Department of Electrical and Computer Engineering, Concordia University, September 2008. → pages 38

[219] Mohamed H. Zaki, Ghiath Al Sammane, Sofiène Tahar, and Guy Bois. Combining symbolic simulation and interval arithmetic for the verification of AMS designs. In *FMCAD* [3], pages 207–215. → pages 16

[220] Mohamed H. Zaki, Sofiène Tahar, and Guy Bois. A practical approach for monitoring analog circuits. In Gang Qu, Yehea I. Ismail, Narayanan

Vijaykrishnan, and Hai Zhou, editors, *ACM Great Lakes Symposium on VLSI*, pages 330–335. ACM, 2006. → pages 12, 21, 38, 40

[221] Mohamed H. Zaki, Sofiène Tahar, and Guy Bois. Formal verification of analog and mixed signal designs: A survey. *Microelectronics Journal*, 39(12):1395 – 1404, 2008. → pages 18

[222] Bo Zhang, M. Goodson, and R. Schreier. Invariant sets for general second-order low-pass delta-sigma modulators with DC inputs. In *Circuits and Systems, 1994. ISCAS '94., 1994 IEEE International Symposium on*, volume 6, pages 1 –4 vol.6, may-2 jun 1994. → pages 39

[223] V. Zolotov, D. Blaaw, S. Sirichotiyakul, M. Becer, C. Oh, R. Panda, A. Grinshpon, and R. Levy. Noise propagation and failure criteria for VLSI designs. pages 587–594, November 2002. → pages 65

Appendices

A

Geometrical Properties of Projectagons

In the dissertation, we focused on the application of projectagons in reachability analysis as described in Section 4.2. However, there are many interesting geometrical problems of projectagons. We summarize them in this appendix.

A.1 Non-Emptiness Problem is NP-Complete

Problem *Non-Emptiness Problem:* Given a d-dimensional projectagon **P** represented by n^s projection polygons, is the feasible region of the projectagon empty or not? The feasible region is the set of points whose projections lie in all projection polygons. We assume each projectagon polygon p_i corresponds to a two-dimensional plane s_i and has n_i vertices. We use $n = \sum_{i=1}^{n^s} n_i$ to denote the total number of polygon vertices of **P**. The coordinates of vertices are rational numbers where both numerators and denominators can be represented by at most k bits.

We first show that the feasible region of a projectagon is not empty iff the projectagon contains at lease one point whose coordinates are "small" numbers.

Lemma A.1.1 *If a d-dimensional projectagon* **P** *is feasible and the coordinates of all vertices are rational numbers using at most k bits, it must contain at least one feasible point which can be represented by* $O(n^3k)$*-bit rational numbers.*

Let us define $bt(a)$ as the minimum number of bits required to represent a value a, *i.e.*, $bt(a) = \lceil log(a) \rceil$. From the definition, it is easy to see:

$$bt(\sum_{i=1}^{n} v_i) \leq \max_{i \in \{1,\cdots n\}} (bt(v_i)) + log(n) \quad \text{(A.1)}$$

$$bt(\prod_{i=1}^{n} v_i) \leq \sum_{i=1}^{n} bt(v_i) \quad \text{(A.2)}$$

If the feasible region of **P** is non-empty, the region is a high-dimensional polyhedron, and all polyhedron vertices are intersections of hyper-planes. As described in Section 4.2, each hyper-plane corresponds to one polygon edge. Therefore, coordinates of a polyhedra vertices are the solution of a linear system: *i.e.*, $Ax = b$, where A is a COHO matrix.

It is obviously that all numbers of A and b can be represented by $O(k)$-bit numbers. This is because each constraint of the linear system corresponds to one polygon edge and values of A_i and b_i are computed from two k-bit vertices. Using the least common multiple of denominators of all rational numbers of A and b, the linear systems can be translated to an equivalent integer linear system $\hat{A}x = \hat{b}$. By Equation A.2, the least common multiple uses at most $O(d^2)k$ bits, thus, all integers of $\hat{A}$ and $\hat{b}$ can be represented by $O(d^2k)$ bits[1].

By Cramer's rule [206, Chapter 1.8], the solution of the linear system $\hat{A}x = \hat{b}$ can be expressed as

$$x_i = \frac{det(\hat{A}_i)}{det(\hat{A})} \quad i = 1 \cdots d, \quad \text{(A.3)}$$

where $det(\hat{A})$ denotes the determinant of matrix $\hat{A}$, and $\hat{A}_i$ is the matrix formed by replacing the i^{th} column of $\hat{A}$ by the column vector $\hat{b}$. The determinate of matrix $\hat{A}$ is defined as

$$det(\hat{A}) = \sum_{\sigma \in S_d} sgn(\sigma) \prod_{i=1}^{d} \hat{A}_{i,\sigma_i} \quad , \quad \text{(A.4)}$$

[1]There are at most $2d$ non-zero elements in the COHO matrix. Therefore, all integers of $\hat{A}$ and $\hat{b}$ are at most $O(dk)$ bits.

where σ is a permutation of the set $\{1, 2, \cdots d\}$ and S_d is the set of all permutations.

By Equation A.2, the product $\prod_{i=1}^{d} A_{i,\sigma_i}$ in Equation A.4 is at most $O(d^3k)$ bits large because because all number of $\hat{A}$ and $\hat{b}$ uses at most $O(d^2k)$ bits. Similarly, by Equation A.1 the value of $det(A)$, which is the sum of $d!$ number of products, are at most $O(d^3k + dlog(d)) = O(d^3k)$ bits. Therefore, the solution x are rational numbers whose numerators and denominators uses at most $O(d^3k)$ bits. ∎

Theorem A.1.2 *The non-emptiness problem of projectagons is NP-complete.*

Proof We shall next show that the non-emptiness problem is in NP by showing that it has polynomial certifications and it is NP-hard by constructing a polynomial-time reduction from the 3-SAT problem.

$\Longrightarrow$**The non-emptiness problem is in NP.**

From Lemma A.1.1, the non-emptiness of a projectagon can be solved by checking all points whose coordinates uses at most $O(d^3k)$ bits. Given such a point pt, we show there is a polynomial time certification. By the definition of projectagons, the point pt is inside $\mathbf{P}$ iff the two-dimensional projection of pt onto each projection plane is contained by the corresponding projection polygon of $\mathbf{P}$. It is well known that the algorithm that checks if a two-dimensional point inside a polygon has time complexity $O(n_i)$ [171, Chapter 2.2.1] where n_i is the number of polygon vertices. Because pt uses at most $O(d^3k)$-bit, the total running time T of the containment testing procedure on all projection planes is at most $T = \sum_{i=1}^{n^s}(O(n_i) \cdot O(d^3k)) = O(nd^3k)$. Because the size of $\mathbf{P}$ is $|\mathbf{P}| = o(nk)$, we have $T = o(|\mathbf{P}|^4)$. Therefore, the certification can be verified in polynomial time thus the non-emptiness problem is in NP.

$\Longleftarrow$ **The non-emptiness problem is NP-hard.**

Given a 3-SAT problem $\mathbf{S}$ with n variables $x_1, \ldots, x_n$ and m clauses, a corresponding projectagon $\mathbf{P}$ can be constructed within a polynomial time as follows:

1. Create a variable $X_{i,j}$ for all $i, j \in \{1, \ldots, n\}, i < j$.

2. Create a projection plane for each pair of variables $X_{i,j}, X_{k,l}$ where $i < j < k < l$ for all $i, j, k, l \in \{1, \ldots, n\}$. On each plane, place 16 squares with unit length as shown in Figure A.1. Each cell denotes one possible assignment to

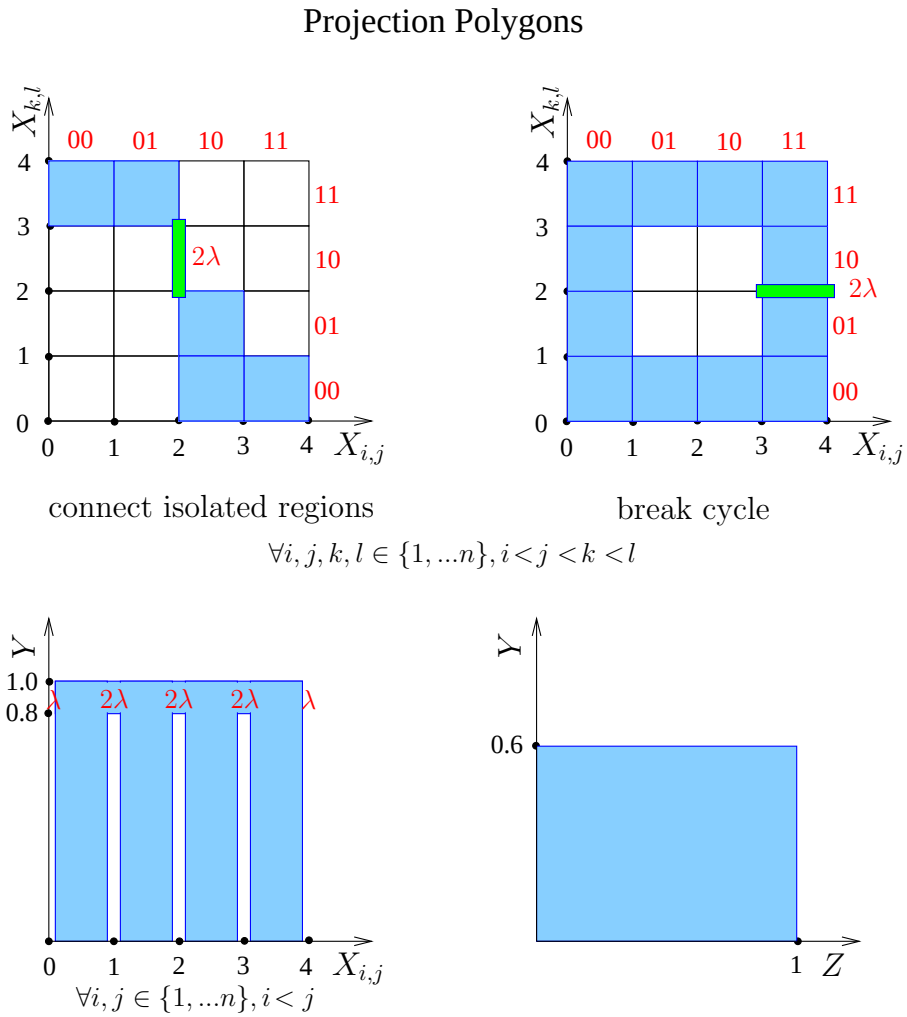


Figure A.1: Reduction from a 3SAT Problem to a Non-Emptiness Problem

variables x_i, x_j, x_k, x_k in the 3-SAT problem **S**. All cells are labeled as feasible at the beginning.

3. For each clause of **S** with three variables, find all planes that contains these variables (at most $n-3$ planes) and remove all cells correspond to the unsatisfiable states of the clause from all these planes.

4. Construct a simple polygon that contains all feasible cells left at the end of previous steps. Feasible regions resulted from above step may contain separated regions as shown in the top-left figure of Figure A.1 or loops as shown in the top-right figure of Figure A.1. Simple polygons can be contracted by either connecting isolated regions by thin bridges with length 2λ ($\lambda \ll 1$) or breaking cycles by removing thin ditches as shown in Figure A.1.

5. Add the projectagon planes and projection polygons as shown in the bottom figures of Figure A.1.

The projectagon has $\binom{n}{2}+2 = O(n^2)$ variables and $\binom{n}{4}+\binom{n}{2}+1 = O(n^4)$ projection planes. Obviously, step 1 costs $O(n^2)$ time, step 2 costs $O(n^4)$ time. The running time of step 3 is at most $O(m \cdot n^4)$ when all projection planes are checked for all clause. Step 4 is completed in $O(n^2)$ time because each polygon can be constructed in constant time[2]. Step 5 costs $O(n^2)$ time. Therefore, the transformation is polynomial-time ($O(|\mathbf{S}|^4)$).

We now claim that the projectagon **P**, as constructed above, has non-empty feasible region iff **S** is satisfiable. For suppose that **S** has a satisfiable assignment to variables $x_1, \ldots, x_n$, and the satisfiable assignment corresponds to a feasible region for all variables $X_{i,j}$. On each projection plane, the square corresponds to the projection of the feasible region are not removed in step 3. Therefore, **P** is not empty at the end because it must contain a part of the feasible region (boundaries are trimmed by λ in step 4).

For the *if* part, suppose that **P** has a non-empty feasible region; then the feasible region must contains at least one cell that corresponds to an assignment to variables $x_1, \ldots, x_n$. Projection polygons constructed in step 5 requires all variables

[2]Step 4 can be pre-computed as there are only 2^{16} possible results.

$X_{i,j}$ must be away from integer variables by at least λ. This eliminates "false" feasible regions caused by bridges in step 4. Therefore, the assignment corresponds to the feasible region satisfies **S**.

Therefore, a procedure for transforming a 3-SAT problem to a non-emptiness problem have been presented. The construction of the non-emptiness problem can be carried out in polynomial time. The non-emptiness problem is also in NP as proved above; hence it is NP-complete. ∎

Corollary A.1.3 *Projectagon is not a canonical representation method.*

Apparently, there are more than one projectagon to represent the empty region. ∎

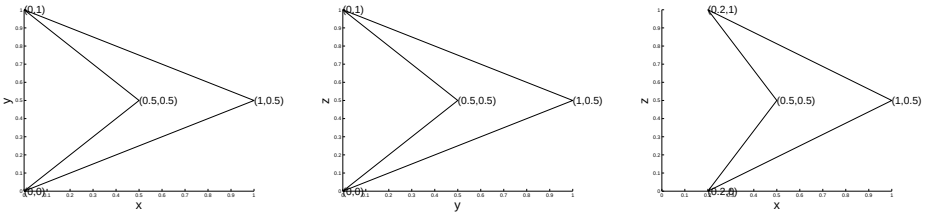
A.2 Removing Infeasible Regions

As described in Section 4.3.3, the infeasible region of a projectagon must be removed in each step of COHO's reachability analysis. We presented an algorithm to over-approximate the feasible projectagon as described in Algorithm 4 (lines 23-32). In each iteration of the algorithm, each projection polygon is clipped according to the constraints from convex hulls of other projection polygons. However, this procedure might take infinite number of iterations. We present an example here.

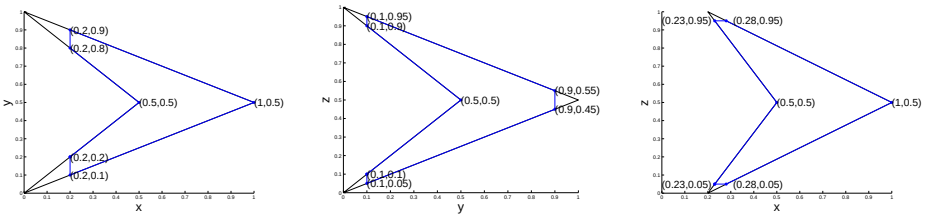
The example is 3-dimensional with three projection planes: $(x,y),(y,z)$ and (x,z). As shown in Figure A.2 (Step 0), three projection polygons are:

$$\begin{aligned}(x,y) &= \left[\begin{array}{cccc}(0,0) & (1,0.5) & (0,1) & (0.5,0.5)\end{array}\right]\\ (y,z) &= \left[\begin{array}{cccc}(0,0) & (1,0.5) & (0,1) & (0.5,0.5)\end{array}\right]\\ (x,z) &= \left[\begin{array}{cccc}(0.2,0) & (1,0.5) & (0.2,1) & (0.5,0.5)\end{array}\right]\end{aligned}$$

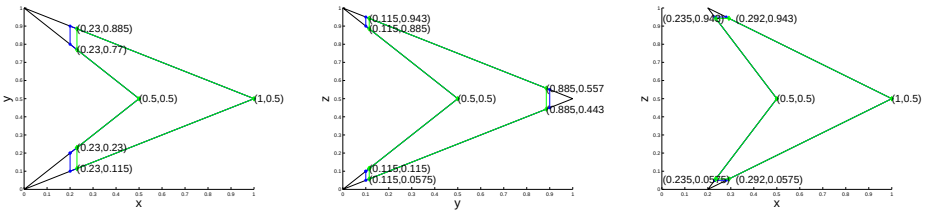
From the (x,z) projection polygon, the range of the value of x can be computed easily as $[0.2,1]$. Therefore, projection polygons on planes (x,y) and (y,z) are clipped according to our algorithm in Algorithm 4. Figure A.2 (Step 1) shows the trimmed projection polygons (blue color polygons). At the end, the values of x,y and z are



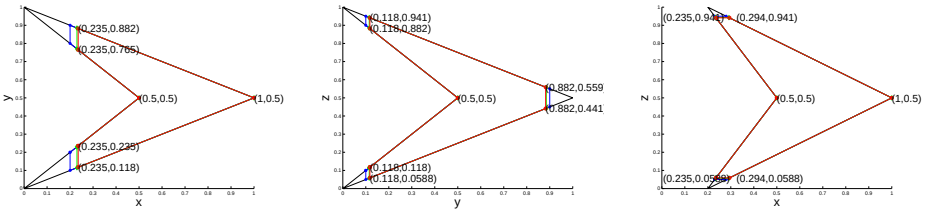
Step 0: Initial projectagon



Step 1: Trimmed projectagon after the first iteration



Step 2: Trimmed projectagon after the second iteration



Step ∞: Final feasible projectagon

Figure A.2: A 3-D Example of Removing Infeasible Regions

narrowed down to $[0.23, 1]$, $[0.1, 0.9]$ and $[0.05, 0.95]$, respectively. Similarly, projection polygons are clipped in the second iteration as shown in Figure A.2 (green color polygons in Step 2). At each iteration, the values of x, y and z are narrowed down to smaller intervals. However, the progress of each step becomes smaller and smaller. In each iteration, the value of x shrinks from $[a, 1]$ to $[\frac{4+3a}{20}, 1]$ and converges to $[\frac{4}{17}, 1]$ in the limit. Similarly, the values of y and z converge to $[\frac{2}{17}, \frac{15}{17}]$ and $[\frac{1}{17}, \frac{16}{17}]$, respectively. The red color polygons in Figure A.2 (Step ∞) shows the feasible projectagon. Therefore, approximation algorithms are applied in COHO.

A.3 Minimum Projectagons

Definition *Minimum Projectagons*: Given a high-dimensional region R, its minimum projectagon is a projectagon with minimum projection polygons whose feasible region contains R, *i.e.*, the projectagon does not contain R if any part of any projection polygon is removed. If the feasible region is exactly R, we call the projectagon *tight*; otherwise, we call it *non-tight*.

Problem *Unique Minimum Projectagon Problem*: Given a region R, is the minimum projectagon unique?

We believe the minimum projectagon is unique. However, we do not have a proof of the uniqueness problem now. We believe the minimum projectagon can be obtained by clipping infeasible regions.

Corollary A.3.1 *If the minimum projectagon is unique, the minimum projectagon is a canonical representation.*

Problem *Feasibility Problem*: Given a projectagon, make it feasible to its geometry representation, *i.e.*, projection polygons are feasible to each other.

We presented an algorithm to make a projectagon feasible to its inequality representation in Section 4.3.3. The algorithm may take infinity number of iterations to obtain the exact result as shown in Section A.2. Therefore, approximation techniques are applied in the COHO implementation as shown in Algorithm 4 (in Section 4.3.3). However, it is still an open problem to make a projectagon feasible

to its geometry representation. In fact, the non-emptiness problem is NP-complete as proved in Section A.1. Therefore, we believe approximation techniques must be applied in any implementation. A possible approach is to partition a non-convex projectagon into convex pieces, make each convex projectagon feasible, and compute the union of feasible convex pieces.

Problem *Closure Problem*: If $\mathbf{P}$ is a minimum projectagon, is $\mathit{convex}(\mathbf{P})$ a minimum and tight result? If $\mathbf{P_1}$ and $\mathbf{P_2}$ are two minimum projectagons, is the intersection $\mathit{intersect}(\mathbf{P_1}, \mathbf{P_2})$ a minimum and tight result? How about the union?

Apparently, the union of two minimum projectagons is neither minimum or tight as the union operation returns over-approximated results. We believe the results of intersection and convex hull operations are minimum and tight. However, they are also open problems.

B

Soundness of COHO Algorithms

This appendix proves the soundness of two algorithms presented in Section 4.4.1. These algorithms project an advanced face onto one corresponding projection plane to reduce projection errors. To ensure soundness, the face to be advanced are either bloated inward by f_b in the first approach or enlarged on all other directions by f_h in the second approach.

As described in Section 4.3.3, all trajectories from the current projectagon move by at most Δd on each direction in the time step $[0, \Delta t]$. With this assumption, if the value of f_b is no less than $2\Delta d$ as shown in Figure B.1(a), the soundness of the first approach can be proved easily. Because all points on a projectagon face can move inward by at most Δd, the advanced face on time Δt must be outside the blue face as shown in Figure B.1. Similarly, all trajectories that may reach points outside the blue face must has distance smaller than Δd from the blue face. Therefore, all such trajectories are included in the bloat face illustrated as red regions in Figure B.1(b). Hence, the first approach guarantees all approximations are conservative.

In the second approach, the height of face is increased by $f_h = 2\Delta d + h(2\Delta d)$, where $h(2\Delta d)$ is computed by the interval closure algorithm as shown in Figure B.2. Figure B.1(b) provides an example for the computation of f_h. We first bloat a face corresponds to an edge e inward by $2\Delta d$, then apply the interval closure algorithm described in Section 4.2.3 to find the height of the bloated face. The height is increased further by 2Δ as extra guards like we used in the first approach.

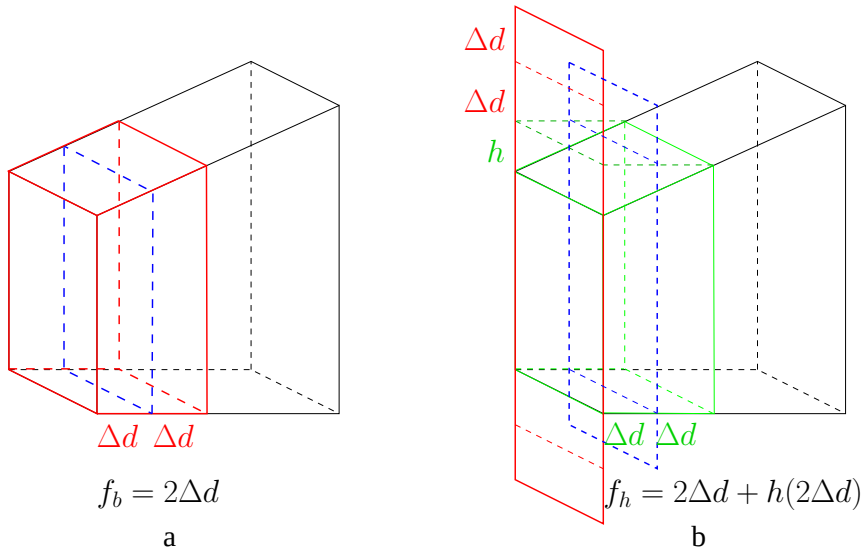


Figure B.1: Computation of f_b and f_h.

```
HyperRectangle IntervalClosure(Edge e, ProjectionPolygon p, Real Δd) {
    /* e is an edge of polygon p.
        Δd is the bloat amount for the current time step.
        Return the hyper-rectangle of interval closure bounds
            for e for the current time step.
    */
    Let r be the oriented rectangle that contains all points within
    distance 2Δd of e by the ℓ∞ metric.
    Let q be the intersection or r and p.
    Let b be the bounding box of q.
    Let h0 be the hyper-rectangle obtained by interval closure
        starting with b and using all of the other projection polygons.
    Let fh = bloat(h0, 2Δd). return(fh).
}
```

Figure B.2: Computing Height of Faces to be Advanced.

To establish the soundness of the interval closure method from Figure B.2, we consider a projectagon, and assume that one of the projection planes has the basis (x,y). We show that all points reachable from the projectagon by the end of the timestep are contained in the (x,y) projection polygon at the end of the timestep. Let p be an arbitrary point of the projection polygon at the beginning of the timestep. If p is further than 2Δ from the boundary of the (x,y) projection polygon at the beginning of the timestep, then any point reachable from p at the end of the timestep will be inside the time advanced polygon, because trajectories from p can move by at most Δd units outward and points on the boundary of projection polygon can move by at most Δd units inward during the timestep.

Otherwise, let e be an edge of the (x,y) projection polygon that is within distance $2\Delta d$ of p. By construction, the bloated face contains p. Accordingly, the constructed face to be advanced has feasible regions that extend by 2Δ in all of the other dimensions beyond the nearby (i.e. within 2Δ) points of projectagon. These extensions create a "parapet" to ensure that trajectories from faces for other projection polygons cannot "escape" this polygon. In particular, the face may shrink by at most Δd along any dimension (illustrated as blue regions in Figure B.1(b)), and any point can only reach other points that are within distance Δd (by the ℓ_∞) metric of itself. Thus, to reach a point outside of the (x,y) polygon, a trajectory from p would have to touch the feasible region for one of the faces arising from the (x,y) polygon. This means that points reachable from p are also reachable from the face that it touched, and therefore project to points on the (x,y) plane that are inside the time advanced projection polygon.

The values of f_b and f_h to ensure soundness are generally much larger than necessary. In the implementation of COHO, parameters are provided to users for obtaining trade-offs between approximation errors and soundness in theory. From our experiences, Δd is a reasonable value for both f_b and f_h.