EFFICIENT, SOUND FORMAL VERIFICATION FOR ANALOG/MIXED-SIGNAL CIRCUITS

by

Andrew N. Fisher

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

The University of Utah

August 2015

Copyright © Andrew N. Fisher 2015

All Rights Reserved

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Andrew N. Fisher**

has been approved by the following supervisory committee members:

Chris J. Myers	, Chair	**May 29, 2015** Date Approved
Kenneth Stevens	, Member	**May 29, 2015** Date Approved
Priyank Kalla	, Member	**May 29, 2015** Date Approved
Scott Little	, Member	**June 2, 2015** Date Approved
Eric Mercer	, Member	**May 29, 2015** Date Approved

and by **Gianluca Lazzi**, Chair of

the Department of **Department of Electrical and Computer Engineering**

and by David Kieda, Dean of The Graduate School.

ABSTRACT

The increasing demand for smaller, more efficient circuits has created a need for both digital and analog designs to scale down. Digital technologies have been successful in meeting this challenge, but analog circuits have lagged behind due to smaller transistor sizes having a disproportionate negative affect. Since many applications require small, low-power analog circuits, the trend has been to take advantage of digital's ability to scale by replacing as much of the analog circuitry as possible with digital counterparts. The results are known as *digitally-intensive analog/mixed-signal* (AMS) circuits. Though such circuits have helped the scaling problem, they have further complicated verification. This dissertation improves on techniques for AMS property specifications, as well as, develops sound, efficient extensions to formal AMS verification methods.

With the *language for analog/mixed-signal properties* (LAMP), one has a simple intuitive language for specifying AMS properties. LAMP provides a more procedural method for describing properties that is more straightforward than temporal logic-like languages. However, LAMP is still a nascent language and is limited in the types of properties it is capable of describing. This dissertation extends LAMP by adding statements to ignore transient periods and be able to reset the property check when the environment conditions change.

After specifying a property, one needs to verify that the circuit satisfies the property. An efficient method for formally verifying AMS circuits is to use the restricted polyhedral class of *zones*. Zones have simple operations for exploring the reachable state space, but they are only applicable to circuit models that utilize constant rates. To extend zones to more general models, this dissertation provides the theory and implementation needed to soundly handle models with ranges of rates.

As a second improvement to the state representation, this dissertation describes how octagons can be adapted to model checking AMS circuit models. Though zones have efficient algorithms, it comes at a cost of over-approximating the reachable state space. Octagons have similarly efficient algorithms while adding additional flexibility to reduce the necessary over-approximations.

Finally, the full methodology described in this dissertation is demonstrated on two examples. The first example is a switched capacitor integrator that has been studied in the context of transforming the original formal model to use only single rate assignments. Th property of not saturating is written in LAMP, the circuit is learned, and the property is checked against a faulty and correct circuit. In addition, it is shown that the zone extension, and its implementation with octagons, recovers all previous conclusions with the switched capacitor integrator without the need to translate the model. In particular, the method applies generally to all the models produced and does not require the soundness check needed by the translational approach to accept positive verification results. As a second example, the full tool flow is demonstrated on a digital C-element that is driven by a pair of RC networks, creating an AMS circuit. The RC networks are chosen so that the inputs to the C-element are ordered. LAMP is used to codify this behavior and it is verified that the input signals change in the correct order for the provided SPICE simulation traces.

Baruch Hashem

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

Doing a second Ph.D. was not much easier than doing a first. I learned many new skills, not only in my new field, but also in how to do research in general. I would like to thank my advisor, Chris Myers, for his contributions to this new knowledge. Chris taught me a lot about how to write papers, create presentations, and do reviews. His support in rounding out my academic experience has been invaluable.

Next, I would like to thank my family. They have been a great source of love and support for all things not academic. They have been there through the highs and the lows that come from endeavoring on a difficult course. I would like to especially thank my parents, Kevin and Rhonda Fisher, whose unfailing support has enabled me to finish this degree.

I would also like to thank my labmates, Curtis Madsen, Nick Roehner, and Zhen Zhang. They each helped me to really become part of the group. Curtis taught me a lot about how to be an admin for the lab and was a great companion for discussing computer related topics. Nick provided conversations on the more general topics of philosophy, music, and culture and was over all a very positive force in the lab. Zhen has been my sounding board for numerous topics related to my work and helped me to get my thoughts straight. He was also a willing participant in conversations with me regardless of whether they were closely related to the topic at hand or were something that just occurred to me. In addition, I would like to thank the rest of my labmates, Dhanashree Kulkarni, Leandro Watanabe, Tramy Nguyen, Scott Glass, and Tim Pruss for putting up with me for these few years.

I would like to thank my committee, Ken Stevens, Priyank Kalla, Eric Mercer, and Scott Little. They have been great in helping me throughout the degree. They gave me several ideas for current and future work and have been there whenever I asked for help.

I would like to thank my friends who were not directly related to my research, including Neil Cotter, Derek Crane, Derek Moore, and Jason Parkin. Neil and I both have an interest in mathematics and our chats have provided much needed distractions and have helped me keep in touch with my math background. Derek Crane introduced me to the topic of cybersecurity and fired my interest in that direction. Derek Moore and Jason Parkin have helped to make the long train rides to the university more enjoyable and have been fun to talk to about computer engineering related topics.

Finally, I would like to thank all the past members of the lab who contributed to LEMA. Without their work, this work would not be possible. In particular, I would like to thank Satish Batchu, Kevin Jones, Dhanashree Kulkarni, Scott Little, and David Walter for their contributions to LEMA.

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. CCF-1117515 and an ARCS fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

CHAPTER 1

INTRODUCTION

As embedded systems become more popular, it is increasingly difficult to meet the challenge of creating higher performance circuits within smaller sizes and lower power constraints. Digital devices have been successful in meeting this demand. As one example, in 1971 the Intel 4004 processor had 2,300 transistor and in 2012 the number had grown to over 1.4 billion transistors in the 3rd Generation Intel Core I7 [60]. In the same time frame, the process node decreased from 10μm to 22nm. However, shrinking transistor sizes and increasing transistor counts are not enough to create useful designs, especially when the number of transistors is in the billions. Abstraction is necessary to handle such complex designs. Here too, the simplicity of digital circuits has met the challenge by introducing *computer-aided design* (CAD) tools that support a wide range of abstraction levels and automate large portions of the design process. In particular, several methods have been successfully applied to verifying the correctness of designs [13, 23, 24, 26, 27, 47, 63, 65].

In contrast, analog circuits have not been able to keep pace. Performance doubles about every two years for digital circuits while it takes four to five years for analog performance to double [83]. In particular, scaling is not as favorable to analog circuits. Noise is more prevalent, gains are harder to achieve, and the lower power requirement only exacerbate these concerns. However, analog circuits are necessary in any application that interacts with the real world, such as cell phones and sensors. To cope with this challenge, analog designers have turned to digital alternatives as much as possible resulting in *analog/mixed-signal circuits* (AMS). In fact, some designs have reached the point of essentially adding a processor and are more accurately described as *digitally-intensive AMS* circuits. Of course, adding this additional circuitry increases the complexity, making errors more likely, especially when designs are still being done by hand. Although CAD tools have made some strides in areas like sizing and layout [28], the verification problem is particularly difficult to handle in the AMS setting. The traditional simulation methodology is difficult to adapt to the large transistor counts resulting from the digital portions while digital techniques do not support the continuous nature of the analog portion. Even so, strides have been made to adapt simulation-based verification and convert digital verification methodologies to the AMS domain. These methods are briefly surveyed in Section 1.1. Section 1.2 details this dissertation's contribution to these methods. Finally, Section 1.3 concludes the chapter with an outline of the rest of the dissertation.

1.1 Survey of AMS Circuit Verification

Since AMS circuits combine both the analog and digital domains, it is natural to start with either and attempt to extend the techniques to the entire AMS design. Indeed, most AMS verification techniques have been developed in this way and fall into one of four categories: simulation-based, equivalence checking, theorem proving, and model checking. Each of these categories is explored in greater detail in the following subsections. For additional information, one can consult recent survey papers [6, 103, 117].

1.1.1 Simulation-Based

Traditional simulation-based verification utilizes some variant of the *Simulation Program with Integrated Circuit Emphasis* (SPICE). SPICE [85] simulation does not require any additional work to apply to AMS designs, since, after all, the digital circuits are nothing more than transistors with a convenient abstraction. The difficulty comes when handling the large number of transistors introduced by the digital portion. For example, a digitally-intensive *phase locked loop* (PLL) can take days or even weeks to verify due to the large number of transistors. Longer simulation time naturally leads to less simulation runs and severely slows down the design process. In turn, functional bugs are more likely to be missed. Simulation is also plagued by the need for a designer to check simulations by hand to determine whether the circuit is functioning correctly, which becomes more difficult as the designs become more complex.

Instead of having a designer check all the simulations by hand, an alternative is to use monitors [5, 22, 31–33, 36, 42, 44, 75–79, 99, 106, 107, 113, 114, 119]. The first step in automatization is to develop a language for describing correct behaviors. Such descriptions are usually written in a type of formal language like *linear temporal logic* (LTL) [13, 97] or the *property specification language* (PSL) [1], which can be parsed algorithmically. Once the property language is chosen, it is used to create assertions that are checked against the traces. This check is performed in one of two ways: online or offline. Online monitoring usually relies on constructing a sequence recognizer that continually checks or monitors the signal. As soon as a trace violates the property, an error is raised. In offline monitoring, the whole set of simulations is constructed beforehand and then checked against the property.

Without having a runtime constraint, offline monitoring is more flexible and can use more complicated algorithms. Though offline and online monitors have made advances towards automation, they remain susceptible to missing errors due to incomplete simulations.

To mitigate the concerns with simulation, one can determine the *robustness* [32, 36] of the property. With robustness, one attempts to determine how sensitive a property is to changes in the simulation traces. Tolerances are provided to indicate how far traces can drift before a property is violated. So robust properties allow traces to vary widely before they are violated. The tolerances allow one to reduce the number of simulations since only one trace needs to be tested in the region defined by the tolerance. Though robustness helps to ensure that all necessary regions are tested, all traces have to be recalculated if any parameters change and sensitive properties still require a large number of simulations.

Another method to reduce the number of required simulations is *event-driven* (ED) simulation. ED simulation attempts to make the cost of creating simulations cheaper by using triggers to determine when to solve differential equations and which equations need to be solved. Like the online methods, however, these approaches ultimately cannot guarantee the correct functioning of a circuit and could miss important behaviors not being simulated.

If the cost of simulations is not a concern, one can use Monte Carlo simulation [113] to determine a properties sensitivity to changes in parameters. With Monte Carlo methods, one runs randomly chosen simulations and aggregates the data to determine the possible distribution of solutions. Confidence intervals are provided to determine the likelihood of failure when conditions change. Of course, the downside is that even more simulations need to be calculated compounding the problem when simulations are expensive.

In contrast to the above methods, which still calculate individual traces, *symbolic simulation* [108, 111] gathers several traces together in an abstract representation. The simulation is then calculated by manipulating the whole collection. The method of [108] uses a first-order language to describe the state space. The states are formally manipulated to determine how the collection evolves. In [111], the authors use a similar approach by representing states as conjunctions involving interval constraints. Symbolic solution techniques are then used to determine how the state evolves. Though several traces are simulated together, there is still no guarantee that all behaviors are explored and the describing formulas can experience exponential blow-up.

1.1.2 Equivalence Checking

A related notion to verifying circuit correctness is *equivalence checking* where one verifies two circuits perform the same task. In fact, a method of equivalence checking can often be used as a method of verification by using one circuit description as the specification and the other as the circuit to verify. Then, determining the circuits are equivalent is the same as determining that the circuit matches the specification. In the digital context, equivalence checking is simply the process of determining if two descriptions perform the same function. This ability enables digital circuits to be designed at different levels of abstraction by ensuring that each level is equivalent to the others. For example, one can ensure that a *register-transfer level* (RTL) description is the same as the synthesized transistor implementation.

With all the benefits equivalence checking brings to digital circuits, it would be advantageous to have this ability for AMS designs. Although it is straightforward to define the equivalence of two digital circuits, AMS circuits are not as simple. Equivalences have been based on frequency domain characteristics [15, 56, 101] and time domain characteristics [55, 59]. In addition, methods have utilized *hardware description languages* (HDL). Specifically, the AMS extension of the *very high speed integrated circuit hardware description language* (VHDL), called VHDL-AMS [100]. In each case, equivalence checking is used to determine if descriptions on different levels of abstraction or different implementations on the same level are equivalent.

The methods based on frequency domain characteristics define two descriptions to be equivalent if their transfer functions are equivalent. In [15], the authors map the transfer function to the Z-domain and utilize a digital representation using adders, multipliers, and delay elements. Staying in the s-domain, [56] uses interval arithmetic and tolerances to establish that the transfer functions describe essentially the same behavior. Finally, [101] turns the check into an optimization where one optimizes the conformation of the magnitude and phase response of the implementation to a tolerance around the specification. Though this is useful for analog circuits, it does not extend well to the full range of behavior for AMS circuits since not all behavior needed is frequency based.

Staying in the time domain, instead of mapping to the frequency domain, [55] establishes equivalence by creating a correspondence between solutions of the respective governing differential equations. To identify this correspondence, local linearizations are employed so that the problem reduces to finding transformation matrices. Thus, for a given pair of systems, the local vector fields are computed for a point, then the transformation matrix is found, and a new point along the solution curve is selected. The process of finding local linearization and stepping along the solution curves is equivalent to numerically integrating the differential equation, so the method is as costly as jointly solving two sets of differential equations and is similar to the calculations needed for SPICE.

Instead of looking at local linearizations, [59] restricts the problem to circuits that have a global linear behavior. Thus, two circuits are equivalent if they exhibit the same linear behavior within a specified tolerance. To determine this linear behavior, a set of random inputs is applied to each system, the output responses are recorded, and the results of linear regression are compared. For general circuits, the design must first be partitioned into appropriate sized blocks and each of these blocks must be verified separately.

On a higher level of abstraction, [100] establishes equivalence based on VHDL-AMS circuit descriptions. The check is split into identifying the digital and analog pieces separately. The digital pieces are compared by creating a mitre and then using Boolean satisfiability (SAT) and *binary decision diagram* (BDD) [3, 21] techniques to verify equivalence. For the analog portion, the corresponding analog pieces are tied together with a comparator and are equated using a combination of rewriting rules and simulation to determine equivalence. Since the digital and analog portions are handled separately, their interaction is not taken into account.

1.1.3 Theorem Proving

Theorem proving verifies circuit correctness by mapping the circuit and specification to a formal description and establishing a proof that the circuit satisfies the property. Formal descriptions include combinations of linear constraints [46, 48–50], *systems of recurrence relations* (SRE) [4, 5], and the *differential temporal logic* (dTL) [61, 82, 88–95].

In [48–50], the linear or piecewise linear nature of many analog components is exploited to map the behavior of voltages and currents to a conjunction of linear constraints. After also giving the specification in terms of linear constraints, a decision procedure is applied to determine if the circuit description implies the specification. As a variant of this idea, [46] maps a circuit netlist together with a VHDL-AMS specification to the *prototype verification language* (PVS) [86] and applies the PVS decision procedures to prove that the netlist satisfies the property. In this context, the circuit description is again based around linear constraints and is thus applicable to analog circuits that can be described by linear or piecewise linear functions. Since these methods are based on intervals and linearizations, they are approximate; however, steps are taken to ensure that the results are conservative.

SREs are a more general class of equations than linear constraints. They allow for more arbitrary combinations of additions and multiplication, as well as allowing for division, logical expressions, and if-then-else constructs. The work of [4] maps AMS circuits and corresponding properties to SREs by using a set of rewriting rules. After obtaining the SRE description, the proof proceeds by an induction argument. In [4], the properties are written almost directly in terms of SREs; however, [5] extends the property descriptions to PSL [1, 45].

Instead of extending the the class of equations, one can build on the logic specification idea. The series of papers [61, 82, 88–95] directly incorporate differential equations into the logic semantics creating the *differential dynamic logic* (d$\mathcal{L}$) and its extensions *differential temporal dynamic logic* (dTL) and dTL^2. With d$\mathcal{L}$, dTL, and dTL^2, one can directly translate a hybrid automata model into a hybrid program that is a description of the circuit in the formal language. Once the model is constructed, one can then proceed to prove properties about the system using formal proofs in d$\mathcal{L}$, dTL, or dTL^2. In particular, one can write properties for the correct operation of the circuit and prove whether the system satisfies them. In all cases, the proofs can be partially automated. When running the automation, one of three cases occurs: a proof that the property is satisfied, a proof that the property is not satisfied, or the theorem prover cannot produce a proof either way. When the theorem prover cannot produce a proof, it is necessary that additional information be found (by hand) to give to the theorem prover so it can make progress.

1.1.4 Model Checking

Model checking, also known as reachability analysis, is the process of finding all the possible states a system can reach from the initial conditions. Properties are translated into forbidden states and are checked by determining if these states are in the reachable set. Model checking is a well established methodology in the digital domain [26, 27, 63]. Though digital model checking has to contend with large state counts, the state space is computable and the basic process is straightforward. AMS model checking, on the other hand, has an infinite state space that is usually not computable [58, 96]. Even still, progress has been made with approaches ranging from discretizing the state space [51–53, 105], performing numeric next state calculations [39, 70, 71, 73, 109, 110, 118], manipulating symbolic next states [112], and combining different methods [116].

One of the earliest ideas for model checking AMS designs stems from the already available digital model checking tools. If the AMS problem is turned into a digital problem, then the rest of the checking can be done utilizing already existing tools. In [51–53], the authors attempt to turn the AMS model checking problem into a digital one by cutting the continuous space into hyper-cubes thereby discretizing the space. The hyper-cubes are formed by selecting a finite set of discrete points for each coordinate direction. Each hyper-cube becomes a state and transitions are added between any two states for which there is a local linear solution that starts in one state and ends in the other. In this approach, trajectories are best approximated when they move perpendicular to the face of the hyper-cubes; however, the hyper-cubes are fixed regardless of the direction of the dynamics. To achieve a better approximation, [105] extends the hyper-cube construction so they follow the path of the trajectories. To verify properties of interest, it is often necessary to pick fine-grained partitions which result in a large state space for the digital checker. Hence, the state space limit of digital model checking is often reached.

To avoid a complete discretization, one usually adopts a model formalism that has both discrete and continuous parts such as *linear hybrid automata* (LHAs) [7] or *labeled Petri nets* (LPNs) [74]. Due the presence of continuous variables, the state space is infinite, so equivalence classes of states are formed to produce a finite representation. These equivalence classes are usually some subclass of polyhedra. The choice of which subset is a trade-off between accuracy and complexity of representation. After selecting the subclass, the next challenge is to determine how the state space evolves. In [70], the authors allow time to flow forwards until a constraint on the continuous variables is violated. SPICE simulations are run to determine which constraints are reachable from the current state. Avoiding SPICE calculations, [118] uses a more symbolic approach. The state space is represented as a formula on intervals and interval arithmetic together with Taylor series approximations to find estimates on the solutions to the differential equations that start in the given state. Allowing for even more general sets of solutions, [39, 71] introduce a class of polyhedra defined by a fixed set of prechosen unit vectors. The next states are found by essentially incorporating a linear numerical integrator that is lifted to sets. By using set representation, these methods have the advantage of collecting together multiple solutions, though the complexity of finding these solutions is still high.

In an effort to simplify these calculations, [73] chooses a more restrictive polyhedra set. By reducing the complexity of the representing space, they are able to reduce the complexity of the next state calculations to essentially setting the upper bound variable constraint to their maximum values allowed by any constraints defined on the variables. Though the method reduces the calculation overhead, the restrictive nature of the polyhedral class leads to large over-approximations of the exact state space.

An even more symbolic approach to constructing next states is to encode the behavior as a BDD or SAT formula. In [112], the authors present two methods: the first is based on BDDs and the second on SAT. In both cases the states are represented symbolically as constraints on the system. In the BDD version, the method starts with the set of failure states and then iteratively calculates which states could have reached the current collection of states. The process continues until a fixed point is reached, which is then intersected with the initial states to determine if any of the initial states can lead to the failure. The SAT method starts with a symbolic representation for the initial state and then iteratively adds constraints that represent the next states reachable from the current states. This process is similar to the unrolling of a programming loop with each stage indicating what states are reachable after n iterations. Though these methods avoid performing numerical integration, the BDDs are difficult to maintain as the state space increases while the SAT method is only able to perform *bounded model checking*, that is modeling checking that is bounded in either time or number of iterations.

Recently, [116] has proposed a hybrid approach combining SPICE simulations and SAT solvers. Since SAT solvers can have an exponential runtime, [116] first uses SPICE simulations to reduce give an initial estimate of the state space and then runs a SAT solver to provide soundness. Running too many SPICE simulations follows the law of diminishing returns where the extra run-time needed to calculate the SPICE traces does not significantly reduce the run-time for the SAT solver. To find an optimal point, [116] uses a Bayesian estimate to determine how many SPICE simulations should be run.

1.2 Contributions

This dissertation presents improvements in the tools and algorithms used to specify and verify properties for AMS circuits. The four contributions of this dissertation are the following.

- An extension to the *language for analog/mixed signal properties* (LAMP).
- Developing the necessary theory for soundly exploring LPNs with ranges of rates and applying it to extending zones.
- Implementing a state exploration algorithm based on octagons.
- Demonstrating the new verification flow for `LEMA`.

The first contribution is to extend the AMS property language LAMP. The goal of LAMP [37, 66, 67] is to be a simple, intuitive language that is easier for nonexperts to use. The LAMP language is written more like a programming language and describes properties that are checked procedurally. Thus, LAMP avoids the need for analog designers to learn and understand temporal logics. In addition, LAMP is easy to translate into LPNs, making model checking simpler. However, LAMP is still at a nascent stage and does not support many constructs. Two new constructs are added to LAMP: a delay statement and a generalized always block. The delay statement allows LAMP to ignore transient periods without performing a check. The always block is given a sensitivity list that allows the property check to break out of the loop when the signal changes. This change allows the property to handle more general environments since checks can be aborted when the conditions change.

The second contribution is to provide the underlying theory necessary to perform zone-based state exploration for LPNs with ranges of rates. Part of the flexibility of the LPN semantics is to allow a continuous variable to be assigned a range of possible rates. Such generality aids in doing linear approximations to nonlinear dynamics and is built into the automatically generated LPNs in [73]. While some versions of state exploration can handle these ranges of rates, methods like zones are designed around having a single rate. Though zones are less accurate than other methods, they are easier to implement and have efficient algorithms, so it would be useful to apply them to LPNs with ranges of rates. It is shown that it is enough to consider only the extremal and zero rates. Using this fact, the zone based model checker algorithm of [72] is extended to handle ranges of rates.

The third contribution is to implement an octagon-based model checker. While zones have the advantage of having a simple representation and efficient algorithms, they lead to large over-approximations of the state space, especially in the presence of negative rates. Octagons add a symmetry that reduces the negative-rate over-approximations to the same level as positive rates, thus creating a more accurate state space representation. Moreover, the representation requires only a minor adjustment to the zonal version while the algorithms maintain the same level of complexity.

The fourth contribution is to demonstrate the complete new verification flow for `LEMA`. This dissertation adds additions to LAMP, extends the zone-based method to ranges of rates, and adds the new octagon-based model checker. With these additions, `LEMA`'s tool flow has been altered some from previous versions. Now, it is possible to learn models, create properties, and directly verify the properties against the models with the zone-based model checker and a new octagon-based model checker. These new changes are illustrated with the aid of a few small examples.

1.3 Dissertation Overview

The rest of the dissertation is divided into six chapters. The basic outline is a background chapter, a chapter per contribution, and a final concluding chapter.

Chapter 2 provides the background material for the dissertation. This chapter introduces the formal syntax and semantics for LPNs, and introduces the *LPN embedded mixed-signal analyzer* (`LEMA`) tool. It is critical to introduce LPNs since they are the formalism used by the work in this dissertation for modeling AMS circuits, as well as representing properties. In fact, the most relevant verification flow for this dissertation is the following. Start with an LPN model of the circuit. Combine this model with an LPN for the property. Finally, use a state exploration algorithm to determine whether a failure transition can ever fire. Every chapter that follows relates back to some aspect of this flow. Since this is the case, Chapter 2 also provides a brief overview of the tool `LEMA`.

Chapter 3 introduces the LAMP language. Before giving the formal definition of LAMP, this chapter provides an overview of other property languages used in specifying AMS properties. Some of these languages are illustrated by creating a property for a *phase interpolator* (PI). LAMP is then formally defined and used to specify the same PI property for comparison. The extensions to LAMP are folded into the definition of LAMP and are further illustrated by means of an example property for a *voltage controlled oscillator* (VCO). Specifically, the previous LAMP statements could not handle the transient behavior of the VCO that is present before the oscillation stabilizes. This behavior can now be ignored by using a delay statement. In addition, the always block is extended to allow breakouts when the environment changes. Previously, properties would incorrectly signal a failure if the environment changed in the middle of the check. The new always block allows the property to reset the check when the environment changes. This ability is illustrated using the the VCO. The always block is written to use the voltage so the correct oscillation can be checked after the voltage changes.

Chapter 4 provides the theory and implementation for handling ranges of rates. This chapter is comprised of 3 stages. It starts with a discussion of previous attempts to handle ranges of rates. In particular, two translational methods are discussed that both seek to solve the problem by changing the LPNs involved. It is already known that one of the approaches does not solve the problem; however, it is shown that the second approach does not work either. It may be possible to adapt these methods, but this approach leads to generating a new LPN model per LPN property. Following this discussion, some additional terminology for LPNs is introduced and then the general theory for handling ranges of rates is presented. Using the theory as inspiration, `LEMA`'s zone-based model checker is modified to handle ranges of rates.

Chapter 5 presents the use of octagons for AMS verification. This chapter starts by presenting an example where `LEMA`'s zone-based model checker falsely declares that an LPN fires a failure transition. It is also shown that by tightening the over-approximation for negative rates with the use of octagons, the false negative can be eliminated. Octagons are then formally defined and the zone-based model checker is modified to implement octagons. The new octagon model checker is then run on the example to show that indeed, the false negative is eliminated. This chapter concludes by indicating how much of an over-approximation is introduced by the operations on octagons.

Chapter 6 demonstrates `LEMA`'s full tool flow starting with properties and SPICE traces and ending with the extensions to the zone-based model checker and using the new octagon-based model checker. The first case study is a switched capacitor integrator. It is shown that the previous property of nonsaturation can be codified in LAMP and that all the previous results for this circuit can be recovered with less of a state count and without needing to translate the model or check additional conditions to ensure correctness. The second example is a digital C-element that is driven by two RC networks. The property provided is to ensure an ordering on the inputs for the C-element. It is shown how the property can be written in LAMP and the circuit is verified.

Chapter 6 demonstrates the complete new verification flow for `LEMA` with the the aid of two examples. In each case, a model is learned using the model generator, a property is written in LAMP, and the zone-based and octagon-based model checkers are applied. The first example is a switched capacitor integrator. This circuit has been studied before in [74]. The main changes to the previous demonstration are the addition of using LAMP, the removal of the model translation steps, the removal of any additional checks in order to accept positive verification results, and the use of octagons. The second example is a digital C-element whose inputs are driven by a pair of RC networks. The SPICE simulation traces were provided by a third party along with the property that the one input signal to the C-element changes before the other. This property is codified in LAMP, a model is learned from the SPICE traces, and the property is verified.

Chapter 7 provides a summary of the contributions of this dissertation. It recaps how LAMP is extended and what additional flexibility this provides. Then, it discusses how the theory for handling ranges of rates lays the foundation for implementations based on single rates. Next, the utility of using octagons is summarized followed by a brief discussion of how the case studies indicate where octagons and zones fit in view of more general polyhedral methods. This chapter concludes with future work.

CHAPTER 2

BACKGROUND

This dissertation utilizes LPNs [17, 66, 68, 73] as the chosen formal model for both modeling the behavior of circuits and for creating properties that check this behavior. As such, nearly every chapter depends on a knowledge of them. Furthermore, the work presented has been incorporated into `LEMA`. This chapter presents this necessary background. Section 2.1 introduces LPNs and Section 2.2 describes `LEMA`.

2.1 Labeled Petri Nets

This section provides an overview of the LPN formalism used in this dissertation. LPNs are a type of Petri net that add the ability to reason with continuous variables in addition to the discrete events supported by LPNs. LPNs are a culmination of a few iterations of extensions to Petri nets, a formalism introduced in [87]. The basic Petri net structure provides a framework for discrete events and naturally supports concurrent events. On this foundation, several variants have been developed including timed Petri nets (TPNs) [80], timed event/level (TEL) structures [18], first-order Petri nets (FOPN) [14], and many others. LPNs are a type of hybrid Petri (HPN), which incorporate timing, discrete events, and continuous variables. The formal definition is provided in Sections 2.1.1 and 2.1.2.

2.1.1 LPN Syntax

An LPN is a type of Petri net that has been augmented with a set of labels for modeling continuous variables and their rates of change. This does not preclude the use of discrete (or Boolean) variables since they can be modeled using a continuous variable with a rate of zero. LPNs are assumed to be safe and, in general, continuous variables are allowed to nondeterministically choose a rate from an interval of possible rates. Formally, an LPN is a tuple $N = \langle P, T, T_f, V, F, M_0, Q_0, R_0, L\rangle$ where:

- P is a finite set of places;
- T is a finite set of transitions;

- $T_f \subseteq T$ is a finite set of failure transitions;
- V is a finite set of continuous variables;
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation;
- $M_0 \subseteq P$ is the set of initially marked places;
- $Q_0 : V \to \mathbb{Q}$ is the initial value of each continuous variable;
- $R_0 : V \to \mathbb{Q} \times \mathbb{Q}$ is the initial range of rates for each continuous variable;
- L is a tuple of labels defined below.

Failure transitions are used by LPNs to signal when a failure has occurred, and the *flow relation*, F, is used to describe how the places and transitions are connected. Every transition $t \in T$ has a *preset* denoted by $\bullet t = \{p \mid (p, t) \in F\}$ and a *postset* denoted by $t\bullet = \{p \mid (t, p) \in F\}$. The set $\mathbb{Q} \times \mathbb{Q}$ is identified with the set of intervals so that $(a, b) \in \mathbb{Q} \times \mathbb{Q}$ corresponds to the interval $[a, b]$ with $a \leq b$. Furthermore, the intervals are restricted to either be nonnegative (that is, $a \geq 0$) or negative (that is, $b < 0$). The *labels*, L, for an LPN are defined by the tuple $L = \langle En,\ DA,\ VA,\ RA \rangle$:

- $En : T \to \mathcal{P}_\phi$ labels each transition $t \in T$ with an enabling condition;
- $DA : T \to \mathcal{P}_\chi$ labels each transition $t \in T$ with an expression for the delay before a transition t can fire;
- $VA : T \times V \to \mathcal{P}_\chi$ labels each transition $t \in T$ and continuous variable $v \in V$ with an expression for the continuous variable assignment that is made to v when t fires;
- $RA : T \times V \to \mathcal{P}_\chi$ labels each transition $t \in T$ and continuous variable $v \in V$ with an expression for the rate assignment that is made to v when t fires.

The enabling conditions are Boolean expressions, $\mathcal{P}_\phi$, that satisfy the grammar:

$$\phi \quad ::= \quad \textbf{true} \mid \neg\phi \mid \phi \wedge \phi \mid v \geq c$$

where $\neg$ is negation, $\wedge$ is conjunction, v is a continuous variable, and c is a rational constant. The expressions **false** and $\vee$ are defined from these. In addition, the negation of $v \geq c$ is defined as $v \leq c$ since strict inequalities are not supported in the zone-based verification that is extended in this dissertation. The assignments are numerical formulae, $\mathcal{P}_\chi$, that satisfy the following grammar:

$$\chi \ ::= \ c \mid \infty \mid v \mid (\chi) \mid -\chi \mid \chi + \chi \mid \chi * \chi \mid \texttt{INT}(\phi) \mid \texttt{rate}(v) \mid [\chi_i, \chi_j]$$

where the function $\texttt{INT}(\phi)$ converts a Boolean expression that evaluates to **true** or **false** to 1 or 0, respectively, the function $\texttt{rate}(v)$ returns the current range of rates for the continuous variable v, and $[\chi_i, \chi_j]$ defines an interval of values that depends on the values of χ_i and χ_j when evaluated.

As a running example, consider a sequence of capacitors that are charged sequentially. The charging phase of the first capacitor is initiated by a switch sw_0. After 20μs of charging, a switch sw_1 is turned on, initiating the second capacitor's charging phase, and so on. When the switch sw_0 is turned off, the first capacitor starts discharging and the switch sw_1 is turned off, starting the second capacitor to discharge, and so on. Fig. 2.1 shows an LPN model of the i-th capacitor where the charging is some uncertain rate between 1mV/μs and 2mV/μs. The initial marking is $M_0 = \{p_{1,i}\}$ and is represented by the filled in circle. The values $V_i = 0$ and $V_i' = 0$ are the initial conditions for the voltage V_i. The variables sw_i and $sw_{(i+1)}$ are essentially Boolean variables with initial values of 0, representing **false**. The enabling conditions, delays, and variable assignments are in the curly braces, square brackets, and angle brackets, respectively. In this example, the delays are constants rather than bounds. Initially, the capacitor is not charging. When the signal sw_i is set to 1, charging is initiated by assigning the interval $[1, 2]$ to V_i' , which indicates the rate of V_i can be any rate between 1mV/μs and 2mV/μs. The capacitor is allowed to charge for 20μs (given as a delay on the transition $t_{2,i}$) before setting the variable $sw_{(i+1)}$ to 1. Once the charging is turned off, that is, when sw_i is set to 0, the capacitor begins to discharge at a rate of -1mV/μs. Finally, when the capacitor is fully discharged, the $t_{0,i}$ transition fires, setting the rate to zero.

Property checks are added to an LPN model by adding failure transitions T_f. Failure transitions are specialty transitions that, when fired, indicate a failure. The red transition, *tFail*, in Fig. 2.2 is an example of a failure transition. This transition checks the property that, when $sw_i \geq 1$ is **true**, if V_i is greater than 15 after 10 time units, then V_i is greater than 30 after an additional 10 time units. Failure transitions can be added directly to a model or to a separate LPN as in Fig. 2.2. When the failure transition is in a separate LPN, the LPN with the failure transition is called a *property LPN*. For example, the LPN in Fig. 2.2 is a property LPN that checks a property for Fig. 2.1.

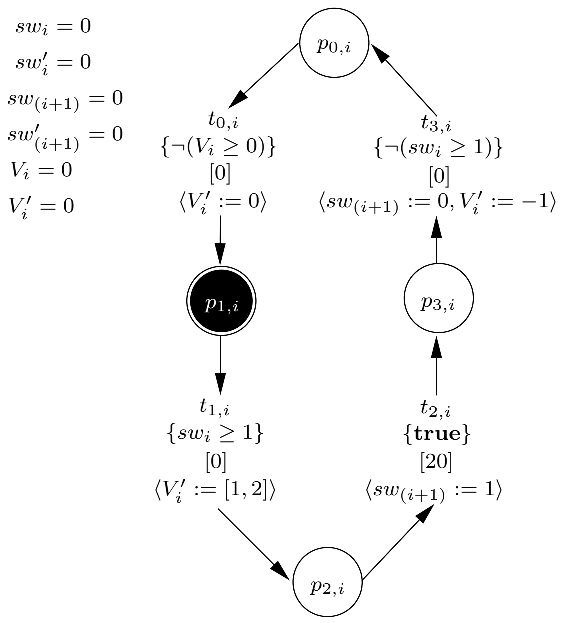


Figure 2.1: A model of a capacitor whose charging is turned on by sw_i. After a time delay of 20μs, the switch $sw_{(i+1)}$ is turned on (i.e., set to 1), initiating the charging of the next capacitor. Note that $[d]$ is used when $d_l(t) = d_u(t) = d$.

2.1.2 LPN Semantics

The state of an LPN consists of the set of marked places, the time each transition has been enabled, the delay range for each transition, the values of the continuous variables, the rate for each continuous variable, and the range of rates for each continuous variable. To determine which transitions can fire, the state also includes the truth value of each predicate, $v_i \geq c_i$. The set of all inequalities is denoted by $\mathcal{I}$. The state of an LPN is formally defined as a tuple $\sigma = \langle M, C, D, Q, R, RR, I\rangle$ where:

- $M \subseteq P$ is the set of marked places;
- $C : T \rightarrow \mathbb{Q}$ is the value of each transition's clock;
- $D : T \rightarrow \mathbb{Q} \times \mathbb{Q}$ is the delay range for each enabled transition;
- $Q : V \rightarrow \mathbb{Q}$ is the value of each continuous variable;
- $R : V \rightarrow \mathbb{Q}$ is the rate of each continuous variable;
- $RR : V \rightarrow \mathbb{Q} \times \mathbb{Q}$ is the range of rates for each continuous variable;
- $I : \mathcal{I} \rightarrow \{\mathbf{false}, \mathbf{true}\}$ is the value of each inequality.[1]

[1]Recording the inequality values is not strictly necessary. It is included as a matter of convenience so that an implementation does not need to calculate it repeatedly.

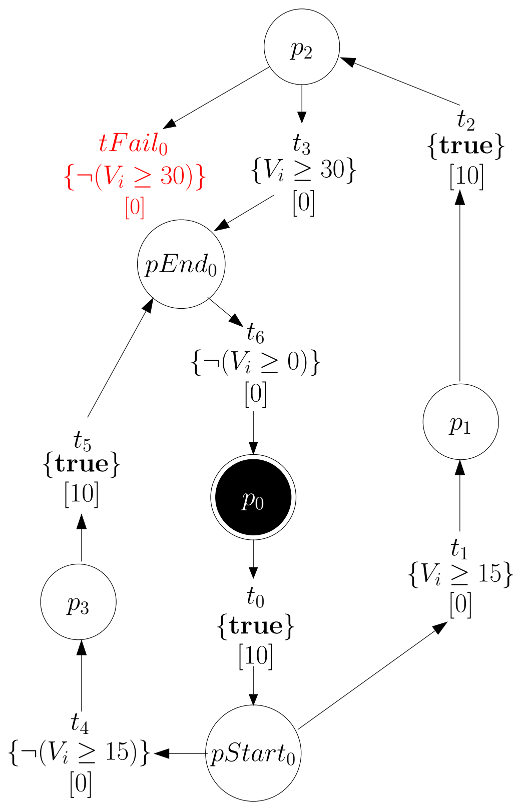


Figure 2.2: A property LPN for a capacitor stage in Fig. 2.1. When sw_i is 1, the property checks that V_i is above 15mV after 10μs, and then that V_i is more than 30mV after an additional 10μs. The property is violated if the fail transition, $tFail$, fires.

The collection of all states is Σ and the set of all enabled transition is $\mathcal{E}(\sigma)$, described further below. Again, the expressions $(a, b) \in \mathbb{Q} \times \mathbb{Q}$ are identified with intervals $[a, b]$. Furthermore, it is assumed that $a \leq b$, and the entire interval is either nonnegative (that is, $a \geq 0$) or is negative (that is $b < 0$). In the case of the delay, the ranges clearly must be nonnegative. Associated with the delays, D, and the range of rates, RR, are the functions $d_l(t)$, $d_u(t)$, $r_l(v)$ and $r_u(v)$, which access the lower and upper bounds of the ranges. Specifically, if $D(t) = [a, b]$, then $d_l(t) = a$ and $d_u(t) = b$. Similarly, if $RR(v) = [a, b]$, then $r_l(v) = a$ and $r_u(v) = b$.

In LPNs, the Boolean value of an inequality is evaluated in a nonstandard way at the boundary. For example, if the inequality is $v \geq 5$ and v is equal to 5, then the inequality is considered **true** if the rate of v is nonnegative and **false** if the rate is negative. The intuition for these semantics is that when the rate is negative and the variable is at the boundary, then the inequality is about to become **false** while when the rate is positive, then the inequality remains **true** as time progresses. Formally, the evaluation of an inequality is given by:

$$\texttt{evalInequalities}(\sigma)(v \geq c) = \begin{cases} R(v) \geq 0 & \text{if } Q(v) = c; \\ Q(v) \geq c & \text{otherwise.} \end{cases}$$

The initial state, σ_0, for an LPN consists of the initial markings, M_0, the initial value of each continuous variable, Q_0, the initial range of rates, R_0, an initial rate within this range, the initial value of the inequalities, and the time each transition has been enabled set to 0. The value $I(V_i \geq 0) = \mathbf{true}$ is an example of the nonstandard evaluation of the inequalities. Since the value of V_i is equal to 0 (the boundary for the inequality) and the rate is zero, the value of the inequality is **true**. The initial rate for each variable, v, is determined using the function $\texttt{resetRates}(RR)$, which is defined by:

$$\texttt{resetRates}(RR)(v) = r_l(v),$$

where $r_l(v)$ returns the lower bound rate. Similarly, $r_u(v)$ returns the upper bound. The initial state, σ_0, of the LPN in Fig. 2.1 has $M = \{p_{1,i}\}$, $Q(V_i) = Q(sw_i) = Q(sw_{(i+1)}) = 0$, $C(t_{0,i}) = C(t_{1,i}) = C(t_{2,i}) = C(t_{3,i}) = 0$, $D(t_{1,i}) = [0, 0]$, $R(V_i) = R(sw_i) = R(sw_{(i+1)}) = 0$, $RR(V_i) = RR(sw_i) = RR(sw_{(i+1)}) = [0, 0]$, $I(V_i \geq 0) = \mathbf{true}$, and $I(sw_i \geq 1) = \mathbf{false}$. The initial marking is indicated by the filled circles and the initial values of the variables are given in the upper left. Delays are only important for enabled transitions, hence the only delay given is for $t_{1,i}$. This convention is used for all states presented.

Requiring that the initial rate is given by the function $\texttt{resetRates}$ is not a limiting assumption, since the initial choice of rate is immaterial. If $\sigma_0 = \langle M, C, D, Q, R, I, RR \rangle$ has rates $R(v_i) = r_i$, $\sigma_0' = \langle M', C', D', Q', R', I', RR' \rangle$ has rates $R(v_i) = r_i'$ and $M = M', C = C', D = D', Q = Q', RR = RR'$, then the state σ_0' can be obtained from σ_0 by a sequence of rate change events E_i (formally defined below) such that for each i, the event E_i changes the rate of v_i from r_i to r_i'. So any future for σ_0' is a possible future for σ_0. Furthermore, interchanging the roles of σ_0 and σ_0' shows that any future of σ_0 is also a possible future of σ_0'.

The state σ can change to a new state $\sigma' = \langle M', C', D', Q', RR', R', I' \rangle$ by firing a transition, advancing time, or changing a rate. Collectively, transition firings, time advancements, and rate changes are known as *events*. A time advancement that results in the truth value of an inequality changing is an *inequality event*.

A transition $t \in T$ is *enabled* when all the places in its preset are marked (that is, when $\bullet t \subseteq M$) and the enabling condition on t evaluates to **true** (that is, when $\texttt{Eval}(En(t), \sigma)$ is **true** where the function $\texttt{Eval} : \mathcal{P}_\phi \times \Sigma \rightarrow \{\mathbf{false}, \mathbf{true}\}$ evaluates an expression given a state $\sigma \in \Sigma$). The set of all enabled transitions in a state σ is given by $\mathcal{E}(\sigma)$. When a transition becomes enabled, the delay assignment, $DA(t)$, is evaluated by $\texttt{EvalAssign} : \mathcal{P}_\chi \times \Sigma \rightarrow \mathbb{Q} \times \mathbb{Q}$. The transition must fire after the minimum delay $d_l(t)$ and before the maximum delay $d_u(t)$. The state σ' created as a result of firing the transition t is defined by:

$$\begin{aligned}
M' &= (M - \bullet t) \cup t\bullet; \\
\forall t \in T.C'(t) &= \begin{cases} 0 & \text{if } t \in \mathcal{E}(\sigma') \wedge t \notin \mathcal{E}(\sigma); \\ C(t) & \text{otherwise;} \end{cases} \\
\forall T \in \mathcal{E}(\sigma).D'(T) &= \begin{cases} \texttt{EvalAssign}(DA(t)) & \text{if } t \in \mathcal{E}(\sigma') \wedge t \notin \mathcal{E}(\sigma) \\ D(t) & \text{otherwise} \end{cases} \\
\forall v \in V.Q'(v) &= \text{EvalAssign}(\text{VA}(t, v), \sigma); \\
\forall v \in V.RR'(v) &= \texttt{EvalAssign}(RA(t, v), \sigma); \\
R' &= \texttt{resetRates}(RR'); \\
I' &= \texttt{evalInequalities}(\sigma').
\end{aligned}$$

When a transition is fired, the marking is updated and any assignments to the continuous variables and their rates are performed. The firing of a transition, t, causing a change from a state σ to a state σ' is denoted by $\sigma \xrightarrow{t} \sigma'$. As an example, consider the state σ_i, which is identical to σ_0 except $sw_i = 1$. The new state, σ_{i+1}, after $t_{1,i}$ that fires in Fig. 2.1 is $M = \{p_{2,i}\}$, $C(t_{0,i}) = C(t_{1,i}) = C(t_{2,i}) = C(t_{3,i}) = 0$, $D(t_{2,i}) = [20, 20]$, $Q(V_i) = 0$, $Q(sw_i) = 1$, $Q(sw_{(i+1)}) = 0$, $R(V_i) = 1$, $R(sw_i) = R(sw_{(i+1)}) = 0$, $RR(V_i) = [1, 2]$, $RR(sw_i) = RR(sw_{(i+1)}) = [0, 0]$, and $I(V_i \geq 0) = I(sw_i \geq 1) = \textbf{true}$.

Time can advance by any amount τ such that $\tau \leq \tau_{\max}(\sigma)$ where $\tau_{\max}(\sigma)$ is the largest allowable time advancement before an inequality changes value or a transition is forced to fire due to its maximum delay expiring.

$$\tau_{\max}(\sigma) = \min \begin{cases} \frac{c - Q(v)}{R(v)} & \forall (v \geq c) \in \mathcal{I}.I(v \geq c) \neq (R(v) \geq 0); \\ d_u(t) - C(t) & \forall t \in \mathcal{E}(\sigma). \end{cases}$$

In this equation, division by 0 is interpreted as yielding ∞. Thus, a zero rate variable does not limit the maximum time advancement. The new state σ' after advancing τ time units is given by:

$$\begin{aligned}
M' &= M;\\
\forall t \in T.C'(t) &= \begin{cases} C(t)+\tau & \text{if } t \in \mathcal{E}(\sigma);\\ 0 & \text{otherwise;}\end{cases}\\
D' &= D;\\
\forall v \in V.Q'(v) &= Q(v) + \tau * R(v);\\
RR' &= RR;\\
R' &= R;\\
I' &= \texttt{evalInequalities}(\sigma').
\end{aligned}$$

A time advancement by an amount τ is denoted by $\sigma \xrightarrow{\tau} \sigma'$. If a time advancement, τ, results in the change of truth value of an inequality in the set $\mathcal{I}$ (that is, the time advancement is an inequality event), then the event is denoted by $\sigma \xrightarrow{\tau,\mathfrak{I}} \sigma'$ where $\mathfrak{I}$ is the set of inequalities that change truth value. In addition, the rates are reset to the initial conditions (i.e., $R' = \texttt{resetRates}(RR'))$. Note that a time advancement results in an inequality event, if and only if, $\tau = \tau_{\max}(\sigma)$. In state σ_{i+1} above, $\tau_{\max} = 20$, since after 20 time units, the timer for the transition $t_{2,i}$ expires. The new state, σ_{i+2}, after a time advancement of 10 time units, is $M = \{p_{2,i}\}$, $C(t_{0,i}) = C(t_{1,i}) = C(t_{3,i}) = 0$, $C(t_{2,i}) = 10$, $D(t_{2,i}) = [20, 20]$, $Q(V_i) = 10$, $Q(sw_i) = 1$, $Q(sw_{(i+1)}) = 0$, $R(V_i) = 1$, $R(sw_i) = R(sw_{(i+1)}) = 0$, $RR(V_i) = [1, 2]$, $RR(sw_i) = RR(sw_{(i+1)}) = [0, 0]$, and $I(\,V_i \geq 0) = I(sw_i \geq 1) = \textbf{true}$.

The final type of state change is a rate change event. This event changes the rate of a single continuous variable $\hat{v} \in V$ to a new rate $\hat{r} \in RR(\hat{v})$. Since the truth value of an inequality depends on the rate, a rate change requires the updating of the inequalities involving $\hat{v}$. The corresponding new state is given by:

$$\begin{aligned}
M' &= M;\\
C' &= C\\
D' &= D;\\
Q' &= Q;\\
R'(v) &= \begin{cases} \hat{r} & \text{if } \hat{v} = v;\\ R(v) & \text{otherwise;}\end{cases}\\
RR' &= RR;\\
I' &= \texttt{evalInequalities}(\sigma').
\end{aligned}$$

After a rate change event for a continuous variable, the rate cannot change again until another nonrate event occurs. This restriction disallows the possibility of the state changing infinitely often solely due to the rates of continuous variables changing. Generality is not sacrificed in imposing this condition since the rate can be set to any value prior to the advancing of time, which is all that matters to the final trajectory of the continuous variable concerned. A rate change for a particular variable $\hat{v}$ to the rate $\hat{r}$ is denoted by $\sigma \xrightarrow{R(\hat{v})\leftarrow\hat{r}} \sigma'$. In the state σ_{i+2}, the state σ_{i+3} after changing the rate of V_i to a rate of 1.5 is given by $M = \{p_{2,i}\}$, $C(t_{0,i}) = C(t_{1,i}) = C(t_{3,i}) = 0$, $C(t_{2,i}) = 10$, $D(t_{2,i}) = [20, 20]$, $Q(V_i) = 10$, $Q(sw_i) = 1$, $Q(sw_{(i+1)}) = 0$, $R(V_i) = 1.5$, $R(sw_i) = R(sw_{(i+1)}) = 0$, $RR(V_i) = [1, 2]$, $RR(sw_i) = RR(sw_{(i+1)}) = [0, 0]$, and $I(V_i \geq 0) = I(sw_i \geq 1) = \textbf{true}$.

A trace in an LPN is a finite or infinite sequence $T = \sigma_0 \xrightarrow{E_0} \sigma_1 \xrightarrow{E_1} \sigma_2 \ldots$ where σ_0 is the initial state $\langle M_0, C_0, Q_0, \texttt{resetRates}(R_0), I_0, R_0\rangle$. The initial values for the inequalities is given by $I_0(v \geq c) = (Q_0(v) > c) \vee ((Q_0(v) = c) \wedge (r \geq 0))$ for all $(v \geq c) \in \mathcal{I}$ where r is the rate chosen by $\texttt{resetRates}(R_0)$. Each E_i is either a transition firing, a time advancement, or a rate change. The previous state before an event E_i and the successor state after an event E_i are given by $P(E_i)$ and $S(E_i)$, respectively. Thus $P(E_i) \xrightarrow{E_i} S(E_i)$. A trace fragment $\hat{T}$ is a finite or infinite sequence $\hat{T} = \sigma_i \xrightarrow{E_i} \sigma_{i+1} \xrightarrow{E_{i+1}} \sigma_{i+2} \ldots$ such that $\hat{T}$ can be extended to a trace $T = \sigma_0 \xrightarrow{E_0} \sigma_1 \ldots \sigma_i \xrightarrow{E_i} \sigma_{i+1} \xrightarrow{E_{i+1}} \sigma_{i+2} \ldots$. The set of all traces is denoted by $\mathbb{T}$. An example trace for the LPN in Fig. 2.1 is:

$$\sigma_0 \xrightarrow{t_{1\_i}} \sigma_1 \xrightarrow{10} \sigma_2 \xrightarrow{R(v)\leftarrow 1.5} \sigma_3.$$

For verification, LPNs use failure transitions that signal the failure of a system. Thus, a system passes verification if the set all possible transition sequences does not contain a sequence that leads to a failure transition firing and fails otherwise. So, it is enough to just consider all possible sequences of transition firings. This fact suggests that for a trace fragment $\hat{T} = \sigma_i \xrightarrow{E_i} \sigma_{i+1} \xrightarrow{E_{i+1}} \sigma_{i+2} \ldots$, what is really important is only the transition events and not the entire sequence of events $(E_j)_{j\geq i}$, which may also include rate and inequality events. To extract only the sequence of transition events from a trace fragment $\hat{T}$, define $subTran(\hat{T})$ to be the subsequence consisting of only the transition events in the sequence of events $(E_j)_{j\geq i}$. Formally, $sub_t(\hat{T})$ is the subsequence $(E_{j_k})_{k\geq 0}$ of $(E_j)_{j\geq i}$ satisfying the following two conditions:

- for all k, the event (E_{jk}) is a transition event;
- for all transition events E_j such that $j \geq i$, there exists a k such that $(E_{jk}) = E_j$.

It is important to note that since $sub_t(\hat{T})$ is defined as a subsequence of $(E_j)_{j\geq i}$, the transitions that occur in $sub_t(\hat{T})$ appear in the same order as in the original sequence $(E_j)_{j\geq i}$.

The definition of $sub_t(\hat{T})$ leads to an equivalence relation on traces defined as follows. If two traces T_1 and T_2 are such that $sub_t(T_1) = sub_t(T_2)$, then T_1 and T_2 are called *transition equivalent*, denoted $T_1 \sim_T T_2$. To verify whether an LPN has a failure (equivalently, to determine if a failure transition fires), it is enough to consider one representative per equivalence class in $\mathbb{T}/\sim_T$.

2.2 LEMA

LEMA is a tool for the formal verification of AMS circuits and has a tool flow as shown in Fig. 2.3. LEMA takes the transistor-level SPICE simulation traces from a traditional analog circuit verification approach and a set of discrete thresholds. It then applies a model generator to produce an LPN (Section 2.1). The properties that LEMA verifies can be provided using LAMP (described in Chapter 3), which is a simple, intuitive language for expressing AMS circuit properties [37, 66, 69]. LEMA includes a property compiler that can convert a LAMP property into an LPN. The model and property LPNs can then be combined in order to check that the model satisfies the property. This checking can be done either through simulation or model checking. For simulation, LEMA includes a translator that can convert LPNs into a SystemVerilog model that can then be simulated using a standard SystemVerilog simulator [17]. Formal verification can also be performed by LEMA using one of four model checkers: an exact BDD model checker [112], a SMT bounded model checker [112], a conservative model checker that uses *zones* [72], and a new conservative model checker that uses *octagons* (Chapter 5). All four model checkers provide a pass or fail result and, in the case of failure, provide a failure trace.

Subsection 2.2.1 describes the model generator. Subsection 2.2.2 presents the translator to SystemVerilog. Finally, Subsection 2.2.3 briefly describes LEMA's previous three model checkers.

2.2.1 Model Generator

In order to use formal verification on AMS designs, one needs a method for constructing formal models that takes into account the continuous nature of analog circuits. For example, before the *phase-locked loop* (PLL) in Fig. 2.4 can be verified, one needs to be able to construct a formal model that takes into account the analog nature of the *voltage controlled oscillator* (VCO). A PLL is a circuit that outputs a clock signal that is in-phase with an input reference clock and whose frequency is a multiple of the reference clock's frequency. Although the VCO outputs a clock, the VCO is a purely analog circuit that outputs a clock

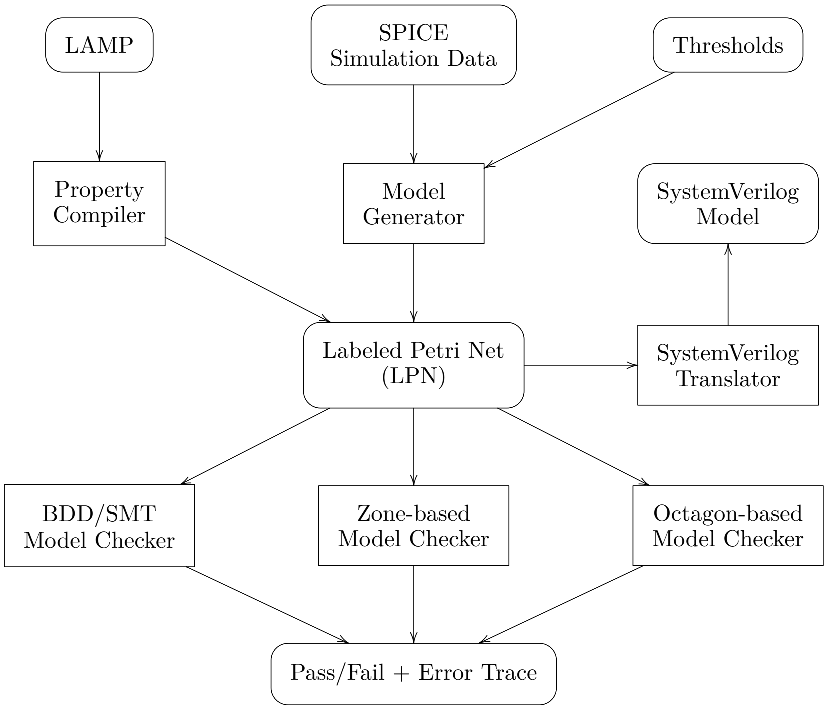


Figure 2.3: `LEMA`'s tool flow.

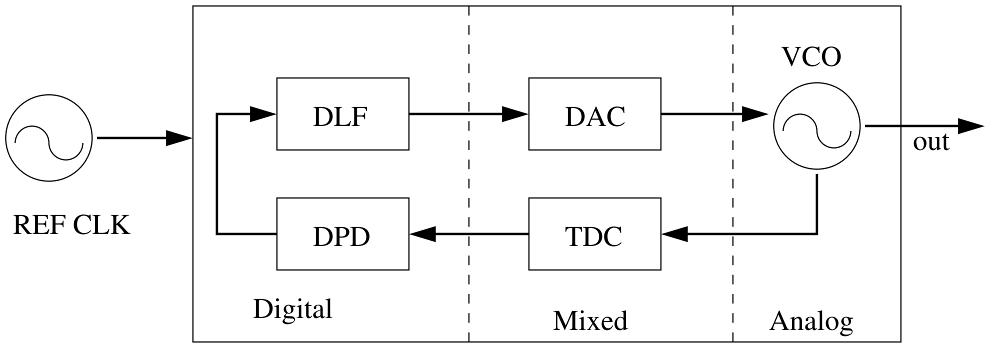


Figure 2.4: Digitally-intensive AMS design of a PLL.

whose frequency depends on an input voltage. An example LPN generated by `LEMA` for the VCO is shown in Fig. 2.5.

Creating LPNs by hand is a tedious process, and it is not easy to convince AMS designers to do. Consequently, `LEMA` provides a model generator that takes as input the more familiar transistor-level SPICE simulations together with some threshold values and automatically constructs an LPN model [17, 66, 68, 73]. The thresholds divide the space of continuous variables into regions. These regions become the places and the boundaries between the regions become the enabling conditions on the transitions. Furthermore, the model gener-

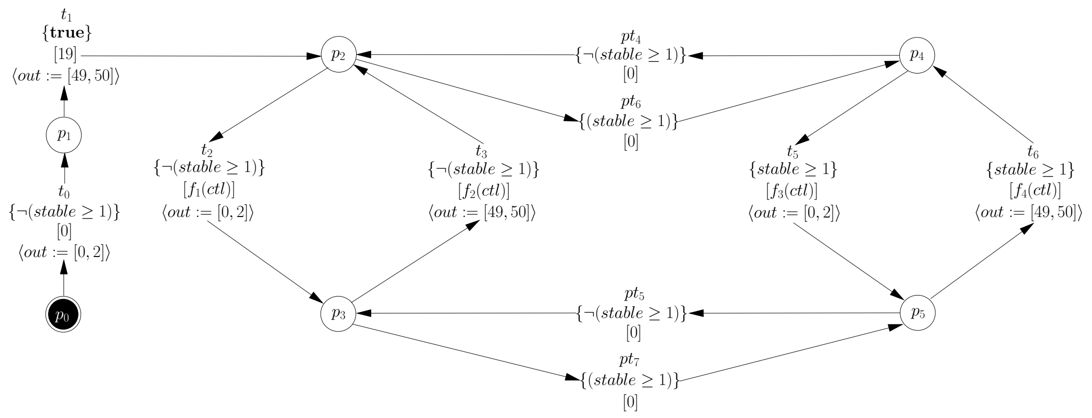

t_1
{true}
[19]
$\langle out := [49, 50]\rangle$
p_1
t_0
$\{\neg(stable \geq 1)\}$
[0]
$\langle out := [0, 2]\rangle$
p_0
p_2
t_2
$\{\neg(stable \geq 1)\}$
$[f_1(ctl)]$
$\langle out := [0, 2]\rangle$
t_3
$\{\neg(stable \geq 1)\}$
$[f_2(ctl)]$
$\langle out := [49, 50]\rangle$
p_3
pt_4
$\{\neg(stable \geq 1)\}$
[0]
pt_6
$\{(stable \geq 1)\}$
[0]
pt_5
$\{\neg(stable \geq 1)\}$
[0]
pt_7
$\{(stable \geq 1)\}$
[0]
p_4
t_5
$\{stable \geq 1\}$
$[f_3(ctl)]$
$\langle out := [0, 2]\rangle$
t_6
$\{stable \geq 1\}$
$[f_4(ctl)]$
$\langle out := [49, 50]\rangle$
p_5

Figure 2.5: LPN model for a VCO.

ator can identify discrete transitions and assign an appropriate delay.

The VCO model shown in Fig. 2.5 is generated using a set of three traces providing the frequency for three separate voltage values. LEMA creates a discrete variable *out* representing the output clock and adds delay functions $f_1(ctl)$, $f_2(ctl)$, $f_3(ctl)$ and $f_4(ctl)$, which vary based on the input control voltage. These functions produce a linear interpolation between the points of observation in the provided simulation traces. The delay functions are given by:

$$\begin{aligned}
f_1(ctl) &= ((ctl \geq 2) \wedge \neg(ctl \geq 3)) * [(ctl * (-2) + 17), (ctl * (-2) + 113)] \\
&\quad + ((ctl \geq 3) \wedge \neg(ctl \geq 4)) * [(ctl * (0) + 11), (ctl * (0) + 107)] \\
&\quad + (ctl \geq 4) * [(ctl * (0) + 11), (ctl * (0) + 107)] \\
f_2(ctl) &= ((ctl \geq 2) \wedge \neg(ctl \geq 3)) * [(ctl * (-2) + 21), (ctl * (-3) + 26)] \\
&\quad + ((ctl \geq 3) \wedge \neg(ctl \geq 4)) * [(ctl * (-2) + 21), (ctl * (-2) + 23)] \\
&\quad + (ctl \geq 4) * [(ctl * (-2) + 21), (ctl * (-2) + 23)] \\
f_3(ctl) &= ((ctl \geq 2) \wedge \neg(ctl \geq 3)) * (ctl * (-2) + 19) \\
&\quad + ((ctl \geq 3) \wedge \neg(ctl \geq 4)) * (ctl * (-1) + 16) \\
&\quad + (ctl \geq 4) * (ctl * (-1) + 16) \\
f_4(ctl) &= ((ctl \geq 2) \wedge \neg(ctl \geq 3)) * (ctl * (-5) + 3) \\
&\quad + ((ctl \geq 3) \wedge \neg(ctl \geq 4)) * (ctl * (-2) + 21) \\
&\quad + ctl \geq 4) * (ctl * (-2) + 21)
\end{aligned}$$

The two states, $p4$ and $p5$, create the oscillations of the output clock. When the control voltage changes, the circuit cannot instantly respond with the appropriate frequency, as it takes some amount of time for the output to settle into the right value. During this time, the circuit is unstable and has a varying frequency of oscillation until it settles into the right value. This unstable behavior is represented in the model by the places p_2 and p_3 together with delay functions $f_1(ctl)$ and $f_2(ctl)$. When the control voltage changes, the model changes the *stable* signal to **false** (indicating the unstable phase) and one of the transitions pt_4 or pt_5 fires moving the model into the left diamond. After some time, the *stable* signal is changed to **true** (indicating the stable phase) and one of the transitions pt_6 or pt_7 fires moving the model into the right diamond. In order to construct this unstable period, LEMA also includes an algorithm for recognizing the unstable part of the oscillation provided in the simulation trace. This procedure is described in more detail in [17, 66].

2.2.2 Translator

In order to check a property using a system-level simulation, LEMA can encode an LPN in SystemVerilog. In SystemVerilog, places become logic variables and transitions become wires. A low signal in a logic variable implies that the place is not marked and a high signal indicates that it is marked. Initially, all places are set low, then after a delay, the initial places are marked to start the simulation. A transition fires by sending a pulse on the wire, that is, the transition wire is set high and then set back low. This process is handled by an assign statement whose delay is set by a custom function and an assignment composed of a conjunction of the marking needed for this transition and the transition's enabling condition. The custom *delay* function handles the setting of the wire high after suitable delay and resetting the wire low immediately after the transition occurs. Finally, an always statement is added that is triggered by the positive edge of the transition wire. The body of the always statement handles updating the state by setting the incoming places low, the outgoing places high, and making any necessary signal assignments. A portion of the VCO model could be translated as shown in Listing 2.1.

2.2.3 Model Checkers

Verification can also be performed using a model checker that determines all possible reachable states and whether or not a failure transition can occur. LEMA has three different model checkers. It has a BDD-based model checker that is exact, but it trades performance for memory efficiency [112]. It also has an SMT-based bounded model checker that scales better, but it can only prove there are no failure transitions in a specified number of iterations [112]. Finally, it has a conservative zone-based model checker that lies somewhere between the SMT and BDD model checkers [72]. While it is not exact, it has better performance than the BDD model checker, and it can prove that failure transitions never

```
'timescale 1ps/1fs
module VCO(input real ctl,input real stable,output real out);
  wire t0,t1,t2,t3,t4,t5,t6;
  wire pt4,pt5,pt6,pt7;
  logic p0,p1,p2,p3,p4,p5;
  initial begin
    p0=0; p1=0; p2=0; p3=0; p4=0; p5=0;
    #1 p0 = 1; // initial marking
  end
  assign #(delay(~t0,0)) t0=(p0 && ~(stable>=1));
            ...
  always@(posedge t0) begin
    p0 = 0; p1 = 1; out = uniform(0,2);
  end
endmodule
```

Listing 2.1: Portion of the SystemVerilog for the VCO model.

fire which the SMT model checker cannot. With all three methods, LEMA provides a pass or fail answer and in the case of failure, a failure trace is provided.

In order to perform the BDD and SMT based methods, LEMA constructs a *symbolic model* of the given LPNs. The symbolic model consists of three pieces: an *invariant*, a set of *possible rates*, and a set of *guarded commands*. The invariant ensures that any states considered are reachable when ignoring the continuous variables and that the time elapsed since a transition was enabled does not exceed the maxium allowable delay for that transition. The possible rates simply encode the possible rate assignments. Finally, the guarded commands indicate which transitions are enabled and the effect on the state when a transition fires.

After constructing the symbolic model, LEMA can then proceed with the BDD model checker. In this method, facts about continuous variables are expressed using *hybrid separation logic* (HSL). These statements are then converted to a set of canonical representations which, in turn, are mapped to BDD variables. Next a fixed-point algorithm is performed to determine if a failure is possible. The algorithm starts with which states fail. Next, all states are found which could have occured just prior to the failure occuring. Then, time is evolved backwards as far as possible without violating the invariant. These two steps are then repeated until a fixed-point occurs, that is, until the states just found all occur in the set of states already found. Finally, the set of all states found is returned. These states represent the set of states that lead to failure when taken as an initial state.

Like the BDD-based model checker, the SMT-based checker uses the symbolic model of the LPN. Basically, the SMT approach is to construct a statement that asserts which states are reachable in some fixed number of steps n. One then adds a statement that indicates which states cause failures. Finally, the statement is passed to an SMT solver to determine if it is satisfiable, meaning there is some sequence of states (less than n) that leads to a failure. Thus, the process boils down to constructing the statement passed to the solver. The statement is constructed one step at a time. The first part of the statement is to assert the initial states. Then, in each following iteration, a statement is constructed that asserts the invariant and asserts which next states are possible by elapsing time or firing transitions. Finally, a statement is constructed indicating when a failure occurs. The statement sent to the SMT solver is then constructed by combining these statements.

LEMA's last model checker prior to this dissertation is a zone-based model checker that uses a form of *reachability analysis*. Zones are a subset of Euclidean space formed by intersecting half-planes associated with equations of the form $v \leq a$, $v \geq a$, or $v_i - v_j \leq a$ where v, v_i, and v_j are variables and a is a constant. In reachability analysis, one finds all possible states that are reachable from the initial states. Of course, with continuous variables, the state space is infinite; however, the zones allow for a finite representation. The basic algorithm is a depth first search. The algorithm starts with the initial state set and finds all possible events. An event is chosen and fired. Then, the resultant state set is found, time is allowed to move forward as far as possible without causing another event, and then all possible events are found again. This process continues until one reaches a state set found before or no events are possible. At this point, the algorithm backs up to the previous state set and another event is chosen. The algorithm ends when all possible state sets have been found. Initially, zone-based methods were used to verify timed models such as *timed automata* (TAs) [10, 19, 30] and *timed Petri nets* (TPNs) [43]; however, LEMA uses warping [72] to allow zones to be applied to nonrate one continuous variables in addition to clock variables. With warping, variables are scaled by their rate, turning them into rate-one variables and then the resulting subset is over-approximated by a zone. Building on this algorithm, Chapter 4 adds the ability to handle ranges of rates and Chapter 5 makes zones more accurate by allowing additional types of constraint.

CHAPTER 3

LAMP

This chapter introduces the Language for Analog/Mixed-Signal Properties (LAMP) to provide AMS designers with a easier, more intuitive property language to use. To demonstrate the utility of LAMP, this chapter describes how it can be used to specify verification properties for a *phase interpolator* (PI) and *voltage controlled oscillator* (VCO). In particular, the property for the PI that is verified is that it changes to the appropriate phase for a given control signal. The property for the VCO is that the appropriate phase occurs after a suitable settling time. LAMP is incorporated into our AMS verification tool `LEMA` (Section2.2), which uses LPNs (Section 2.1) as its primary model for verification. Accordingly, `LEMA` includes a compiler for the language, which converts statements into a property LPN that can be combined with a model LPN for the AMS circuit, and then model checking techniques can be performed to check that the AMS circuit satisfies the property of interest.

This chapter is organized as follows. Section 3.2 introduces the PI and VCO circuits, which are used as a motivating examples. Section 3.3 describes LAMP and sketches how a property in LAMP is compiled by `LEMA` into an LPN. Section 3.4 shows the results of using LAMP for the verification of a PI and a VCO circuit, and finally, Section 3.6 gives the conclusions and future work.

3.1 Related Work

It is well-known that the process of verifying analog circuits is not nearly as automated as its digital cousin. The difficulty is exacerbated when these areas are combined to create AMS circuits. To address this, there has been significant recent interest in developing formal approaches for verifying AMS circuits [117]. In order to apply formal verification approaches, such as *model checking* or *monitors*, it is necessary to create or extend formal languages to describe the time-dependent properties of AMS designs. Several languages have been proposed, which for the most part fall into two categories. They are either inspired by temporal logics like *linear temporal language* (LTL) [13, 97] and *computational* *tree logic* (CTL) [13] or have a grammar closer to a programming language, similar to SVA [2].

A few examples of languages inspired by LTL/CTL are *metric temporal logic* (MTL [64]), *metric interval temporal logic* (MITL [9]), *signal temporal logic* (STL [79]), and *ana CTL* [29]. MTL augments LTL with timing [64], but unfortunately, it is not decidable in general [9]. MITL creates a balance between decidability and expressiveness by relaxing the continuous model of time [9]. In [75] and [78], the authors study the use of MITL in online monitoring while in [79], the authors extend MTL to create STL. As an extension of CTL, Ana CTL adds statements to match analog signals to allow CTL to be used in the verification of analog circuits. These languages have been difficult to convince the analog and AMS community to use in practice since the formalism is so foreign to designers, and it is often difficult to determine which expression is needed to capture a desired property.

In addition to languages being built from LTL or CTL-like formalisms, several languages have been proposed taking inspiration from assertion languages. A prominent example is the *property specification language* (PSL [1, 35]), which can be used for specifying properties both in the digital and AMS domains. For example, [20] uses PSL to express temporal properties of AMS designs, [62] uses PSL to describe the behavior of the DDR2 memory protocol in terms of assertions, and [5] extends PSL to better combine the language with verifying SRE circuit descriptions. Furthermore, [105] extends PSL to the *analog specification language* (ASL) to better describe continuous state space properties of AMS designs. As an alternative to PSL, [104] uses SystemVerilog [2] to describe the inherent asynchronous behavior in synchronous circuits, and [54] introduces *real-time SVA* (RT-SVA) as an extension of SVA adding more direct support for continuous assertions. Despite their generality and their aim to provide designers with a more program-like language, it is still difficult to craft a particular property of interest when using these languages.

3.2 Motivating Examples

This section introduces the two circuits used to demonstrate LAMP's capabilities. The first circuit, the PI, is used in [66, 69] to motivate why LAMP was initially developed. This discussion is included here for context. In particular, the initial PI verification results of [69] serve as motivation for the **always** block extension presented in this chapter. The second circuit, the VCO, is used to illustrate how the LAMP extensions can be utilized. This section is separated into two parts: Section 3.2.1 introduces the models and Section 3.2.2 discusses the properties.

3.2.1 Models

A circuit implementation of a PI circuit is shown in Fig. 3.1. A PI circuit is a circuit that takes an input clock, *phi*, and according to the value of a control signal, *ctl*, produces a shifted output clock, *omega*. Fig. 3.2 gives an example of an LPN model generated by `LEMA` from simulation data for a PI circuit with four different phase shifts. For the example LPN shown in Fig. 3.2, transition, t_0, is enabled, and it fires immediately since its delay assignment is 0. This moves the marking from place p_0 to p_1 and changes *omega* to a value between 194 and 195. At this point the circuit waits until *phi* goes high (i.e., above 0), then checks the value of the *ctl* signal to determine which uniform statement is evaluated in the delay condition. It then waits the specified amount of time before firing transition t_1 to set *omega* to a value between 245 and 246, and moves the marking to place p_2. Note that in Fig. 3.2 all the values are integers. During the model generation process, all of the continuous variables are scaled (with the same factor) to ensure that all the values are integers. Similarly, the time is scaled by a factor to ensure integer values as well. These two scaling factors are returned to the user to adjust their properties accordingly. For the LPN in Fig. 3.2, the time units are in picoseconds and the values of *phi*, *omega*, and *ctl* are in 10^{-2} volts. While this scaling is not strictly necessary, it does simplify the implementation.

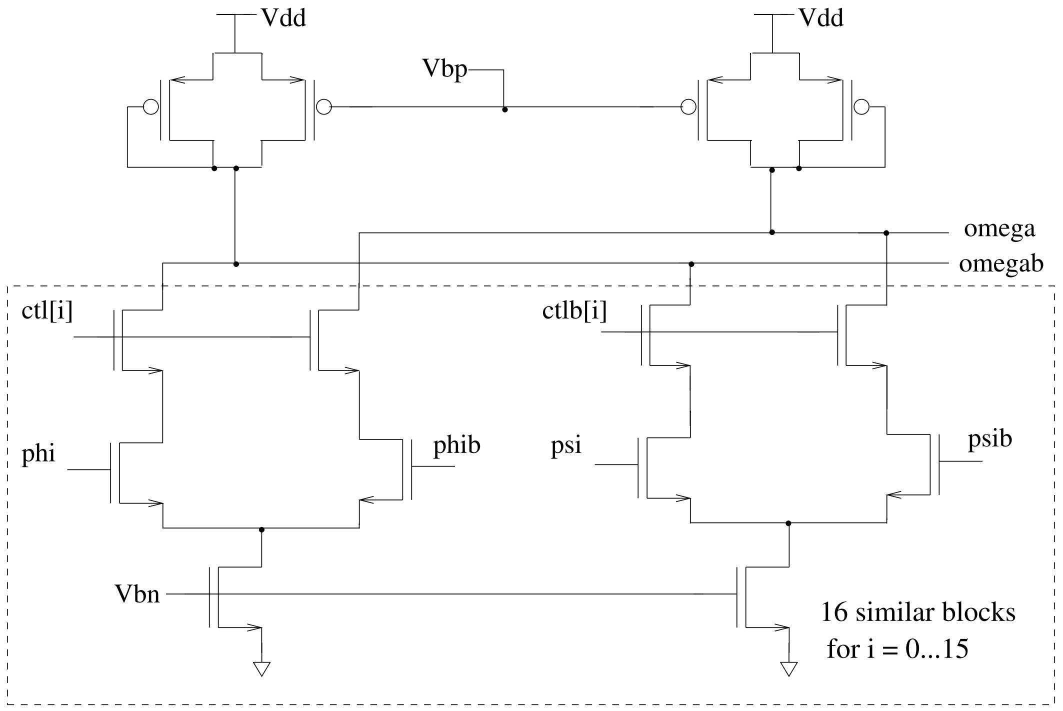


Figure 3.1: Phase interpolator circuit implementation.

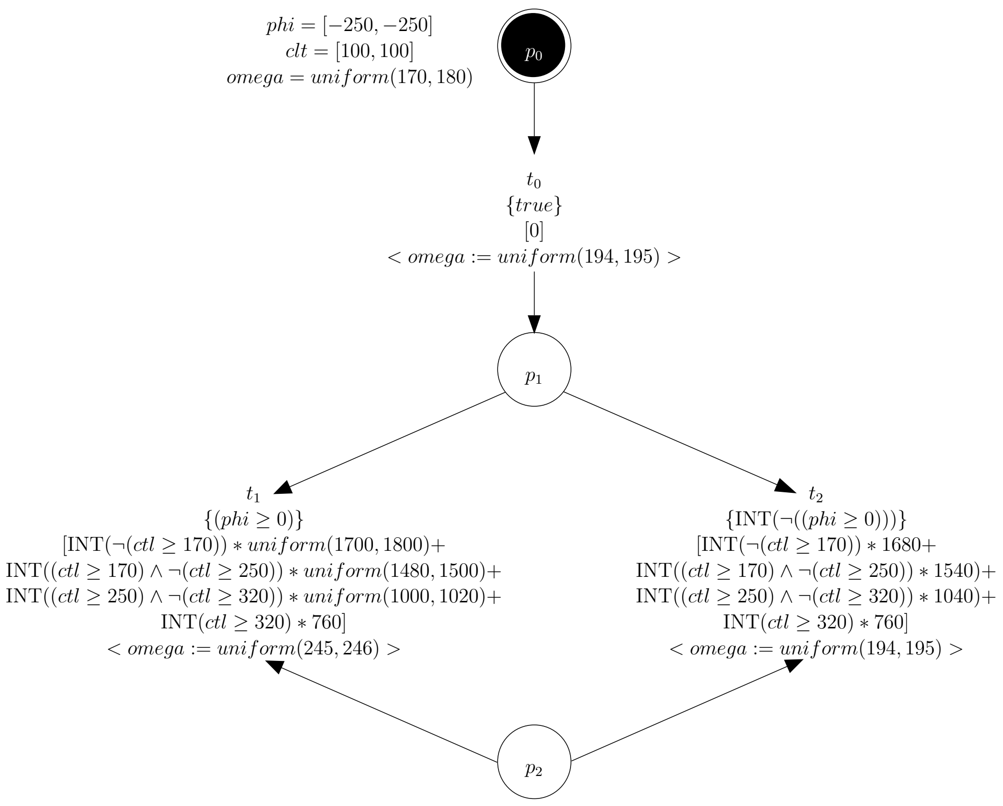
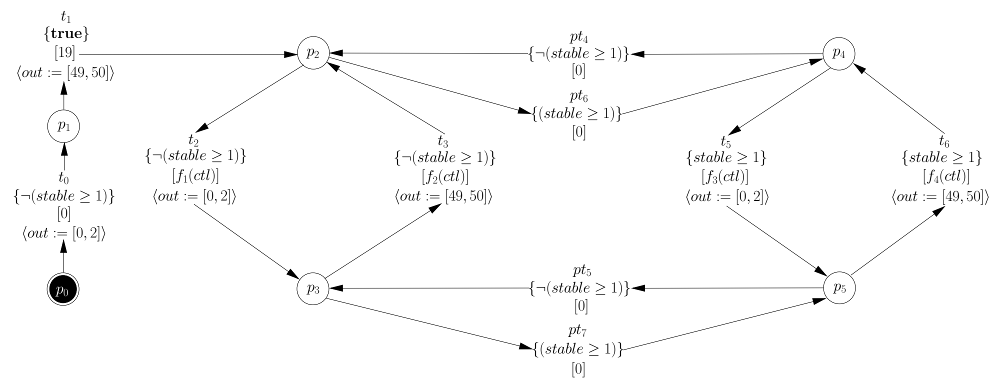

$phi = [-250, -250]$
$clt = [100, 100]$
$omega = uniform(170, 180)$
p_0
t_0
$\{true\}$
$[0]$
$< omega := uniform(194, 195) >$
p_1
t_1
$\{(phi \geq 0)\}$
$[\mathrm{INT}(\neg(ctl \geq 170)) * uniform(1700, 1800) +$
$\mathrm{INT}((ctl \geq 170) \wedge \neg(ctl \geq 250)) * uniform(1480, 1500) +$
$\mathrm{INT}((ctl \geq 250) \wedge \neg(ctl \geq 320)) * uniform(1000, 1020) +$
$\mathrm{INT}(ctl \geq 320) * 760]$
$< omega := uniform(245, 246) >$
t_2
$\{\mathrm{INT}(\neg((phi \geq 0)))\}$
$[\mathrm{INT}(\neg(ctl \geq 170)) * 1680 +$
$\mathrm{INT}((ctl \geq 170) \wedge \neg(ctl \geq 250)) * 1540 +$
$\mathrm{INT}((ctl \geq 250) \wedge \neg(ctl \geq 320)) * 1040 +$
$\mathrm{INT}(ctl \geq 320) * 760]$
$< omega := uniform(194, 195) >$
p_2

Figure 3.2: Generated LPN model of a PI circuit.

The second example this chapter explores is a VCO, which is a circuit that outputs a clock signal, *out*, whose frequency changes according to the voltage level of a control signal, *ctl*. A model for a VCO is shown in Figs. 3.3 and 3.4. This model is generated using `LEMA`'s model generator on simulation data for three control voltages 2 V, 3 V, and 4 V together with interpolation between these values as described in Section 4 of [66]. A sample trace is shown in Fig. 3.5. The model consists of two phases: an *unstable* phase signified by $stable = 0$ and a *stable* phase signified by $stable = 1$. The unstable state is modeled by the p_2, t_2, p_3, and t_3 loop, while the stable phase is modeled by the p_4, t_5, p_5, and t_6 loop. When the control signal changes, it takes the system some amount of time before the signal settles into the expected phase. This transient behavior is modeled by setting the *stable* signal to 0 when the control changes and then setting *stable* signal to 1 after some delay, signifying a shift to the stable phase. The setting of the *stable* signal is handled by the model in Fig. 3.4. The delays in Fig. 3.3 are:



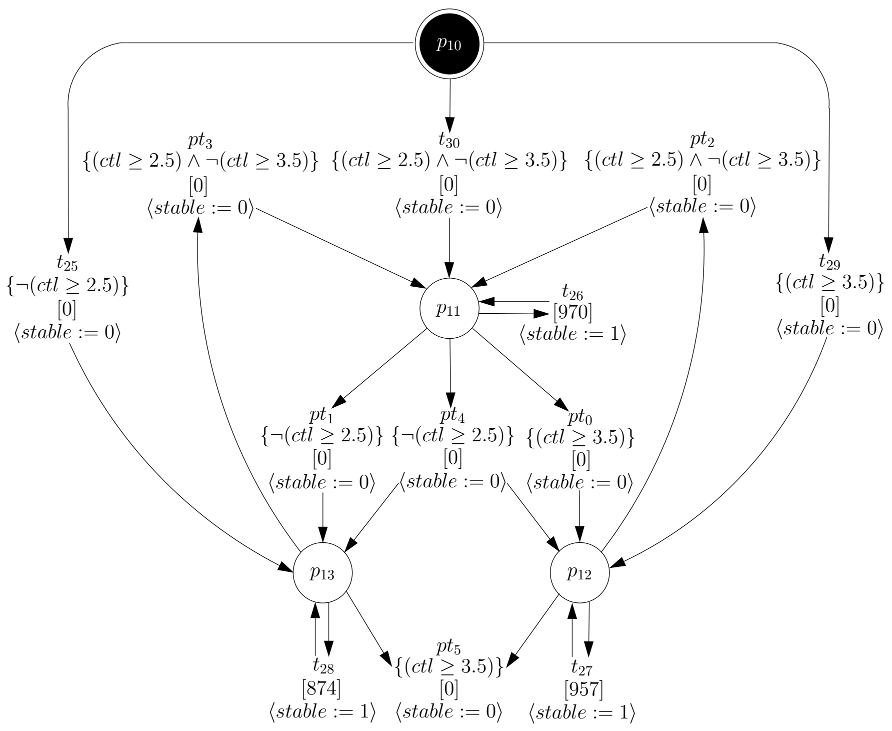
t_1
{true}
[19]
$\langle out := [49,50]\rangle$
p_1
t_0
$\{\neg(stable \geq 1)\}$
[0]
$\langle out := [0,2]\rangle$
p_0
p_2
t_2
$\{\neg(stable \geq 1)\}$
$[f_1(ctl)]$
$\langle out := [0,2]\rangle$
t_3
$\{\neg(stable \geq 1)\}$
$[f_2(ctl)]$
$\langle out := [49,50]\rangle$
p_3
pt_4
$\{\neg(stable \geq 1)\}$
[0]
pt_6
$\{(stable \geq 1)\}$
[0]
pt_5
$\{\neg(stable \geq 1)\}$
[0]
pt_7
$\{(stable \geq 1)\}$
[0]
p_4
t_5
$\{stable \geq 1\}$
$[f_3(ctl)]$
$\langle out := [0,2]\rangle$
t_6
$\{stable \geq 1\}$
$[f_4(ctl)]$
$\langle out := [49,50]\rangle$
p_5

Figure 3.3: Model of a VCO with a stable and unstable phase.

$$\begin{aligned}
f_1(ctl) = {} & ((ctl \geq 2) \wedge \neg(ctl \geq 3)) * [(ctl * (-2) + 17), (ctl * (-2) + 113)] \\
& + ((ctl \geq 3) \wedge \neg(ctl \geq 4)) * [(ctl * (0) + 11), (ctl * (0) + 107)] \\
& + (ctl \geq 4) * [(ctl * (0) + 11), (ctl * (0) + 107)] \\
f_2(ctl) = {} & ((ctl \geq 2) \wedge \neg(ctl \geq 3)) * [(ctl * (-2) + 21), (ctl * (-3) + 26)] \\
& + ((ctl \geq 3) \wedge \neg(ctl \geq 4)) * [(ctl * (-2) + 21), (ctl * (-2) + 23)] \\
& + (ctl \geq 4) * [(ctl * (-2) + 21), (ctl * (-2) + 23)] \\
f_3(ctl) = {} & ((ctl \geq 2) \wedge \neg(ctl \geq 3)) * (ctl * (-2) + 19) \\
& + ((ctl \geq 3) \wedge \neg(ctl \geq 4)) * (ctl * (-1) + 16) \\
& + (ctl \geq 4) * (ctl * (-1) + 16) \\
f_4(ctl) = {} & ((ctl \geq 2) \wedge \neg(ctl \geq 3)) * (ctl * (-5) + 3) \\
& + ((ctl \geq 3) \wedge \neg(ctl \geq 4)) * (ctl * (-2) + 21) \\
& + (ctl \geq 4) * (ctl * (-2) + 21)
\end{aligned}$$

3.2.2 Properties

A simple property to verify for the PI is that the phase shift of the output clock, *omega*, generated by the circuit matches the desired phase shift for the given control signal value. This verification property can be expressed as an LPN, as shown in Fig. 3.6 [17, 66]. The LPN accomplishes this by first waiting for *phi* to go high, which marks the places $pCheckMin$ and $pCheckMax$. At this point, one of the $tMin$ and one of the $tMax$ transitions are enabled depending on the value of *ctl*. If the output clock, *omega*, goes high before the delay on the appropriate $tMin$ transition passes, then the fail transition, $tFailMin$, fires indicating that the phase shift is too small. On the other hand, if the appropriate $tMin$ fires, then $pCheck$ is marked, and the LPN waits for *omega* to go high.



p_{10}
pt_3 $\{(ctl \geq 2.5) \wedge \neg(ctl \geq 3.5)\}$ [0] $\langle stable := 0\rangle$
t_{30} $\{(ctl \geq 2.5) \wedge \neg(ctl \geq 3.5)\}$ [0] $\langle stable := 0\rangle$
pt_2 $\{(ctl \geq 2.5) \wedge \neg(ctl \geq 3.5)\}$ [0] $\langle stable := 0\rangle$
t_{25} $\{\neg(ctl \geq 2.5)\}$ [0] $\langle stable := 0\rangle$
p_{11}
t_{26} [970] $\langle stable := 1\rangle$
t_{29} $\{(ctl \geq 3.5)\}$ [0] $\langle stable := 0\rangle$
pt_1 $\{\neg(ctl \geq 2.5)\}$ [0] $\langle stable := 0\rangle$
pt_4 $\{\neg(ctl \geq 2.5)\}$ [0] $\langle stable := 0\rangle$
pt_0 $\{(ctl \geq 3.5)\}$ [0] $\langle stable := 0\rangle$
p_{13}
p_{12}
t_{28} [874] $\langle stable := 1\rangle$
pt_5 $\{(ctl \geq 3.5)\}$ [0] $\langle stable := 0\rangle$
t_{27} [957] $\langle stable := 1\rangle$

Figure 3.4: Model for the process that changes the stable and unstable phases for Fig. 3.3.

If the delay on the appropriate *tMax* transition passes, then the *tMax* fail transition fires indicating that the phase shift is too large. However, if *omega* goes high first, then *pReset* is marked, and the LPN waits for *phi* to go low and high again before checking the next phase shift. When this LPN is combined with the LPN for the circuit and an LPN for the environment, reachability analysis can be performed to determine if a failure transition is possible [72].

As opposed to the model and environment LPNs, the property LPN must be constructed by hand, which is a tedious and error prone process. It is not very reasonable to require designers to formulate their properties in this way. Therefore, a more intuitive property language is needed that can readily be compiled into property LPNs for verification.

Before creating LAMP, some other existing AMS property language were considered. For example, this same property can be written in STL as follows:

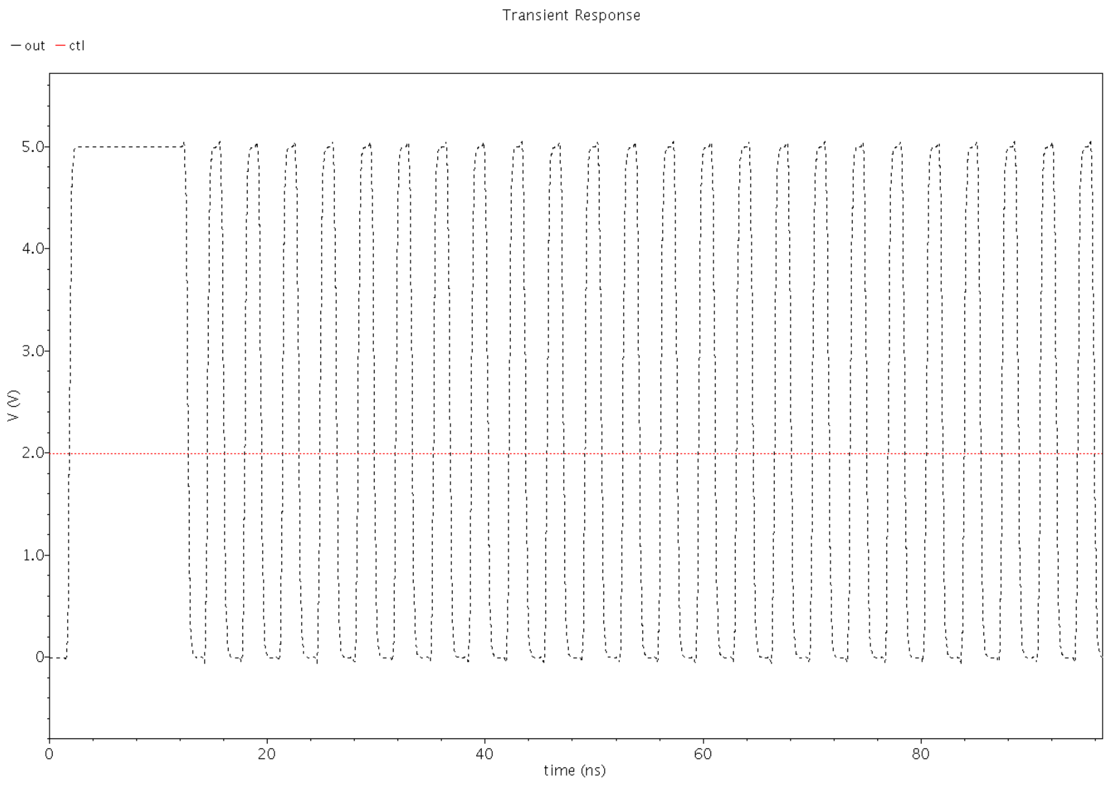


Figure 3.5: Typical simulation trace for a VCO.

$$
\begin{aligned}
&\Box(((\uparrow (phi \geq 0) \wedge (ctl \leq 1.7)) \rightarrow \\
&(\Box_{[0,1699]} omega < 2.2) \wedge (\Diamond_{[1699,1801]} omega > 2.2)) \\
&\vee((\uparrow (phi \geq 0) \wedge (ctl \geq 1.7) \wedge (ctl \leq 2.5)) \rightarrow \\
&(\Box_{[0,1479]} omega < 2.2) \wedge (\Diamond_{[1479,1501]} omega > 2.2)) \\
&\vee((\uparrow (phi \geq 0) \wedge (ctl \geq 2.5) \wedge (ctl \leq 3.2)) \rightarrow \\
&(\Box_{[0,999]} omega < 2.2) \wedge (\Diamond_{[999,1021]} omega > 2.2)) \\
&\vee((\uparrow (phi \geq 0) \wedge (ctl \geq 3.2)) \rightarrow \\
&(\Box_{[0,759]} omega < 2.2) \wedge (\Diamond_{[759,761]} omega > 2.2)))
\end{aligned}
$$

The $\Box$ notation indicates that the property is always checked. The statement $\uparrow$ waits for the positive edge of a Boolean expression. Thus, collectively the first part of the statement checks that *phi* goes high and *ctl* is below 1.7. If this condition is satisfied, then the statement $(\Box_{[0,1699]} omega < 2.2) \wedge (\Diamond_{[1699,1801]} omega > 2.2)$ is checked. The interval subscript on $\Box$ indicates that the statement $omega < 2.2$ must remain **true** for 1699 time units. The next part of the statement $(\Diamond_{[1699,1801]} omega > 2.2)$ requires that the

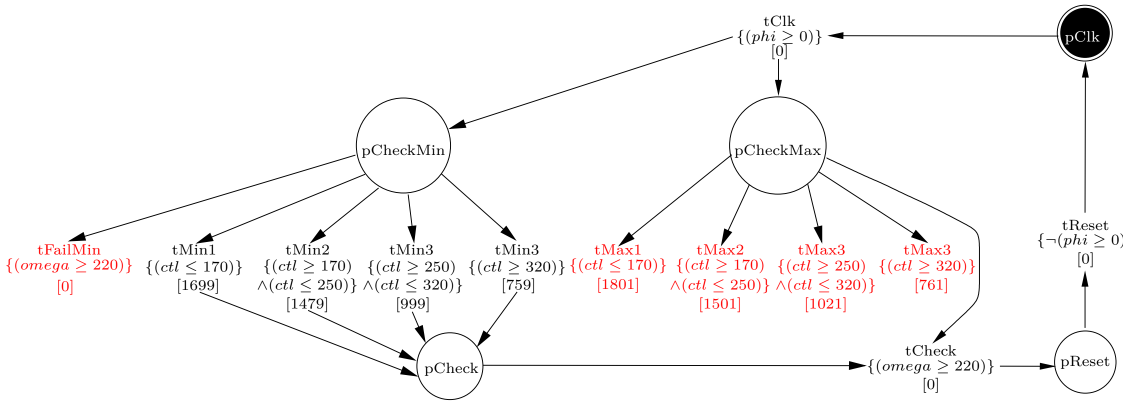


Figure 3.6: An LPN for the phase interpolator verification property.

statement *omega* $>$ 2.2 becomes **true** between 1699 and 1801 time units. The rest of the statements are similar.

Writing a specification in a temporal logic is quite removed from the environment designers commonly use, and thus can be difficult for a designer. To address this issue, this property can be written using STL-PSL as follows:

```
vprop PhaseInterpolator{
  PI1 assert:
    always((rise(a:phi >= 0) and (a:ctl <= 1.7))
    → (always[0,1699](a:omega < 2.2)) and
    (eventually[1699, 1801](a:omega > 2.2)) or
    (rise(a:phi >= 0) and (a:ctl >= 1.7) and (a:ctl <= 2.5))
    → (always[0, 1479](a:omega <2.2)) and
    (eventually[1479, 1501](a:omega > 2.2)) or
    (rise(a:phi >= 0) and (a:ctl >= 2.5) and (a:ctl <= 3.2))
    → (eventually[0, 999](a:omega < 2.2)) and
    (eventually[999, 1021](a:omega > 2.2)) or
    (rise(a:phi >= 0) and (a:ctl >= 3.2))
    → (eventually[0, 759](a:omega < 2.2)) and
    (eventually[759, 761](a:omega > 2.2)));
}
```

This version is less intimidating, but it still requires designers to learn some temporal logic semantics in order to correctly use the **always** and **eventually** statements. Furthermore, it is difficult to determine how to convert this type of language into the LPN with failure transitions that `LEMA` needs.

Another alternative is to write this property using RT-SVA [54] as shown below:

$$(phi \geq 0)[\sim> 1]\ \#\#0$$
$$(((ctl \leq 1.7)[\sim> 1]\ \#\#0$$

$$(!(omega > 2.2))[*1699 : 1801] \ \#\#1 \ (omega > 2.2))$$
$$or$$
$$(((ctl \geq 1.7) \&\& (ctl \leq 2.5))[\sim> 1] \ \#\#0$$
$$(!(omega > 2.2))[*1479 : 1501] \ \#\#1 \ (omega > 2.2])$$
$$or$$
$$(((ctl \geq 2.5) \&\& (ctl \leq 3.2))[\sim> 1] \ \#\#0$$
$$(!(omega > 2.2))[*999 : 1021] \ \#\#1 \ (omega > 2.2))$$
$$or$$
$$(((ctl \geq 3.2))[\sim> 1] \ \#\#0$$
$$(!(omega > 2.2))[*759 : 761] \ \#\#1 \ (omega > 2.2))) \ \#\#1$$
$$(phi < 0)[\sim> 1]$$

In RT-SVA, the expression [$\sim>$ 1] waits for the preceding Boolean expression to become **true**. Thus, the first line waits for the input clock *phi* to become nonnegative. The expression A ##0 B is a concatenation operator that indicates the next expression B should become **true** at some time overlapping when A is **true**. Therefore, the next part checks the value of *ctl* when *phi* goes high. Finally, the $A[*l : u]$ statement specifies that A must be **true** for between l and u time units. Therefore, the last part of the statement checks for a change in *omega* from a low to a high value within a specified amount of time. Considering this example, it again appears to be tricky and somewhat tedious to read and write verification properties in RT-SVA. It is also difficult to translate these properties into LPNs with failure transitions.

The VCO adds an additional complication. The property that this chapter considers is that the correct frequency is output for a given voltage level. The first difficulty arises from the circuits inability to instantly respond to start-up or a change in voltage. Specifically, when the circuit is first provided powered or the voltage changes, it takes some time before the circuit settles into a stable frequency. This behavior is shown by the first oscillation in Fig. 3.5, which is much longer than the rest of the oscillations. A second difficulty appears when the VCO is put in an environment that can nondeterministically change the input voltage. When the voltage is allowed to changed at any time, it is possible that a property will fail due to the check being performed on the previous voltage value instead of the current one.

To solve these two issues, one needs an ability to ignore the curve for some transient period and an ability to reset the property check when a signal like the voltage changes. It is not transparent how one would specify this behavior in the previous formalisms; moreover, the previous version of LAMP [66] is not able to handle these cases. The extensions to LAMP presented in Section 3.3 solve these problems by providing statements that can ignore the initial unstable period and can react to a changing environment.

3.3 LAMP

To make properties easy to read and write, LAMP uses a procedural approach where statements are checked in order. LAMP's statements are also chosen to match straightforward concepts, though the formal semantics are defined in terms of LPNs. Figure 3.7 shows the format for a property in LAMP. The property consists of a name, followed by variable declarations and LAMP statements. The statements of LAMP are as follows:

- **delay**(d) – wait for d time units. This statement is compiled into the LPN shown in Fig. 3.8(a).
- **wait**(b) – wait until the Boolean expression b becomes **true**. This statement is compiled into the LPN shown in Fig. 3.8(b). In this LPN, transition t_0 fires when b becomes **true**. There is no time limit which means that the firing of t_0 can wait as long as necessary for b to become **true**.
- **waitPosedge**(b) – wait for a positive edge on the expression b (i.e., **wait**($\sim b$); **wait**(b)). The LPN for this statement is shown in Fig. 3.8(c) and is the concatenation of the statements **wait**($\sim b$) and **wait**(b).
- **wait**(b, d) – wait at most d time units for the Boolean expression b to become **true**. This statement is compiled into the LPN shown in Fig. 3.8(d). If b is **false** initially, the failure transition, $tFail_0$, is enabled, but it has a delay of d time units. If during this time interval b becomes **true**, t_0 fires immediately, since it has 0 delay. If, however, b remains **false** for d time units, $tFail_0$ fires and a failure is recorded.
- **assert**(b, d) – ensures that the Boolean expression b remains **true** continuously for d time units. This statement is compiled into the LPN shown in Fig. 3.8(e). If b is **true** initially, the transition t_0 is enabled, but it has a delay of d time units. If during this

```
property <name> {
   <declarations>
   <statements>
}
```

Figure 3.7: Format for a LAMP property.

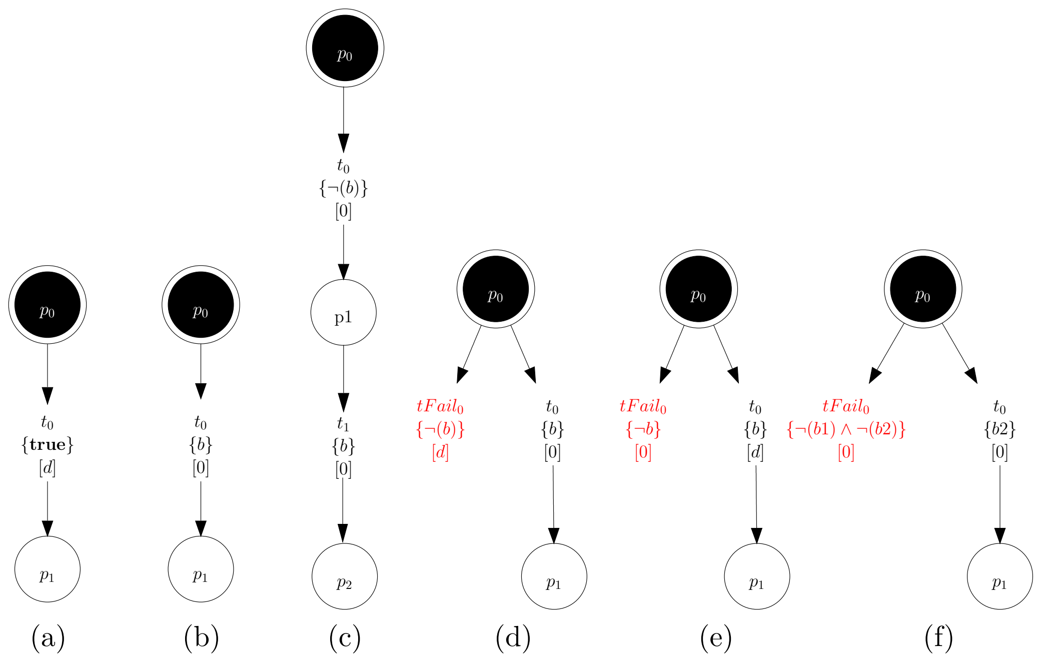


Figure 3.8: LPN translations for LAMP statements. LPN for (a) **delay**(d), (b) **wait**(b), (c) **waitPosedge**(b), (d) **wait**(b, d), (e) **assert**(b, d), and (f) **assertUntil**($b1, b2$) statements.

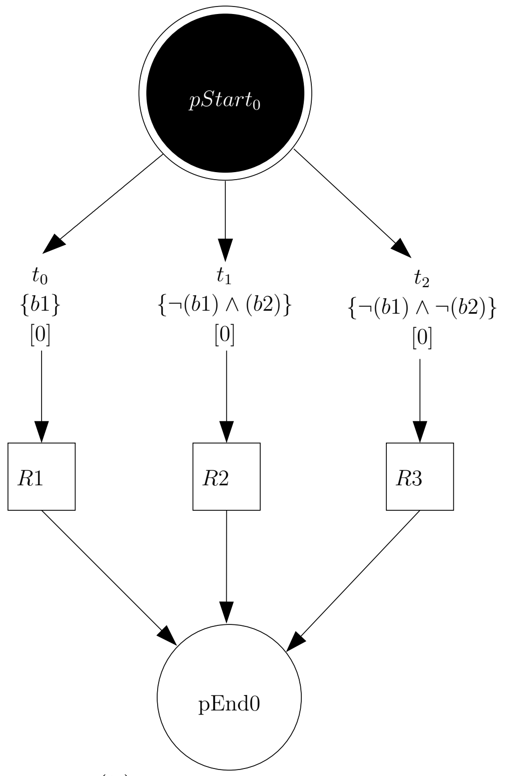
time interval b goes **false**, the failure transition, $tFail_0$, fires immediately indicating a failure. If, however, b remains **true** for d time units, t_0 fires.

- **assertUntil**($b1, b2$) – ensures that the Boolean expression $b1$ remains **true** until the Boolean expression $b2$ becomes **true**. This statement is compiled into the LPN shown in Fig. 3.8(f). In this LPN, the failure transition, $tFail_0$, fires if $b1$ and $b2$ are **false** simultaneously before $b2$ becomes **true**.

- The language also provides an **if-else** statement, as shown in Fig. 3.9.

- **always**(*conditionsList*) {*statements*} – continue to execute *statements* until one of the signals in the list of variables *conditionsList* changes, then break out. The generated LPN is shown in Fig. 3.10, assuming a list containing at least the variables a and b. First, transition t_0 fires and stores the current values of the variables in the list *conditionsList* in a set of new variables $\_a$, $\_b$, ... Then, the statements inside the **always** block continue to execute as long as the condition $alcond = (a = \_a) \wedge (b = \_b) \wedge \ldots$ remains **true**. If *alcond* becomes **false**, an exit transition fires leaving the loop. In particular, every transition in the **always** block has *alcond* added to the enabling condition while every place has an exit transition with $\neg alcond$. If the *conditionsList* is empty, then *alcond* is taken to be **true** and all the exiting transitions are removed.

```
if (b1) {
  R1
}
else if (b2) {
  R2
}
else {
  R3
}
```



(a) (b)

Figure 3.9: LPN translation for the ***if-else*** statement. (a) LAMP syntax for an ***if-else*** statement. (b) LPN for an ***if-else*** statement.

Note that the formal semantics of each of these statements is defined by the corresponding LPN given in Figs. 3.8, 3.9, and 3.10.

A property compiler for LAMP is incorporated into the `LEMA` verification tool. This compiler generates a property LPN from a written property as follows:

1. Create an LPN with the name of the property.

2. For each variable declaration, create a continuous or Boolean variable in the LPN.

3. For each statement, construct an LPN using the templates described above making the last place for each statement the same as the first place for the following statement.

4. When an **always** block is encountered, create a new variable $\_a$ for each variable a in the condition list. Add the transition that stores the variables and add the starting place for the **always** block. Construct all the interior statements according to 3 while adding the *alcond* to each transition constructed. A transition is added from the last place to the starting place with the *alcond* as its enabling condition. Finally, add the exit place and the exit transitions with enabling condition $\neg$*alcond*.

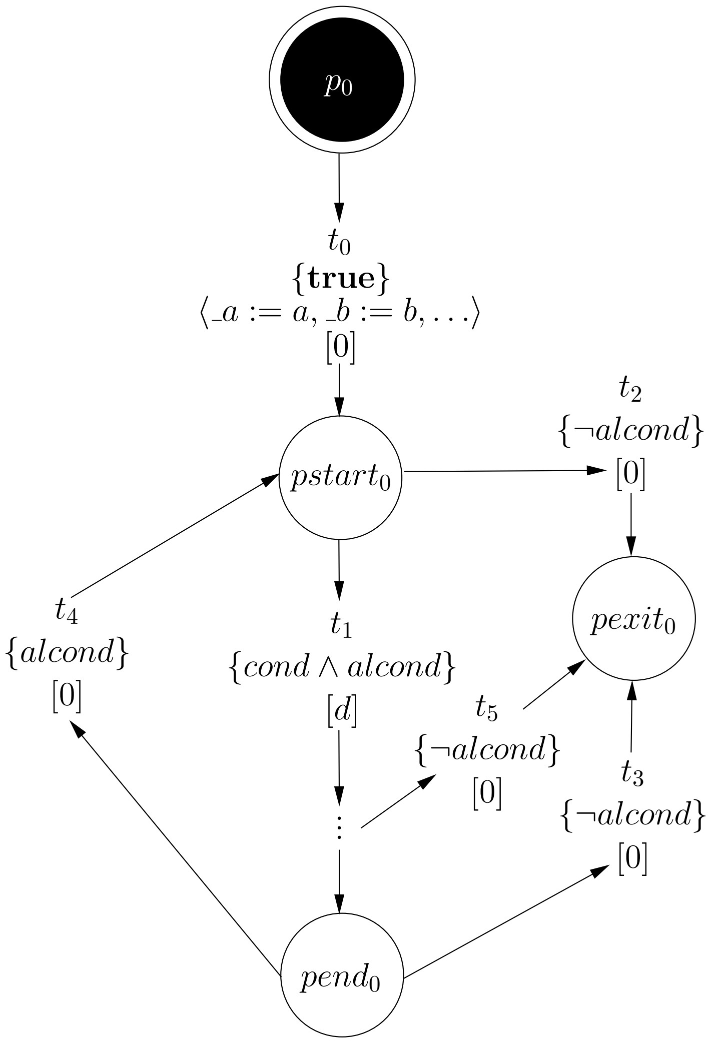


Figure 3.10: LPN for an **always** statement with condition list $\{a, b, \ldots\}$. The LPN associated with the statements in the **always** block go between the $pstart_0$ and $pend_0$ places. Each transition in this LPN and between them has the condition $alcond$ combined with the original condition. The expression $alcond$ is $(a = \_a) \wedge (b = \_b) \cdots$.

5. Mark the initial place of the first statement.

Using LAMP, the property for our PI circuit can be expressed as shown in Fig. 3.11. Note that the values have been scaled according to the factor provided by the model generator to give integer values. After this property is compiled by `LEMA`, the property LPN shown in Fig. 3.12 is obtained.

3.4 Results

Using simulation data and the model generator in `LEMA`, three different PI models were created using PI circuits with 4, 8, and 16 phase shifts, respectively. Fig. 3.2 shows the

```
property PhaseInterpolator {
  real ctl;
  real omega;
  real phi;
  always {
    wait(phi>=0);
    if (~(ctl>=170)) {
      assert(omega<220, 1699);
      wait(omega>=220, 102);
    }
    else if ((ctl>=170)&~(ctl>=250)) {
      assert(omega<220, 1479);
      wait(omega>=220, 102);
    }
    else if ((ctl>=250)&~(ctl>320)) {
      assert(omega<220, 999);
      wait(omega>=220, 102);
    }
    else if (ctl>=320) {
      assert(omega<220, 759);
      wait((omega>=220, 102);
    }
    wait(~(phi>=0));
  }
}
```

Figure 3.11: PI circuit property using LAMP.

model generated for the circuit with 4 possible phase shifts. For each of the circuits, a corresponding property was constructed in LAMP to check that the phase shifts generated by the circuit are correct. One example is shown in Fig. 3.11 for a PI with 4 different phase shifts. After combining the LPN generated for this property with the LPN for the model LPN and an LPN for the environment, LEMA's zone-based model checker was used to verify the PI circuit satisfies the property (i.e., no fail transitions fired). The verification results of PI circuits for 4, 8, and 16 phases are given as the first three entries of Table 3.1. As can be seen from this table, LEMA is able to successfully verify that the output phase shifts are correct.

Also considered were a couple of variations that caused verification errors. First, to simulate an output clock that goes high too soon, the property was changed for the PI with 4 phases so that the assert statement for the control value 40 asserts that *omega* is less than 220 for 680 time units instead of 759. As seen in the fourth entry of Table 3.1, the

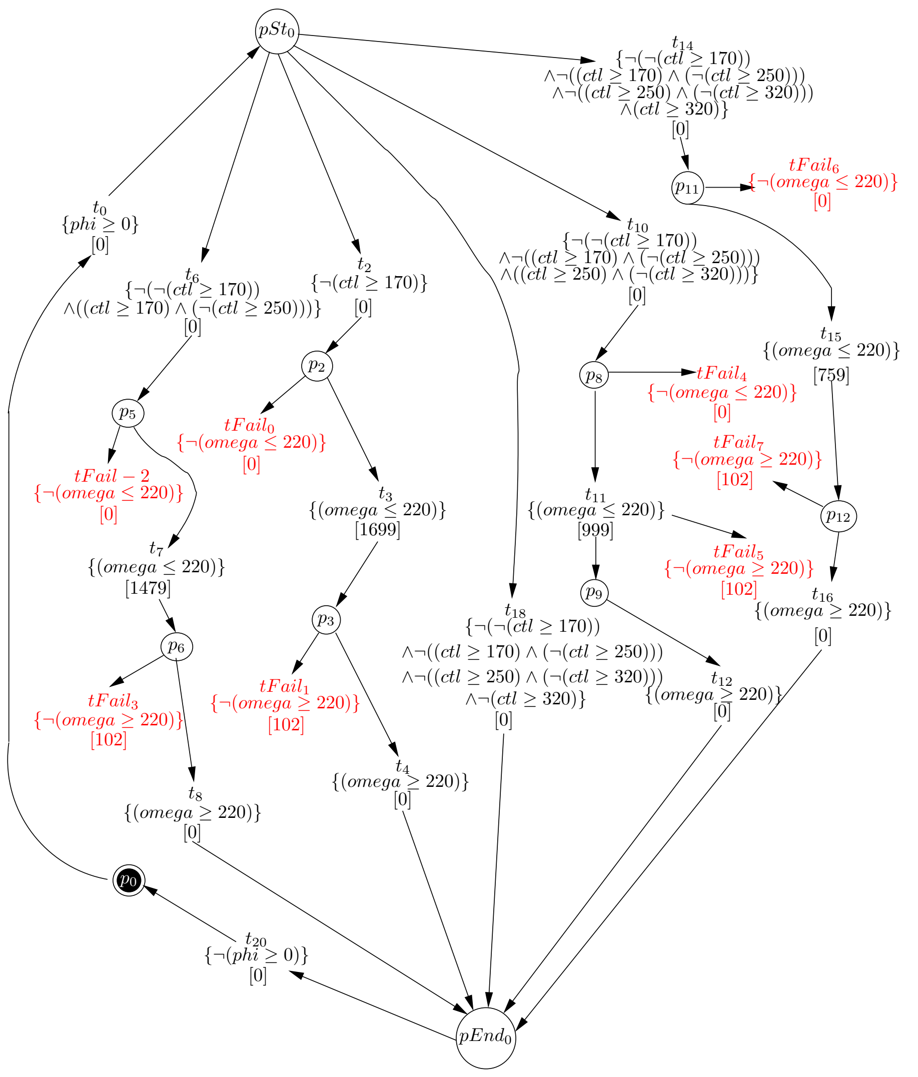

pSt_0
t_{14}
$\{\neg(\neg(ctl \geq 170))$
$\wedge\neg((ctl \geq 170) \wedge (\neg(ctl \geq 250)))$
$\wedge\neg((ctl \geq 250) \wedge (\neg(ctl \geq 320)))$
$\wedge(ctl \geq 320)\}$
[0]
p_{11}
$tFail_6$
$\{\neg(omega \leq 220)\}$
[0]
t_0
$\{phi \geq 0\}$
[0]
t_{10}
$\{\neg(\neg(ctl \geq 170))$
$\wedge\neg((ctl \geq 170) \wedge (\neg(ctl \geq 250)))$
$\wedge((ctl \geq 250) \wedge (\neg(ctl \geq 320)))\}$
[0]
t_6
$\{\neg(\neg(ctl \geq 170))$
$\wedge((ctl \geq 170) \wedge (\neg(ctl \geq 250)))\}$
[0]
t_2
$\{\neg(ctl \geq 170)\}$
[0]
t_{15}
$\{(omega \leq 220)\}$
[759]
p_2
p_8
$tFail_4$
$\{\neg(omega \leq 220)\}$
[0]
p_5
$tFail_0$
$\{\neg(omega \leq 220)\}$
[0]
$tFail_7$
$\{\neg(omega \geq 220)\}$
[102]
$tFail-2$
$\{\neg(omega \leq 220)\}$
[0]
t_3
$\{(omega \leq 220)\}$
[1699]
t_{11}
$\{(omega \leq 220)\}$
[999]
p_{12}
t_7
$\{(omega \leq 220)\}$
[1479]
$tFail_5$
$\{\neg(omega \geq 220)\}$
[102]
p_9
t_{16}
$\{(omega \geq 220)\}$
[0]
p_3
t_{18}
$\{\neg(\neg(ctl \geq 170))$
$\wedge\neg((ctl \geq 170) \wedge (\neg(ctl \geq 250)))$
$\wedge\neg((ctl \geq 250) \wedge (\neg(ctl \geq 320)))$
$\wedge\neg(ctl \geq 320)\}$
[0]
p_6
t_{12}
$\{(omega \geq 220)\}$
[0]
$tFail_3$
$\{\neg(omega \geq 220)\}$
[102]
$tFail_1$
$\{\neg(omega \geq 220)\}$
[102]
t_4
$\{(omega \geq 220)\}$
[0]
t_8
$\{(omega \geq 220)\}$
[0]
p_0
t_{20}
$\{\neg(phi \geq 0)\}$
[0]
$pEnd_0$

Figure 3.12: The LPN generated for the LAMP property in Fig. 3.11.

Table 3.1: Verification results for a PI circuit. These results are generated using `LEMA`, a java-based verification tool, on a 64-bit machine running an Intel Core i5 CPU M 480@ 2.67GHz with 4 processors and 4GB of memory.

Property	Time (s)	States	Verifies?
PI with 4 control signals	0.135	126	Yes
PI with 8 control signals	0.277	300	Yes
PI with 16 control signals	1.362	769	Yes
PI with short delay	0.083	14	No
PI with changing controls	0.779	2407	No

property correctly signals a failure. Each of these checks is done with an environment that can nondeterministically change the control signal shortly before the next time the input clock goes high. If this restriction is removed and the control signal is allowed to change at any time, `LEMA` finds a failure as indicated by the fifth result of Table 3.1. This failure occurs because after the property LPN begins checking the output clock phase for one control signal, the environment can change the control signal to a different value, resulting in a different phase. The property then continues to check for the behavior for the previous control value which indicates a failure. This failure can be fixed by adding a second **always** block around the whole property and adding the control signal *ctl* to the condition list. Dealing with such transient behavior is further illustrated in verifying the VCO model of Figs. 3.3 and 3.4.

To verify that the VCO has the correct delay after a suitable time in the unstable period, one could use the property in Fig. 3.13 and compile it into the LPN shown in Fig. 3.14. This property declares a control variable, *ctl*, and a clock signal, *out*. The first **always** block ensures that the following property is checked repeatedly. Next, the **delay** waits for the clock to stabilize and the **waitPosedge** ensures that the frequency check starts when the clock, *out*, next goes high. The second **always** block includes *ctl* in its condition list, thus the next statements are repeatedly checked unless *ctl* changes. On the event of *ctl* changing, the second **always** block exits and the outer **always** block starts the property check over. The statements inside the inner **always** block check that the output clock remains high (indicated by *out* being at least 40 units) for the appropriate delay f_3. Then, the property checks that the clock goes low (*out* less than 30) within 3 time units and remains low for f_4 time units before, finally, going high again within 5 time units.

The results of applying the property in Fig. 3.14 to the VCO model in Figs. 3.3 and 3.4 are listed in Table 3.2 with the label *Limited Phase Checker*. The first three lines show the results when the control is set to a single control value of 2, 3, or 4 and in each case the system is verified. Next, the environment is modified to nondeterministically change to one of the three values 2, 3, or 4 every 3000 time units and again the system is verified. Finally, the environment is modified to allow the voltage level to change to 2, 3, or 4 at any time. In this case, the property fails. The reason is due to the placement of the delay statement outside the second **always** block. If the control changes just prior to entering the second always block, then the model is in the unstable state while the property LPN checks for the stable frequency. Once the **delay** and **waitPosedge** statements are placed inside the second **always** block, the property is verified in all cases of the environment as shown in

```
property VCO {
  real ctl;
  real out;
  always {
    delay(1000);
    waitPosedge(out >= 40);
    always (ctl) {
      assert(out >= 40, f3);
      wait(out <= 30, 3);
      assert(out <= 30, f4);
      wait(out >= 40, 5);
    }
  }
}
```

Figure 3.13: VCO circuit property using LAMP.

the last five lines of Table 3.2.

3.5 TA Translations

LAMP can be used in the context of formalisms other than LPNs. This section illustrates the compilation of LAMP into a set of TAs inspired by the properties constructed in [16] to verify hybrid systems. Following in a similar vein, Figs. 3.15 and 3.16 show how the statements of LAMP can be compiled into TAs. In these automata, there is a single clock variable x. When a transition is taken (that is, one of the arrows is followed), the clock x is reset as is indicated by being listed in the curly braces. A transition is allowed to fire if the received signal satisfies the Boolean expression (the top expression listed) and the clock variable x satisfies the corresponding inequality. For example, in Fig. 3.15b, the $\neg b$ transition can be taken as long as b has not been received. Each time the transition is taken, the clock is reset. Once a b is received, then the lower transition can be taken, leading to the state q_1.

When compiling a LAMP statement into TAs, one uses the same process as in Section 3.3. So, each statement of LAMP is converted to its corresponding template, making the last state of the prior statement (q_1 in Fig. 3.15, q_2 in Fig. 3.16, and end_0 in Fig. 3.17) the same as the first state in the following statement (q_0 in Figs. 3.15 and 3.16 and $start_0$ in Fig. 3.17). The statements inside an **if-else** are compiled in a similar way and are stated in the corresponding blocks $R1$, $R2$, and $R3$. The first state of the first statement is considered the initial state.

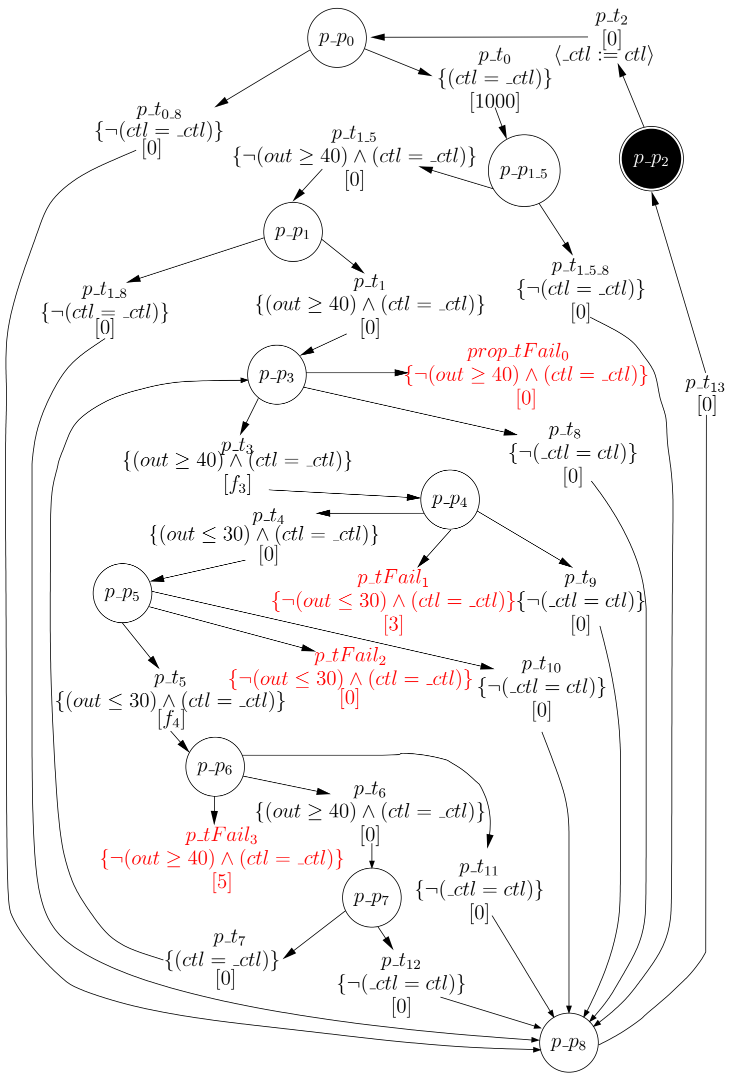


Figure 3.14: The LPN generated for the LAMP property expressed in Fig. 3.11.

The Figs. 3.15, 3.16, and 3.17 comprise all the statements of LAMP except for the **always** block with the conditions list. This full generality is not supported for TAs; however, an **always** block with no conditions list is supported in the same way as in [66], that is, the **always** block indicates the last state of the last statement is identified with the first state of the first statement. Again, the first state of the first statement in the **always** block is considered the first state.

As an example, this section considers a version of the robot property introduced in [16], as shown in Fig. 3.18. A robot can move with a rate of λ and is constrained to an NxN grid where it can move up, down, left, or right. Cells that are marked with C have a time out, T_1, so that the robot cannot stay in cells marked C for more than T_1 time units. Similarly, cells marked D have a time out of T_2 time units. The blacked-out cells (which are considered

Table 3.2: Verification results for a VCO circuit. These results are generated using `LEMA`, a java-based verification tool, on a 64-bit machine running an Intel Core i5 CPU M 480@ 2.67GHz with 4 processors and 4GB of memory.

Property	Control Signals	Time (s)	States	Verifies?
Limited Phase Checker	2	0.144	22	yes
Limited Phase Checker	3	0.177	22	yes
Limited Phase Checker	4	0.177	136	yes
Limited Phase Checker	2,3,4 reg. int.	0.223	185	yes
Limited Phase Checker	2,3,4 random	0.419	322	no
General Phase Checker	2	0.158	18	yes
General Phase Checker	3	0.161	18	yes
General Phase Checker	4	0.161	24	yes
General Phase Checker	2,3,4 reg. int.	0.195	24	yes
General Phase Checker	2,3,4 random	1.411	336	yes

to have a label of E) are walls that the robot should never enter. The robot starts in the A cell indicated by the 'Start' arrow and must reach the end cell labeled B by a time limit T_3.

Collectively, there are four assertions: the time out condition for C cells, the time out condition for D cells, the prohibition of entering E cells, and the time limit to reach the B cell. Multiple, simultaneous properties are not explored in the previous material in this chapter, but for LPNs there is nothing inherently difficult. To check more than one property, one compiles each property LPN and then verifies the model LPN together with all the property LPNs, simultaneously. In like fashion, each of these assertions is given a separate LAMP description. This gives rise to the four properties listed in Figs. 3.19a, 3.20a, 3.21a, and 3.22a.

In Fig. 3.19a, the property declares a Boolean variable C to indicate when the robot enters a cell labeled C. The property is to limit the time the robot stays in a cell labeled C. The **wait** statement waits for C to become **true**, indicating that the robot has entered a C state. The property then waits for C to go **false** in T_1 time units. Thus, a failure occurs if C does not go **false** within T_1 time units. If C does become **false** within T_1 time units, then the bounded **wait** passes and the **always** block brings the checker back to waiting for C to become **true**. The property for D is nearly the same and is shown in Fig. 3.20a.

Fig. 3.21a shows how to assert that the wall cells are never entered. In this property, the **assertUntil** is used to utilize its purely Boolean characteristics. By putting the first

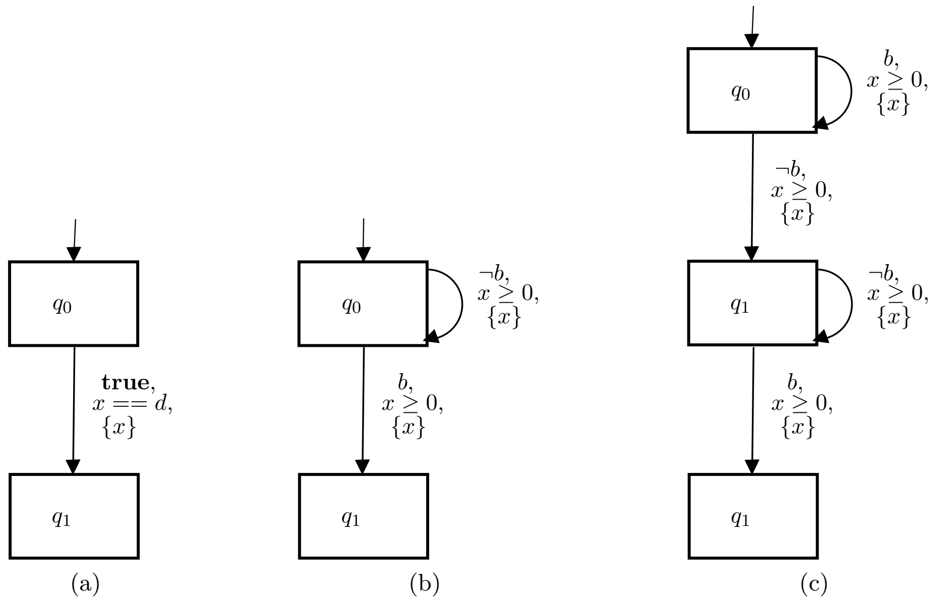
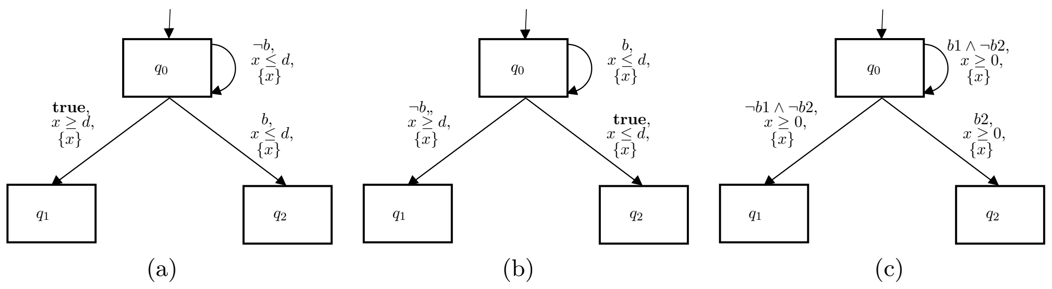


Figure 3.15: TA translations for single argument LAMP statements. TA for (a) **delay**(d), (b) **wait**(b), and (c) **waitPosedge**(b).



q_0
$\neg b$, $x \leq d$, $\{x\}$
true, $x \geq d$, $\{x\}$
b, $x \leq d$, $\{x\}$
q_1
q_2
(a)
q_0
b, $x \leq d$, $\{x\}$
$\neg b$,, $x \geq d$, $\{x\}$
true, $x \leq d$, $\{x\}$
q_1
q_2
(b)
q_0
$b1 \wedge \neg b2$, $x \geq 0$, $\{x\}$
$\neg b1 \wedge \neg b2$, $x \geq 0$, $\{x\}$
$b2$, $x \geq 0$, $\{x\}$
q_1
q_2
(c)

Figure 3.16: TA translations for binary argument LAMP statements. (a) **wait**(b, d), (b) **assert**(b, d), and (c) **assertUntil**($b1, b2$) statements. In each case, q_1 is the incorrect behavior and q_2 is the correct behavior.

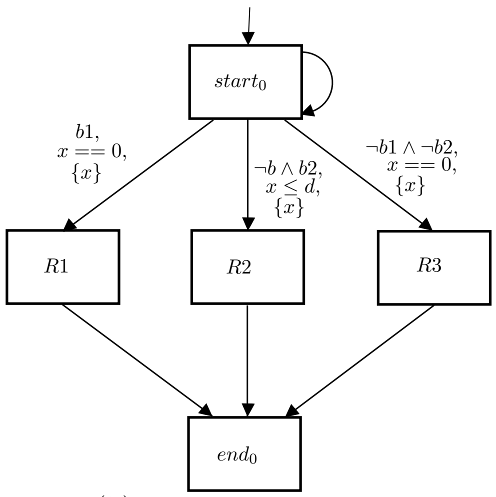
condition to $\sim E$ and the second condition to **false**, the condition asserts that $\sim E$ occurs until **false**. Since **false** never occurs, the **assertUntil** can only be satisfied as long as E remains **false**.

The final property is the assertion that the robot reaches the B cell in T_3 time units. The LAMP translation of this property is shown in Fig. 3.22. This property is just a bounded wait statement that waits for B to become **true** in T_3 time units.

The last thing that needs to be added to these automata are the accepting states that indicate successful runs. To match the safety property semantics of LPNs, every state is marked as accepting, except the failure states of **wait**(b, d), **assert**(b, d), and

```
if (b1) {
   R1
}
else if (b2) {
   R2
}
else {
   R3
}
```

(a)



(b)

Figure 3.17: TA translation for the ***if-else*** statement. (a) LAMP syntax for an ***if-else*** statement. (b) TA for an ***if-else*** statement.

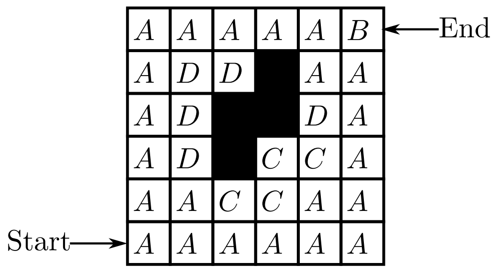


Figure 3.18: Robot grid. The robot starts in the A cell indicated by the 'Start' arrow and must reach the cell labeled B withing T_3 time units. Cells that are blacked out (E cells) cannot be entered. The robot can only remain in cells labeled C and D for T_1 and T_2 time units, respectively.

assertUntil$(b1, b2)$. As an example, Fig. 3.23 shows the translated TA properties for the robot.

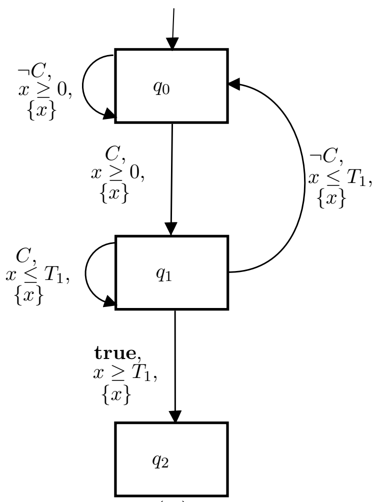
3.6 Conclusion

In order to verify whether an AMS circuit is correct given a model of the behavior, one needs to start with a property to verify. Several options have been proposed that are primarily inspired by LTL/CTL-like formalisms or by programming-like languages such as PSL and SystemVerilog. Although these methods are powerful and quite general, these languages often are difficult to convince designers to use since they have a steep learning curve.

Prior to this work, LAMP had statements to wait an unbounded or bounded amount

```
property Robot_C {
   boolean C;
   always{
      wait(C);
      wait(~ C, T1);
   }
}
```

(a)


q_0
$\neg C$, $x \geq 0$, $\{x\}$
C, $x \geq 0$, $\{x\}$
$\neg C$, $x \leq T_1$, $\{x\}$
q_1
C, $x \leq T_1$, $\{x\}$
true, $x \geq T_1$, $\{x\}$
q_2

(b)

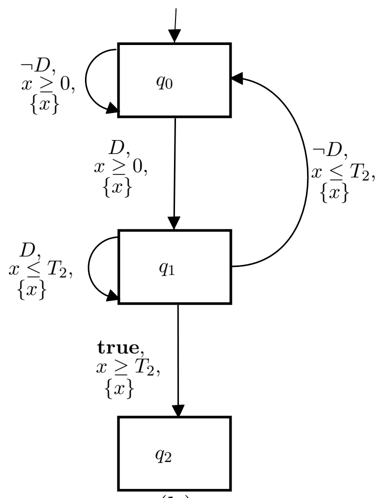
Figure 3.19: Property and TA translation for staying in state C for a limited time. a) A property asserting that the robot cannot be in a state C for more than T_1 time units. b) The TA translation of the property. Runs that enter the q_2 state exhibit the incorrect behavior.

of time for a Boolean variable to become **true**, to wait on the positive edge of a Boolean variable, to assert a Boolean variable remains **true** for a bounded amount of time, and to assert that a Boolean variable must be **true** until a second Boolean variable becomes **true** [66, 69]. LAMP also has the control constructs of **if-then** and **always**. However, the existing statements are not sufficient to verify all properties of interest. In particular, LAMP has difficulty when a transient period needs to be ignored without an assertion or when the environment changes while a property is being checked. Such an ability is helpful when one wants to ignore an unstable period and begin a check after a suitable settling time. This chapter extends the types of properties that LAMP can verify by adding a more flexible **always** statement that can detect when the environment changes and adding a **delay** statement that merely delays a check for a prescribed amount of time.

This chapter presents LAMP, a more intuitive language for AMS property specification, and demonstrates the utility of LAMP by showing how it can be used to express a desired property of a PI and VCO circuit. For the PI, the property is that a precise phase shift should be produced by this circuit under the control of its input signal. This property is shown to be simple to express in LAMP while it is more opaque in formalisms such as STL and RT-SVA. Furthermore, this chapter demonstrates the use of LAMP in a verification setting by verifying that the output phase for various control signals is correct for a PI circuit with 4, 8, or 16 different phase shifts. For the VCO example, the property is to

```
property Robot_D {
   boolean D;
   always{
      wait(D);
      wait(~ D, T2);
   }
}
```

(a)

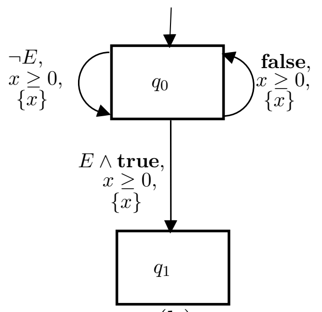

$\neg D$, $x \geq 0$, $\{x\}$
q_0
D, $x \geq 0$, $\{x\}$
$\neg D$, $x \leq T_2$, $\{x\}$
D, $x \leq T_2$, $\{x\}$
q_1
true, $x \geq T_2$, $\{x\}$
q_2

(b)

Figure 3.20: Property for staying in state D for a limited time. (a) The property statement. The property asserts that the robot cannot be in a state D for more than T_2 time units. (b) The automaton translation of the property. Runs that enter the q_2 state exhibit the incorrect behavior.

```
property Robot_E {
   boolean E;
   always{
      assertUntil(~ E, false);
   }
}
```

(a)


$\neg E$, $x \geq 0$, $\{x\}$
q_0
false, $x \geq 0$, $\{x\}$
$E \wedge$ **true**, $x \geq 0$, $\{x\}$
q_1

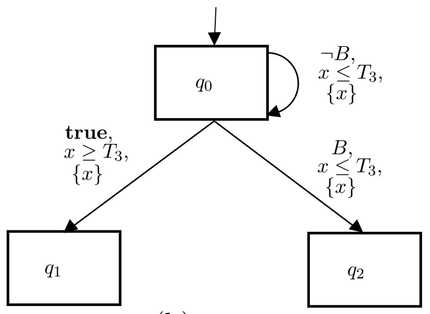
(b)

Figure 3.21: Property for prohibiting entering E. (a) The property statement. The property asserts that the robot cannot be in an E state. (b) The automaton translation of the property. Runs that enter the q_1 state exhibit the incorrect behavior.

verify the appropriate frequency is produced according to the what the voltage signal is after a suitable delay for the frequency to stabilize. This property is checked for three control voltages and it is shown that the property can be written to verify the circuit in an environment that randomly changes the control voltage. Finally, it is shown how LAMP can be translated into TAs and used for verification. As a case study, it is shown how a constraint on the motion of a robot can be written in LAMP and be translated into TAs.

property Robot_B {
  boolean B;
  wait(B, T_3);
}

(a)


q_0
$\neg B$, $x \leq T_3$, $\{x\}$
true, $x \geq T_3$, $\{x\}$
B, $x \leq T_3$, $\{x\}$
q_1
q_2

(b)

Figure 3.22: Property for asserting B must be reached. The property asserts that the robot must reach state B in time T_3. Runs that enter the q_1 state exhibit the incorrect behavior. Runs that enter the q_2 state exhibit the correct behavior.

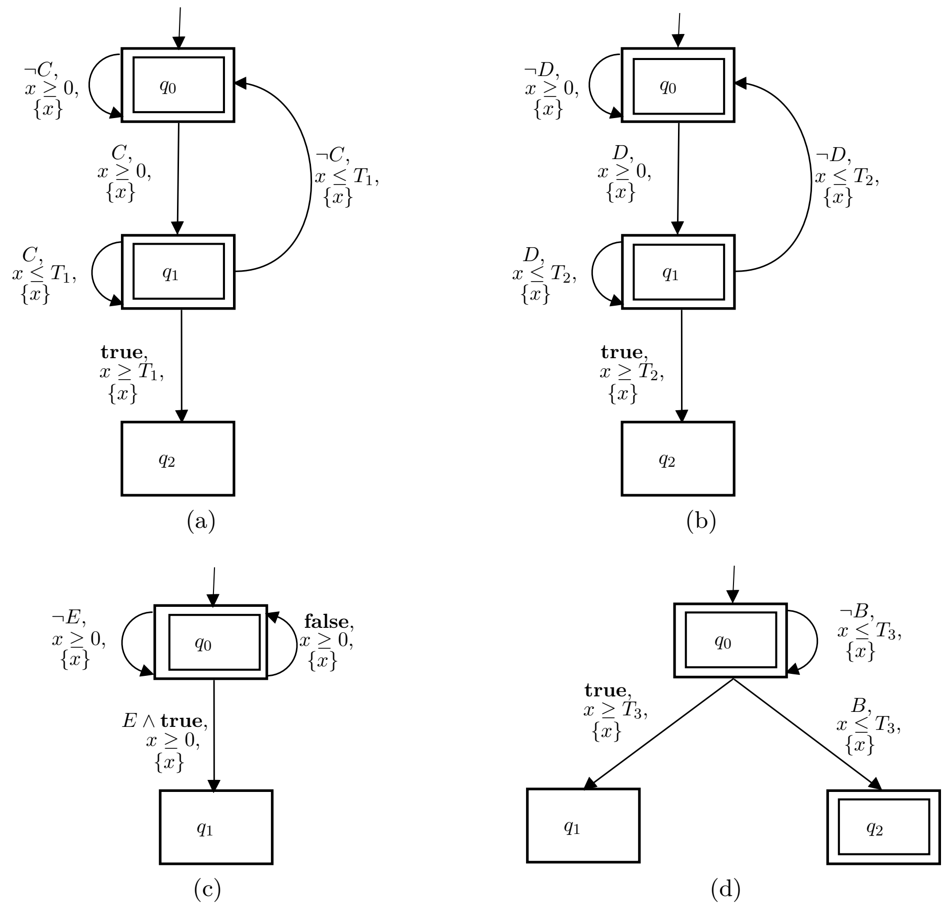

$\neg C$, $x \geq 0$, $\{x\}$
q_0
C, $x \geq 0$, $\{x\}$
$\neg C$, $x \leq T_1$, $\{x\}$
C, $x \leq T_1$, $\{x\}$
q_1
true, $x \geq T_1$, $\{x\}$
q_2
(a)
$\neg D$, $x \geq 0$, $\{x\}$
q_0
D, $x \geq 0$, $\{x\}$
$\neg D$, $x \leq T_2$, $\{x\}$
D, $x \leq T_2$, $\{x\}$
q_1
true, $x \geq T_2$, $\{x\}$
q_2
(b)
$\neg E$, $x \geq 0$, $\{x\}$
q_0
false, $x \geq 0$, $\{x\}$
$E \wedge$ **true**, $x \geq 0$, $\{x\}$
q_1
(c)
q_0
$\neg B$, $x \leq T_3$, $\{x\}$
true, $x \geq T_3$, $\{x\}$
B, $x \leq T_3$, $\{x\}$
q_1
q_2
(d)

Figure 3.23: TAs that accept runs for the correct behavior of the robot. Only the failure states are not marked as accepting. (a) Robot_C (Fig. 3.19). (b) Robot_D (Fig. 3.20). (c) Robot_E (Fig. 3.21). (d) Robot_B (Fig. 3.22).

CHAPTER 4

RANGES OF RATES

A common method for modeling hybrid systems is to use *hybrid automata* (HA) [8], which combine discrete transitions with dynamics described by first-order differential equations. The full generality of hybrid automata is difficult to formally verify, and it is common for authors to restrict their attention to more restrictive subclasses, such as LHAs [7]. Instead of allowing general first-order differential equations, LHAs restrict the invariants, guards, and flow relations to be linear equations over the continuous variables. Even though LHAs represent a restricted class of hybrid automata, they are still useful in describing systems and can approximate more general automata [98].

By restricting to LHAs, one can perform *reachability analysis* to verify that a system satisfies a given condition. Although the exact state space is undecidable [57], methods have been able to verify systems by approximating the reachable state space using classes of polyhedra [11, 39, 40, 102, 115]. The complexity of these methods comes from the choice of polyhedral class along with the methods used to update the space. For example, SpaceEx [41] utilizes template polyhedra and updates the state space by essentially lifting a numerical integrator to the level of sets. By increasing the number of template directions, the accuracy of the approximating state space improves, but at a cost of increasing the storage requirements and the number of operations needed to update the state.

One can avoid numeric integration techniques by restricting the modeling class even further. One option is to use LPNs [73] (Section 2.1). Although LPNs, in general, allow for a range of possible rates for each continuous variable, the authors in [72] assume a constant rate. This simplification allows them to avoid the expense of numerical integration by extending the methods used for *timed automata* (TA) [19, 84] to LPNs whose derivative is a single constant. Their method is based on *zones* that are described by *difference bound matrices* (DBM) [30]. One key advantage to this method is that time advancements can be performed by appropriately adjusting the largest possible value for each continuous variable and then retightening the boundary constraints defining the zone. To handle rates other than one, the zone is *warped* [72], a process where the variables in the original zone are scaled to produce rate 1 variables. After the scaling, the resulting $\mathbb{R}^n$ subset $\mathcal{Z}$ is, in general, no longer a zone. So the subset $\mathcal{Z}$ is replaced with the best over-approximating zone Z such that $\mathcal{Z} \subseteq Z$.

Although the methods used in [72] are straight-forward, they fall short of handling the ranges of rates possible in more general LPNs. To remedy this situation, a couple attempts ([25, 74]) have been made to extend zones to a range of rates. Both methods are based on a translational approach whereby the original model is transformed into a single rate model; however, as explained later, neither fully handles the use of ranges of rates in models.

This chapter shows how the method of zones can be extended to verify LPN models with ranges of rates. Similar to the translational approaches, this extension is based on the fact that states reachable using a rate chosen from a range of possible rates are also reachable using only the extremal rates together with rate zero. Moreover, since the work of [73] extends zones to capture all the states reachable from a set of states advancing with a particular rate, it is only necessary to consider the rate changes at fixed discrete moments in time and allow the zones to capture the simultaneous advancement of a collection of states.

This chapter is organized as follows: Section 4.1 introduces the main example used to illustrate the ideas in this chapter. Section 4.3 presents an algorithm for computing an over-approximation of the reachable state space. Section 4.4 presents a correctness argument for the algorithm. Section 4.5 discusses the related translational approaches followed by Section 4.6, which provides some experimental results. Finally, Section 4.7 gives conclusions.

4.1 Motivating Example

As a running example, consider a sequence of capacitors that are charged sequentially (see Fig. 4.1). The charging phase of the first capacitor is initiated by a switch sw_0. After 20μs of charging, a switch sw_1 is turned on, initiating the second capacitor's charging phase, and so on. When the switch sw_0 is turned off, the first capacitor starts discharging and the switch sw_1 is turned off, starting the second capacitor to discharge, and so on. Fig. 4.2 shows an LPN model of the i-th capacitor where the charging is some uncertain rate between 1mV/μs and 2mV/μs. The initial marking is $M_0 = \{p_{1,i}\}$ and is represented by the filled in circle. The values $V_i = 0$ and $V_i' = 0$ are the initial conditions for the voltage V_i. The variables sw_i and $sw_{(i+1)}$ are essentially Boolean variables with initial values of 0, representing **false**. The enabling conditions, delays, and variable assignments are in the curly braces, square brackets, and angle brackets, respectively. In this example, the delays

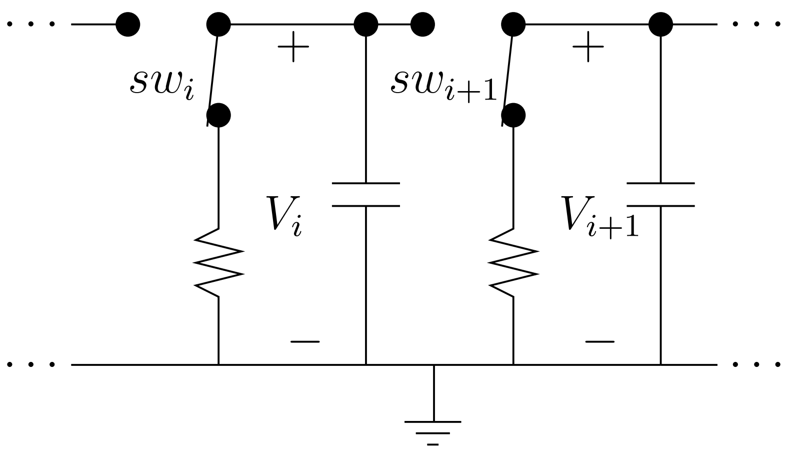


Figure 4.1: Capacitor stages i and $i + 1$.

are constants rather than bounds. Initially, the capacitor is not charging. When the signal sw_i is set to 1, charging is initiated by assigning V_i' the interval $[1, 2]$, which indicates the rate of V_i can be any rate between 1mV/μs and 2mV/μs. The capacitor is allowed to charge for 20μs (given as a delay on the transition $t_{2,i}$) before setting the variable $sw_{(i+1)}$ to 1. Once the charging is turned off, that is, when sw_i is set to 0, the capacitor begins to discharge at a rate of -1mV/μs. Finally, when the capacitor is fully discharged, the $t_{0,i}$ transition fires, setting the rate to zero, which indicates that the capacitor is fully discharged.

4.2 Theory

Zones are the class of polyhedra in Euclidean space $\mathbb{R}^n$ that are formed by intersecting half-planes of the formed by intersecting half-planes of the form $v_j - v_i \leq c$, where v_i and v_j are continuous variables and c is a constant. Given a finite set of continuous variables $v_0, v_1, \ldots, v_{n-1}$, the zone is completely determined by the pairwise inequalities $v_j - v_i \leq c_{i,j}$ together with $v_i - t_0 \leq c_{0,i}$ and $t_0 - v_i \leq c_{i,0}$ where t_0 is a timer that is always zero. By collecting the constants into a matrix, one forms the DBM for the zone. A standard zone is depicted in Fig. 4.3a. This example has two continuous variables v_0 and v_1 and the corresponding inequalities are:

$$\begin{array}{lll}
t_0 - t_0 \leq c_{0,0} = 0 & v_0 - t_0 \leq c_{0,1} = 5 & v_1 - t_0 \leq c_{0,2} = 3 \\
t_0 - v_0 \leq c_{1,0} = -1 & v_0 - v_0 \leq c_{1,1} = 0 & v_1 - v_0 \leq c_{1,2} = 1 \\
t_0 - v_1 \leq c_{2,0} = -1 & v_0 - v_1 \leq c_{2,1} = 3 & v_1 - v_1 \leq c_{2,2} = 0.
\end{array}$$

Collecting the constants into the DBM gives:

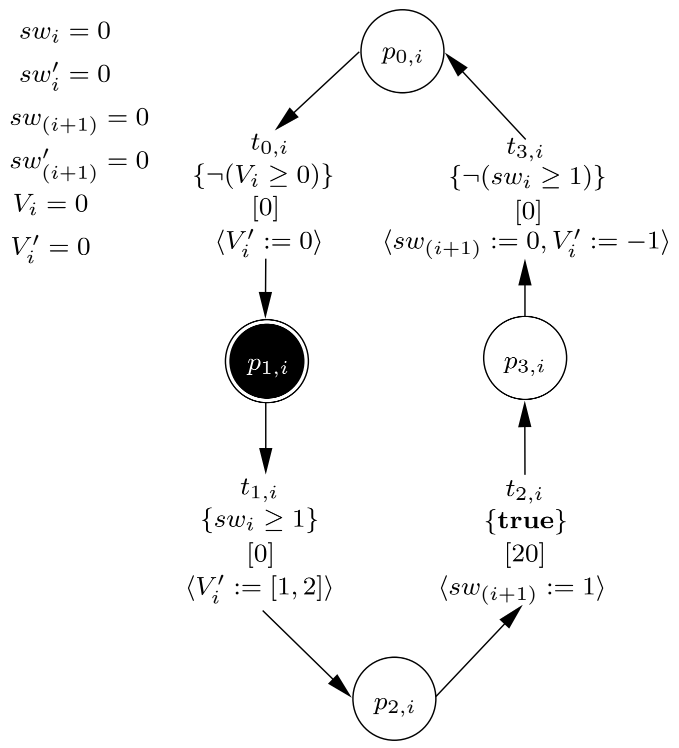


Figure 4.2: A model of a capacitor whose charging is turned on by sw_i. After a time delay of 20μs, the switch $sw_{(i+1)}$ is turned on (i.e., set to 1), initiating the charging of the next capacitor. Note that $[d]$ is used when $d_l(t) = d_u(t) = d$.

$$Z = \begin{array}{c} \\ t_0 \\ v_1 \\ v_2 \end{array} \begin{array}{c} \begin{array}{ccc} t_0 & v_1 & v_2 \end{array} \\ \begin{pmatrix} 0 & 5 & 3 \\ -1 & 0 & 1 \\ -1 & 3 & 0 \end{pmatrix} \end{array}$$

In the verification setting, the continuous variables are both the timers that determine how long a transition has been enabled, as well as the general continuous variables being used to model currents, voltages, and so on. When the rates of the continuous variables are 1, like in the case of timers, then it is simple to evolve the zone forward in time. One simply allows each point to move along a positive 45° angle as depicted in Fig. 4.3b. It is important to note that the set resulting from evolving time is still a zone. Thus, the method of allowing time to move forward is exact in the zone domain, that is, no over-approximations are introduced.

Algorithmically, time-advancements are implemented by setting the upper-bound on each variable to the largest possible value before an event is forced to happen for the variable. In particular, inequalities set the bound on the next inequality that changes as time evolves or infinity if no such inequality exists, and timers are set to the upper bound

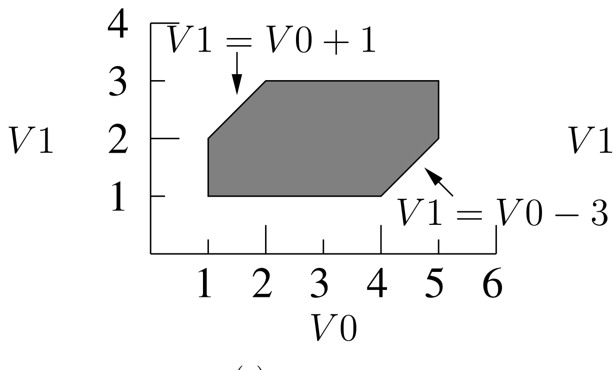


Figure 4.3: Time advancement of a zone. (a) An example of a zone. (b) The zone in evolved forward 1 time unit.

delay $d_u(t)$. The inequalities in the DBM are then tightened by running Floyd's all-pairs shortest path algorithm [84]. This retightening of the bounds handles the fact that time might not be able to evolve to the inequality or the upper bound due to a constraint on the other variables. In other words, the combination of setting timers and variables to their upper bounds and retightening automatically determines $\tau_{\max}$, as explained in Section 2.1.2.

Performing time advancements in this way is possible by assuming that all the variables have a rate of 1. If the rate of a continuous variable has a rate other than 1, then evolving time may result in the points not following the 45° path, resulting in a subset that is no longer a zone. Warping (see Fig. 4.4) was introduced in [72] to address this problem. With warping, if a variable v has a rate r that is not 1, then that variable is replaced with $\hat{v} = \frac{v}{r}$, creating a variable that does evolve with rate 1. Since the resulting variable again has rate 1, time-advancement works exactly the same as before. However, after replacing v with $\hat{v}$, the resulting subset $\hat{Z}$ may not be a zone that is describable by the inequalities $v_j - v_i \leq c$. To solve this issue, the set $\hat{Z}$ is replaced by an over-approximating zone $\mathcal{Z}$ such that $\hat{Z} \subset \mathcal{Z}$.

By combining warping and time-advancements, the following theorem is obtained, which is essentially shown in [74].

Theorem 1 (Section 4.4 of [74])**.** *Let Z be a zone, let $\mathcal{Z}$ be the zone obtained by warping Z, $\tau, r \in \mathbb{R}$ such that $\tau \geq 0$ and $r \neq 0$. Then, for all $z \in Z$, $z/r + \tau \in \mathcal{Z} \oplus [0, \tau]$ where $\oplus$ is the Minkowski sum $X \oplus Y = \{x + y \mid x \in X \text{ and } y \in Y\}$. Consequently, the point $z + rt$ is represented by a point in the time-advanced warped zone $\mathcal{Z} \oplus [0, \tau_{\max}]$.*

Theorem 1 implies that if Z is the original zone, then by changing the rates, warping, and advancing time, one obtains a zone that captures all the reachable states achievable

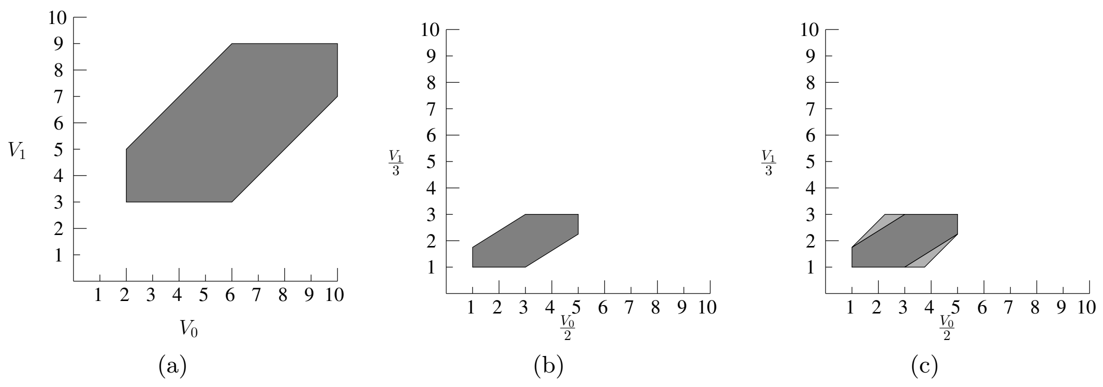


Figure 4.4: Warping a zone. a) Zone before warping. b) Resulting subset after scaling V_0 according to a rate of 2 and V_1 according to a rate of 3. c) Subset after scaling (darker subset) with over-approximating zone.

by starting in the original zone, making a rate change, and allowing time to move forward. Thus, warping provides a means of finding an over-approximation to the reachable state space of LPNs when the rates for each continuous variable evaluate to a constant.

Since a method already exists that handles single rates, ranges of rates can be handled if the range can be reduced to evaluating single rates. The following theorem provides a method for translating a trace of a continuous variable that has a range of rates $[a, b]$ into a trace that uses only the rates a and b.

Theorem 2 (See Theorem 3 for the proof)**.** *Let $a, b \in \mathbb{R}$ with $0 \leq a \leq b$ or $a \leq b \leq 0$, $\tau \in \mathbb{R}$ any nonnegative real number, and $q \in \mathbb{R}$ any real number. Then, for any real number v such that $a\tau + q \leq v \leq b\tau + q$, there exists a $\tau' \in [0, \tau]$ such that $f(\tau) = v$ where:*

$$f(x) = \begin{cases} b(x - \tau') + a\tau' + q & \text{if } \tau' \leq x \leq \tau \\ ax + q & \text{if } 0 \leq x \leq \tau' \end{cases}.$$

Section A.2 in the appendix provides a proof of Theorem 2. The main point of this theorem is that if v is a continuous variable with a range of rates of $[a, b]$ (such that $a < b < 0$ or $0 \leq a < b$), then any point that is reachable by v after τ time units is reachable by an approximating trace that uses only rates a and b.

At first inspection, this reduction may not appear to solve the problem of reducing a range of rates since their is an infinite number of points where the trace may switch that must be considered. It is true that one must consider an infinite number of points when considering single traces; however, the method of this chapter is to use a state set representation, namely zones, which collects an infinite number of traces into a finite representation. Using a zone, all states reachable from a state in zone Z in $\tau_{\max}$ time units using only the lower bound rate are collected together by performing warping and time advancement (Theorem 1). Let Z' be the resulting zone. After changing the rate of a continuous variable, say v, to the upper bound rate, warping and advancing time collects together all the states that are reachable from Z' in $\tau_{\max}$ time units using the new rate for v and the same rate for each of the remaining continuous variables. Let the resulting zone be Z''. By the preceding argument, Z'' contains as all traces (collection of states) that start in Z with all the rates at their lower bound, changes the rate of v to the upper bound at some time less than $\tau_{\max}$, and finally, advances time less than or equal to $\tau_{\max}$ time units. Theorem 2 ensures that every state reachable from z using any rate in the range is also reachable using only lower bound rate with a one time switch to the upper bound rate. Thus, Z'' contains all the reachable states allowing the variable v to assume any rate in its range of rates. For multiple rates, one considers the interleavings of the different possible rate changes and again the zone covers the reachable states.

4.3 Reachability Algorithm

The reachability algorithm presented here is an extension of the zone-based model checking algorithm used by `LEMA` as described in [72]. The main point of using zones (or any polyhedral method) is to reduce the infinite number of possible continuous variable states to a finite set of state sets that collect together several states into a finite representation. A state set is a tuple $\psi = \langle M, D, Q, RR, R, I, Z\rangle$ where:

- $M \subseteq P$ is the set of marked places;
- $D : T \to \mathbb{Q} \times \mathbb{Q}$ is the current range of delays for each transition;
- $Q : V \to \mathbb{Q} \times \mathbb{Q}$ is the range of values for each zero rate continuous variable;
- $RR : V \to \mathbb{Q} \times \mathbb{Q}$ is the current range of rates for each continuous variable;
- $R : V \to \mathbb{Q}$ is the current rate for each continuous variable;
- $I : \mathcal{I} \to \{\mathbf{false}, \mathbf{true}\}$ is the truth value for each inequality;
- $Z : (T \cup V \cup \{c_0\}) \times (T \cup V \cup \{c_0\}) \to \mathbb{Q} \cup \{\infty\}$ is a DBM composed of the transition clocks for the enabled transitions, the nonzero rate continuous variables, and c_0, a reference clock which is always zero.

This definition is modified from that in [72] to accommodate ranges of rates.

The basic reachability algorithm used by `LEMA` is shown in Algorithm 4.1. The algorithm starts by constructing the initial state set, ψ, for the LPN. In the initial state, $M = M_0$, $Q = Q_0$, $RR = R_0$, $R = \texttt{resetRates}(RR)$, and $I = \texttt{evalInequalities}(\psi)$. The DBM, Z, is composed of the initial values for all the continuous variables for which $R(v) \neq 0$. In addition, the DBM contains a clock initialized to 0 for every enabled transition. After adding the initial state to the set of reachable states, Ψ, the algorithm next calls the function `findPossibleEvents` which returns the set of all events, E, that are possible in the current state. The function `select` then chooses an arbitrary event, e, to be the next event to explore. If, after removing the event, e, the event set, E, still has events remaining, these remaining events are pushed onto a stack together with the current state. The next state, ψ', is computed by the `updateState` function and is the result of executing the event, e, in the current state, ψ. If the state ψ' has not been seen before, then the algorithm adds it to the set of reachable states, makes ψ' the current state to search from, and finds the possible events that can be executed from ψ' (now the current ψ). If ψ' has been seen before, then the algorithm checks if there are any event sets left on the stack to explore. If the stack is not empty, then the last record is removed and is used as the new current state, ψ, and current set of events, E. If the stack is empty, then there are no events left to explore, and the result is returned.

The functions `findPossibleEvents` and `updateState` must be modified to take into account a new rate change event. The `findPossibleEvents` algorithm is shown in Algorithm 4.2. Lines 1-7 are the same as in [72] and handle determining which transitions can fire and which inequalities can change. A transition can fire as soon as the clock (stored in the zone) exceeds the lower bound of the delay assignment for that transition. The function $ub(Z, t)$ is used to obtain the largest value of the clock, t, from the zone, Z. An inequality can change if time has advanced far enough for the variable to cross the constant associated with the inequality. Lines 8-10 are added to determine if any rate events are possible. Namely, any variables that are not evolving at their upper rate bound can have a rate event to set it to its upper rate. In all cases, the function `addSetItem` handles the adding and removing of elements from the event set according to which events should occur first.

The modified `updateState` function is shown in Algorithm 4.3. The main modifications to the original algorithm are the addition of `resetRates` and the `rateChange` event. The first step is to restrict the zone according to the knowledge provided by which event has occurred. When a transition fires, this means that the time has advanced at least to the

Algorithm 4.1: `reach`(N, T_{fail})

```
let $N = (P, T, V, F, En, DA, VA, RA, M_0, Q_0, R_0)$;
$\psi_0 = (M, D, Q, R, RR, I, Z) :=$ initialStateSet$(T, V, DA, En, M_0, Q_0, R_0)$;
$\psi := \psi_0$;
$\Psi := \{\psi\}$;
$\mathcal{E} :=$ findPossibleEvents$(T, En, D, R, RR, I, Z)$;
while $(true)$ do
    $E :=$ select$(\mathcal{E})$;
    if $(\mathcal{E} - \{E\} \neq \emptyset)$ then
        push$(\mathcal{E} - \{E\}, \psi)$;
    $\psi' :=$ updateState$(P, T, V, F, En, D, DA, VA, RA, RR, Z, e, \psi)$;
    if $(\psi' \notin \Psi)$ then
        $\Psi := \Psi \cup \{\psi'\}$;
        $\Gamma := \Gamma \cup \{(\psi, \psi')\}$;
        $\psi := \psi'$;
        if $(E \subseteq T_{fail})$ then
            return $Fail;$
    else
        $\Gamma := \Gamma \cup \{(\psi, \psi')\}$;
        if $stack\ not\ empty$ then
            $(\mathcal{E}, \psi) :=$ pop();
        else
            return
        $Success$;
```

lower bound delay and for inequalities this means that the continuous variable has reached the bounding constant. After the restriction, the bounds are retightened. Next, the state is updated according to whether the event is a set of inequalities changing, a transition firing, or a rate change event. In the cases of inequalities changing or transitions firing, the rates are reset via `resetRates`(RR), which resets the rates of each continuous variable according to its range of rates. A rate change event consists of a call to `rateChange` which takes the current state ψ and the rate change event, e, and makes the rate change of $R(v) = r_u(v)$. Note, this change has the effect of changing the rate for every state represented by the state set ψ. After assigning the new rate, the inequalities are updated according to any variable assignments and rate changes. The delays are then evaluated for any newly enabled transitions and the zone is updated according to which transitions are enabled. Finally, the zone is rewarped, time is allowed to advance up to $\tau_{\max}$, and the bounds are retightened.

Algorithm 4.2: `findPossibleEvents`(T, En, D, R, RR, I, Z)

```
1  $E := \emptyset$;
2  foreach $(t \in Z)$ do
3      if $(\texttt{ub}(Z, t) \geq d_l(t))$ then
4          $E :=$ addSetItem$(T, En, D, R, Z, E, t)$
5  foreach $(i \in \texttt{ineq}(En))$ do
6      if $(\texttt{ineqCanChange}(R, I, Z, i))$ then
7          $E :=$ addSetItem$(T, En, D, R, Z, E, i)$
8  foreach $(v \in V)$ do
9      if $(R(v) \neq r_u(v))$ then
10         $E :=$ addSetItem$(T, En, D, R, Z, E, v)$
11 return $E$;
```

4.4 Correctness

The proof that the above algorithm does over-approximate the reachable state space is done in 2 stages (see [38] for additional details). The first stage shows that every state set S' resulting from a transition firing or a set of inequalities changing is captured by some state set ψ'. The second stage handles the intervening rate changes and time advancements. First, suppose that tr is a transition and $S \xrightarrow{tr} S'$. Since $S \in \psi$, the same transition tr is enabled in ψ and is one of the possible event firings that are explored. Thus, one has $\psi \xrightarrow{tr} \psi'$. The state S' is then in ψ', since the same operations of updating the state S to produce S' are performed for all the states in ψ to produce ψ'. For example, the markings, M, are updated in the same fashion, the zone, Z, is updated to reflect the same continuous variable assignments, etc. If S' is the result of a set of inequalities, $\mathcal{I}$, changing, then this same set of inequalities is enabled to change in the state set ψ. Furthermore, the same set of inequalities can change to produce ψ'. Since the only states that are removed from ψ to produce ψ' satisfy the condition $v \neq c$ for each $v \geq c \in \mathcal{I}$, the state S' is not removed since v must equal c for each $v \geq c \in \mathcal{I}$ owing to the fact that the inequality is changing its truth value.

Next, the rate change events and time advancements are handled. Let ψ be the result of a transition firing, a set of inequalities changing, or the initial state set, and let S be a state in ψ. It is shown that if S' is a state resulting from a sequence of rate changes and time advancements up to a total time advancement of $\tau_{\max}$, then S' is in some ψ' resulting from ψ by a sequence of rate changes. For simplicity, assume that $\hat{v}$ is the only continuous variable and that $RR(\hat{v}) = [a, b]$. The argument is to first show that the state S' can be obtained by using a single rate change and then show that the resulting trace is captured by a sequence

Algorithm 4.3: `updateState`$(P, T, V, F, En, D, DA, VA, RA, RR, Z, e, \psi)$

```
1  $Z :=$ restrict$(Z, e)$;
2  $Z :=$ recanonicalize$(Z)$;
3  $R_{old} := R$;
4  if $(e \subseteq \mathcal{I})$ then
5      $\psi :=$ updateInequalities$(\psi, e)$;
6      $R :=$ resetRates$(RR)$;
7  else if $(e \subseteq T)$ then
8      $\psi :=$ fireTransition$(M, F, Q, V, VA, RA, Z, \psi, t)$;
9      $R :=$ resetRates$(RR)$;
10 else
11     $R :=$ rateChange$(\psi, e)$;
12 $\psi :=$ evalInequalities$(\psi)$;
13 forall the $(t \notin Z \wedge t \in \mathcal{E}(\psi))$ do
14     $D(t) =$ EvalAssign$(DA(t), Q, I, Z)$;
15 forall the $(t \in T)$ do
16     if $(t \notin Z \wedge t \in \mathcal{E}(\psi))$ then
17         $Z :=$ addT$(Z, t)$;
18     else if $(t \in Z \wedge t \notin \mathcal{E}(\psi))$ then
19         $Z :=$ rmT$(Z, t)$;
20 $(R, Z) :=$ dbmWarp$(R, R', Z)$;
21 $Z :=$ dbmWarp$(R_{old}, R, Z)$;
22 $Z :=$ advanceTime$(R, I, Z)$;
23 $Z :=$ recanonicalize$(Z)$;
24 return $\psi$;
```

of state sets. Theorem 2 establishes the first part. Using this theorem, there exists τ_1, τ_2 such that $S \xrightarrow{\tau_1} S'' \xrightarrow{R(\hat{v})\leftarrow b} S''' \xrightarrow{\tau_2} S'$. In like fashion, let ψ' be the state set resulting from ψ by changing the rate of $\hat{v}$ to b, and advancing time $\tau_{\max}$, that is $\psi \xrightarrow{R(\hat{v})\leftarrow b} \psi'' \xrightarrow{\tau_{\max}} \psi'$. All that remains to show is that the states S, S', and S'' are captured by the two state sets ψ and ψ'. Using Theorem 1, ψ contains all points $z \in \mathbb{Z}$ that are the result of a time advancement τ such that $\tau \leq \tau_{\max}$ when the rate of $\hat{v}$ is a. Thus, S'' is in ψ. Similarly, the construction of ψ' changes the rate of $\hat{v}$ to b for each state in ψ and captures all time advancements up to $\tau_{\max}$. So, S''' and S' are in ψ'.

Finally, extending to multiple continuous variables is a matter of finding the sequence of switching points for each of the continuous variables and applying the appropriate warping for each dimension.

4.5 Related Work

This chapter presents an extension of zones via a functional approach where the algorithm accounts for the changes needed to handle the ranges of rates. In contrast, the methods of [25, 74] use a translational approach where the original LPN or automaton is transformed by replacing the range of rates with single rate changes. Suppose a variable v has a range of possible rates $[a, b]$ in a given state. The method of [25] replaces the range of rates with 3 stages. The first stage determines the total amount of time the system spends in the state, say τ time units. The second stage determines the value of the continuous variable v after τ time units, provided the rate is a. The third stage determines the possible values for the continuous variable after τ time units for each of the possible rates in the interval $[a, b]$. Similar to the approach of [25], the method used in [74] replaces the state with 2 stages. The first stage sets the rate of v to a and then allows a transition to fire that sets the rate to b.

Both these methods achieve the goal of breaking the range of rates into traces that utilize only single rates, namely, the rates a and b. However, in each case, the traces explored only allow for a single rate change. Such a transformation is enough when the LPN or automaton is used to check a property, but it is not necessarily enough when ranges of rates are used for an LPN or automaton model. The single switching ensures that given a time τ and a range of rates $[a, b]$, every possible value of v at time τ is achievable by setting v to have rate a for some time $\hat{\tau}$, switching the rate to b and then allowing time to advance $\tau - \hat{\tau}$ (Theorem 2). This process breaks down when two sample times are involved. For example, suppose v is required to be at $2b$ at time 2 and at $2b + a$ after 1 more time unit, for $0 < a < b$. Then, it is no longer possible to start with the rate at a and then switch once to b since after 2 time units the rate needs to be changed back to a. A concrete example is given by the property LPN shown in Fig. 4.5. After being initiated by sw_i becoming **true**, the property checks that V_i is above 15 mV after 10 μs and then checks that V_i is more than 30 mV after an additional 10 μs. For V_i to be greater than 15 mV at 10 μs, the rate of V_i must switch at or before 5 μs. However, since the rate has switched, the rate must remain at 2 mV/μs for the next 10 μs resulting in V_i being at least 35 mV. Thus, it is not possible for the failure transition to fire. However, if V_i is 15 mV at 10 μs and the rate is set to 1 mV/μs, then V_i is 25 mV after an additional 10 μs, enabling the failure transition.

Instead of a translational approach, the method of Section 4.3 uses an algorithmic approach that allows the rate to switch once per transition firing or inequality changing. For LPNs, the number of times that a variable needs to be allowed to switch is the number

```
property Example {
  real Vi;
  always {
    delay (10);
    if (Vi >= 15){
      delay (10);
      assert (Vi >= 30, 0);
    }
    else {
      delay (10);
    }
    wait (~(Vi >= 0));
  }
}
```

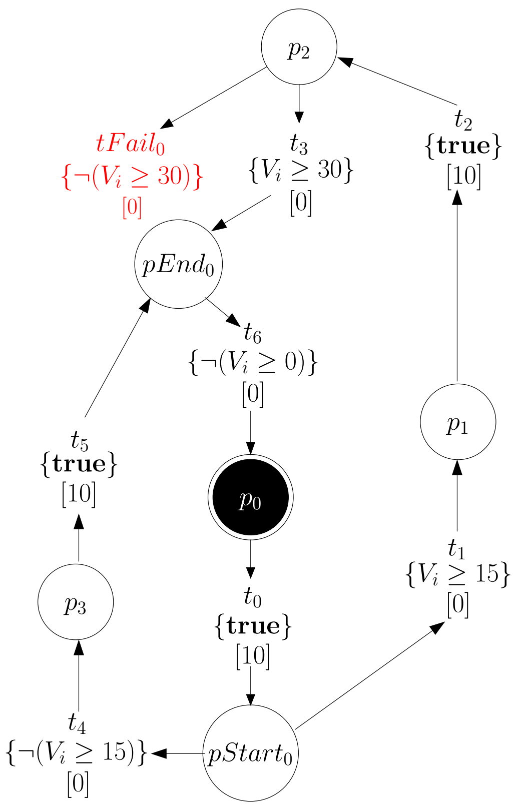


Figure 4.5: A property LPN for a capacitor stage in Fig. 4.2. When sw_i is 1, the property checks that V_i is above 15mV after 10μs, and then that V_i is more than 30mV after an additional 10μs. The property is violated if the fail transition, $tFail$, fires. On the left is a LAMP description of the property and on the right is an LPN translation.

of times that the LPN ‘samples’ the variable, that is, when an inequality changes or a transition fires. It is with these events that something is learned about the values of the continuous variables.

4.6 Experimental Results

This section compares verification results from the translational approach of [74] with results from the algorithmic approach of this chapter by using models having a varying number of capacitor stages (Fig. 4.2). In the translational approach of [74], the capacitor stages in Fig. 4.2 are modified to only use a single rate by setting the rate initially to 1 and then adding a one time transition which optionally sets the rate to 2. The transformed model is shown in Fig. 4.6. The algorithmic approach requires no modifications. The capacitor models are verified against the property in Fig. 4.5 with three different enabling conditions for $tFail$ and t_{11} using `LEMA`, a java-based verification tool. All experiments are

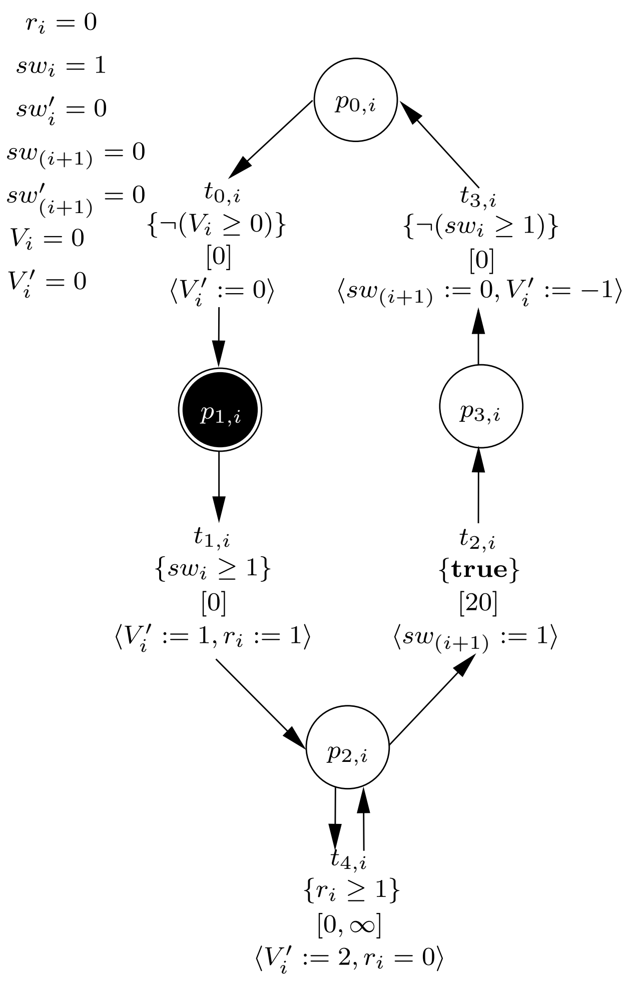


Figure 4.6: An example of translating an LPN model into single rates. The LPN model of Fig. 4.2 transformed according to the process in [74] to have only single rate assignments. In particular, the rate assignment of transition $t_{1,i}$ is changed to $V'_i = 1$ instead of the range $V'_i = [1, 2]$, and the transition $t_{4,i}$ is added to assign the rate to $V'_i = 2$. The delay on $t_{4,i}$ is $[0, \infty]$ to indicate that the rate change can occur at any time. After $t_{4,i}$ fires once, any subsequent firings have no effect, since the rate is already 2. The variable r_i is added to prevent these additional unneeded firings.

run on a 64-bit machine with an 3.4 GHz Intel Core i7-3570 CPU with 4 cores and 12GB of memory with a time limit of 6 hours. In each case, the property is placed on the last stage.

For the first example, the enabling condition on $tFail$ is $\neg(V_i \geq 18)$ (see Fig. 4.7a). In the capacitor models, the smallest possible voltage is 20mv, thus the failure transition should not fire. The verification results for the modified property are shown in Table 4.1. Both the translational approach and this chapter's algorithmic approach give the correct verification result. Namely, that the model satisfies the property. The state spaces are comparable, with the algorithmic approach producing no more than a multiple of 2 more

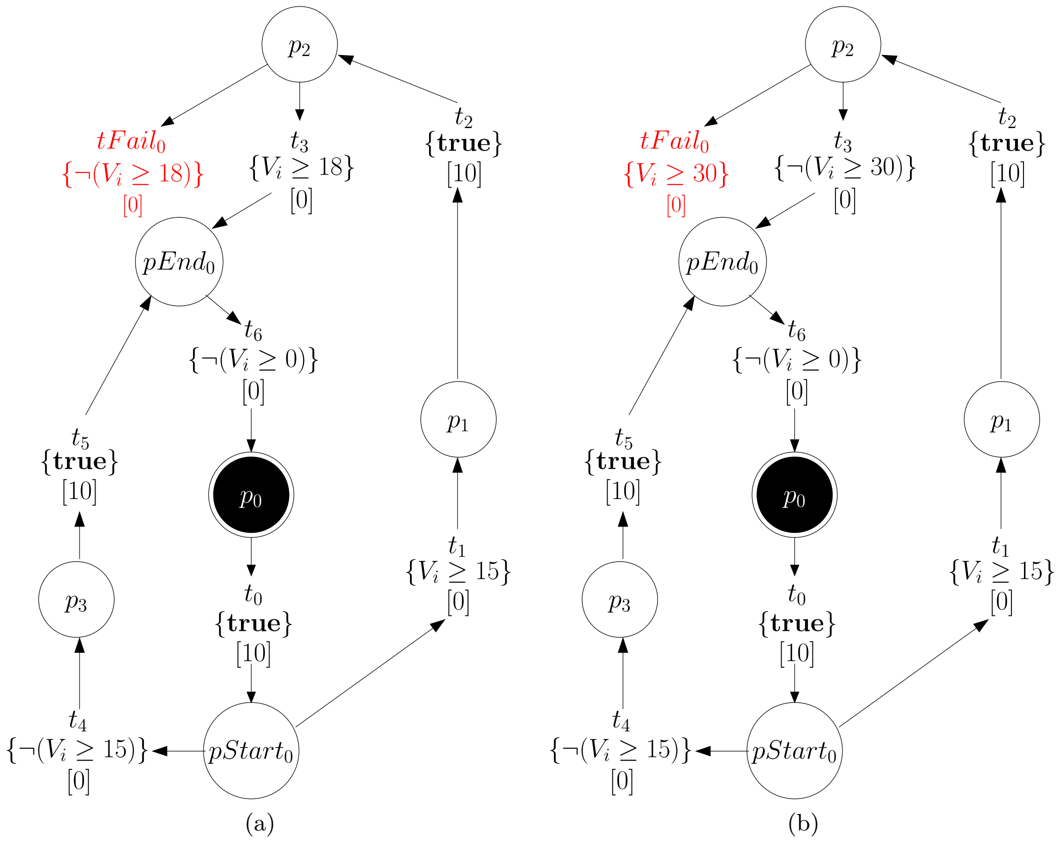


Figure 4.7: Modified capacitor properties. (a) Capacitor property (Fig. 4.5) changed to have the failure condition on $tFail$ to be $\neg(V_i \geq 18)$. (b) Capacitor property (Fig. 4.5) changed to have the failure condition on $tFail$ to be $(V_i \geq 30)$.

than the translational approach; however, the run time quickly explodes. This fact suggests that many new states are either subsets or supersets of previously found zones. To address this problem, `addSetItem` can be modified to ensure that rate events fire before all other events. The results are in the Algorithmic (opt) column in Table 4.1.

Another metric for comparing the three approach is to compare the number of events that are fired. Table 4.2 shows the total number of times an event fires during the state exploration for the translation method, the algorithmic method, and the algorithmic method with the rate optimization. Similar to the time statistics, the number of events for the algorithmic approach increases more rapidly than for the algorithmic approach with the rate optimization, and the number of events for the rate optimized algorithmic approach increase more rapidly than the number of events for the translational approach.

As a second example, the enabling condition for the failure transition $tFail$ is changed to $V_i \geq 30$ (see Fig. 4.7b). In this case, the model does not satisfy the property and both

Table 4.1: Comparison of translational approach [74] to our algorithmic approach with a tFail enabling condition of $\neg(V_i \geq 18)$. All cases verify as correct.

	Translational		Algorithmic		Algorithmic (opt)	
# Caps	Time (s)	States	Time (s)	States	Time (s)	States
1	0.149	72	0.188	59	0.108	35
2	0.268	235	2.01	144	0.457	56
3	0.487	553	40.085	279	0.941	65
4	1.083	881	15311.948	1148	2.954	105
5	3.066	3009	TIMEOUT	-	4.081	207

Table 4.2: The total number of event firings for the property with tFail $V_i \geq 18$.

	Event Count ($V_i \geq 18$)		
# Caps	Translational	Algorithmic	Algorithmic (opt)
1	139	313	106
2	407	29378	6484
3	950	805024	44372
4	3235	174330848	15395081
5	17179	-	414627973

approaches correctly find this result, as is shown in Table 4.3. In this case, the translational approach has state counts that are four to seven times larger than the algorithmic approach for 100, 200, 300, 400, and 500 stages of capacitors. Furthermore, the translational approach is now the one experiencing the rapid increase in time. The state count for the algorithmic approach is relatively small, which indicates that the failure occurs rather early in the state search. This analysis is backup by considering the number of events firing. Table 4.4 shows the number of events fired for both the translational and algorithmic approaches. The number of event firings is less for the algorithmic approach, which matches the fact that the state count is less.

The final property is the one shown in Fig. 4.5. This property first checks that if the voltage V_i is at least 15mV at 10μs, then the voltage must be at least 30mV after an additional 10μs. If this is not true, then *tFail* fires, indicating a failure. The results of verifying the last capacitor stage for models with one capacitor through eight are shown in Table 4.5. For each example, the translational approach indicates the model passes verification; however, this result is incorrect. If V_i has a rate of 1mV/μs for 5μs and then

Table 4.3: Comparison of translational approach [74] to our algorithmic approach with a tFail enabling condition of $V_i \geq 30$ that should not verify to be correct.

	Translational			Algorithmic		
# Caps	Time (s)	States	Verifies?	Time (s)	States	Verifies?
100	108.686	1639	no	6.504	233	no
200	972.568	3239	no	88.599	723	no
300	3496.862	4839	no	287.089	875	no
400	10290.709	6439	no	710.162	1127	no
500	TIMEOUT	-	no	3418.39	1967	no

Table 4.4: The total number of event firings for the property with tFail $V_i \geq 30$.

	Event Count ($V_i \geq 30$)	
# Caps	Translational	Algorithmic
100	2054	232
200	4054	722
300	6054	874
400	8054	1126
500	-	1924

has a rate of 2mV/μs for 5μs, the value of V_i at 10μs is 15mV. If the rate goes back to 1mV/μs for another 10μs, then the value of V_i is 25mV. This trace results in the sequence of transitions t_5, t_9, t_{10}, and $tFail$ in Fig. 4.5. Although zones over-approximate the state space, this trace is missing from the transformed model. Thus, the translational approach does not find this failure trace while the algorithmic approach does. Since the total state count is less for the algorithmic approach than for the translational approach, it is not surprising that the event count is also less as is shown in Table 4.6.

4.7 Conclusion

This chapter shows how a zone-based reachability method can be extended to verify models that utilize a range of rates. Previous methods have opted for a translational approach that converts models to ones with only a single rate change. Although this approach is adequate for properties, it is not enough when used for models. By using a method that allows for multiple resets, one can recover all the necessary behaviors.

Table 4.5: Comparison of translational approach in [74] to our algorithmic approach for the property shown in Fig. 4.5 that should not verify to be correct.

	Translational			Algorithmic		
# Caps	Time (s)	States	Verifies?	Time (s)	States	Verifies?
1	0.162	81	yes	0.146	52	no
2	0.287	240	yes	0.534	143	no
3	0.529	622	yes	2.00	280	no
4	1.31	1550	yes	13.6	481	no
5	3.83	3710	yes	130	877	no
6	13.1	8926	yes	1047	1649	no
7	76.2	52574	yes	860	3798	no
8	410	122014	yes	29709	7489	no

Table 4.6: The total number of event firings for the property with tFail $\neg(V_i \geq 30)$.

	Event Count $\neg(V_i \geq 30)$	
# Caps	Translational	Algorithmic
1	166	169
2	457	1724
3	1134	25597
4	4295	288859
5	22007	2159222

CHAPTER 5

OCTAGONS

Zones have been a successful tool for the formal verification of timed models like TA and TPNs. Zones form the basis of one of the three model checkers provided by `LEMA` [72], as well as the backbone for UPPAAL [19]. While the work of [73, 74] extends zones to models whose variables have rates other than 1, the necessary over-approximations are more extensive when the rates are both positive and negative. When the rates are positive, the best over-approximation can utilize positive 45^{circ} boundary lines to retain some of the relationships between pairs of variables. When one rate is positive and an another is negative, the best over-approximation that can be made is to accept the full rectangle defined by the variables' maximum and minimum values. In other words, one loses any restriction on the pairs of values.

A natural extension to zones that improves the approximation when the rates are different is to allow lines forming negative $45°$ degree angles. Such figures are, predictably, called *octagons*. Octagons can also be represented using a DBM [81], and the algorithm for finding the tightest constraints has complexity $O(n^3)$ [12, 81], which is the same as for zones. These facts make octagons an attractive choice for a simple, more accurate extension to zones. Octagons have been studied in the context of software checking, and so, some of the necessary algorithms are already available, such as restricting an octagon according to an inequality, projecting an octagon onto a single dimension to provide the variable's range, and the algorithm for ensuring the tightest constraints. So, the main algorithms needed for dynamic hybrid system models are warping, determining how time should advance, and adding new continuous variables and timers.

The chapter is structured as follows: Section 5.1 motivates the need for octagons by demonstrating how zones can lead to false negatives that octagons can avoid. Section 5.2 provides the necessary theory including the octagon DBM representation and explanations on how to add variables to the octagon, perform time advancements, and warp the octagon. Section 5.3 describes the necessary changes to the zone-based algorithms to adapt them to octagons. Section 5.4 demonstrates the use of the new algorithms on the motivating example and Section 5.5 provides conclusions.

5.1 Motivating Example

Although zones are able to capture the exact state space when every variable's rate is 1 [84], warping leads to an over-approximation of the state space when any the rate is different from 1. For example, consider the zone in Fig. 5.1a. If the variable y is assigned a rate of 3 and x is assigned a rate of 2, then the zone is first scaled by replacing y with $\frac{y}{3}$ and x with $\frac{x}{2}$. With this change, the subset is no longer bounded tightly by 45° lines on the upper left and lower right. Instead, the slopes of these lines are now both $\frac{2}{3}$ (Fig. 5.1b). Since the slope is no longer 1, the resulting subset is not a zone. To compensate, the subset is bounded by the best approximating slope 1 lines, thus producing the zone in Fig. 5.1c.

While the resulting zone is an over-approximation, the approximation is able to utilize 45° lines to avoid using the full bounding rectangle formed by the extreme values. When the rate is negative, the rectangle is the best possible approximation. Consider again the zone in Fig. 5.1a. When the rate of y is changed to -1, the zone in Fig. 5.1a is reflected across the x-axis and becomes the subset in Fig. 5.2a. The reflection changes the positive 45° lines into $-45°$ lines, which are not representable by inequalities of the form $y - x \leq c$. Thus, these constraints must be removed, creating the rectangle in Fig. 5.2b. With octagons, the negative 45° lines are allowed, so these constraints remain. In fact, Fig. 5.2 is an octagon, thus no approximations are necessary.

Since over-approximations add states to the reachable state space, it is possible that the resulting state space violates properties that the exact state space does not. Such cases

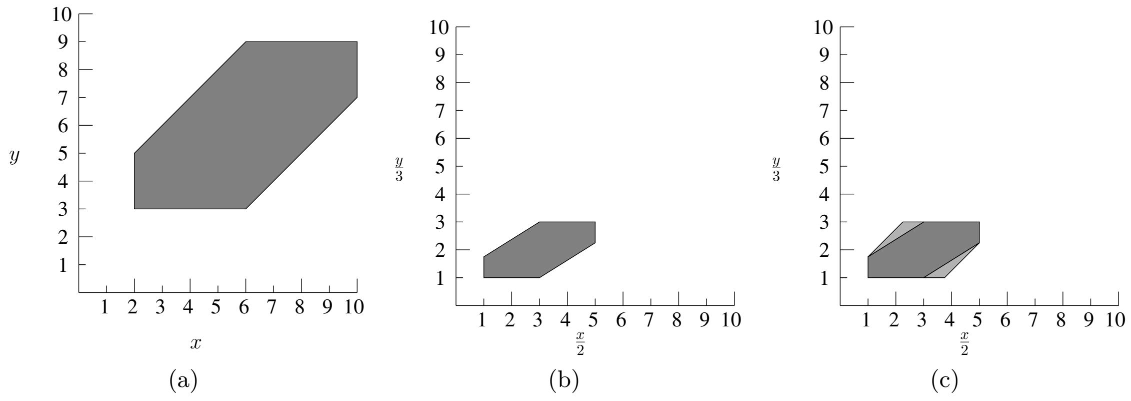


Figure 5.1: Warping a zone. a) Zone before warping. b) Resulting subset after scaling x according to a rate of 2 and y according to a rate of 3. c) Subset after scaling (darker subset) with over-approximating zone.

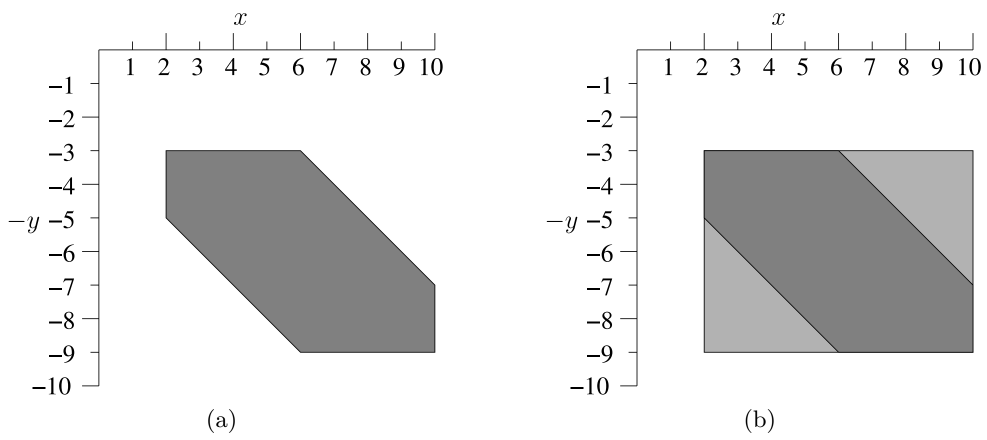


Figure 5.2: Negative warping. a) Subset resulting from changing the rate of y to -1 in 5.1a. b) Best over-approximating zone for negative warping.

are known as *false negatives.* As a concrete example, consider the LPN in Fig. 5.3. In this model, y starts with a range of values $0 \leq y \leq 1$ and a rate of 1. Since t_0 has a delay of $[0, 2]$, y can increase up to 2 units before t_0 fires. Hence, the largest value y can get before t_0 fires is 3. After t_0 fires, y is assigned a negative rate. So, y is largest if t_1 fires immediately. After t_1 fires, y is again assigned a rate of 1. This time y can increase for 1 time unit, resulting in y being 4. Now, either t_3 or t_4 fires once x reaches 5. To maximize y, x needs to be at a minimum in this zone, so y can increase the most. Since x always has a rate of 1, x has its minimum value if x starts at 0, the minimum initial value. Starting with x as 0 and firing the same sequence of transitions (t_0, t_1, and t_2) with the same delays (2, 0, and 1) gives $x = 3$. So, y can increase a maximum of 2 units before t_3 or t_4 must fire. Adding 2 to the maximum value of 4 reached so far yields a maximum value of 6. Thus, t_3 fires and sets y to 0. Since this analysis gives the largest value of y, y never reaches 7, so the transition t_4 never fires.

Even though the failure transition t_4 never fires, due to the over-approximations used by zones and warping, the reachable state space for the system in Fig. 5.3, using zones, indicates that the failure transition does fire, as is shown in Figs. 5.4, 5.5, and 5.6. Note that each of these figures depicts the portion of the zone in the xy-plane. Fig. 5.4a shows the initial zone where $0 \leq x \leq 1$ and $0 \leq y \leq 1$. In this zone, transition t_0 is enabled and has a delay of $[0, 1]$ time units. Thus, time can advance up to 1 time unit before t_0 fires. The resulting zone is shown in Fig. 5.4b.

After t_0 fires, y is assigned a rate of -1. This rate change flips the zone across the x-axis (Fig. 5.4c), which results in a subset that is not a zone. To make the subset into a zone, the

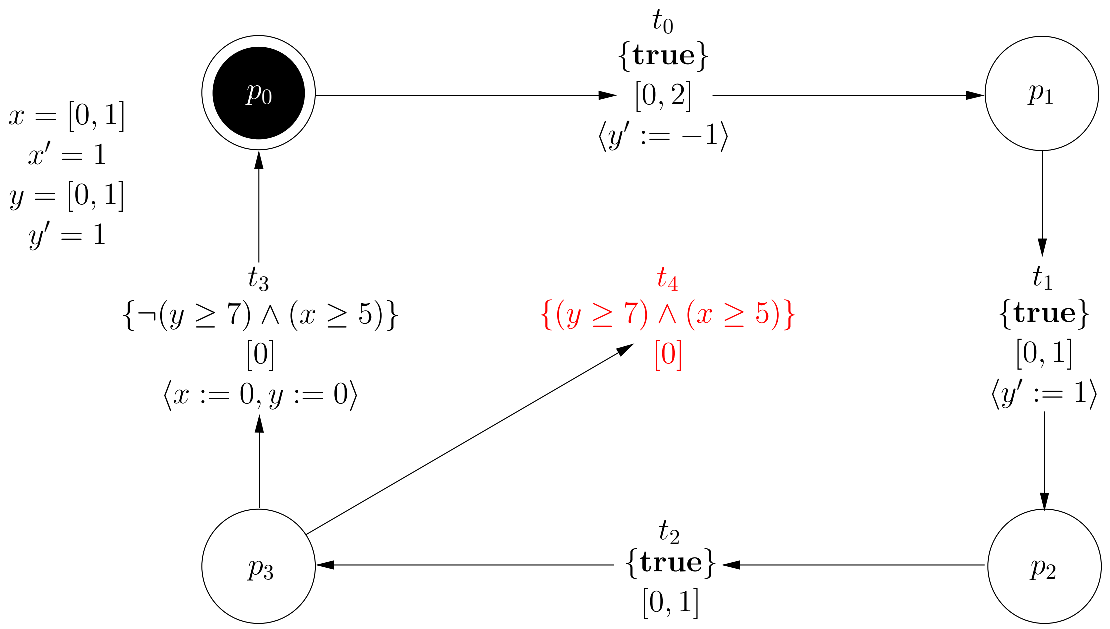

$x = [0,1]$
$x' = 1$
$y = [0,1]$
$y' = 1$
p_0
t_0
{**true**}
$[0,2]$
$\langle y' := -1 \rangle$
p_1
t_3
$\{\neg(y \geq 7) \wedge (x \geq 5)\}$
$[0]$
$\langle x := 0, y := 0 \rangle$
t_4
$\{(y \geq 7) \wedge (x \geq 5)\}$
$[0]$
t_1
{**true**}
$[0,1]$
$\langle y' := 1 \rangle$
p_3
t_2
{**true**}
$[0,1]$
p_2

Figure 5.3: A model where zones leads to a false negative result.

subset is filled out to the rectangle shown in Fig. 5.5a. In this zone, t_1 is enabled and has a delay of $[0, 1]$. So, the zone can advance up to 1 time unit (Fig. 5.5b). After transition t_1 fires, the rate of y is set back to 1, yielding Fig. 5.5c. This subset is again not a zone and has to be filled out to the rectangle in Fig. 5.6a. In this zone, transition t_2 is enabled and has a delay of $[0, 1]$, thus the zone can advance up to 1 time unit, as shown in Fig. 5.6b. Finally, after t_2 fires, the zone can advance until x is 5. The result of this advancement is shown in Fig. 5.6c. The upper right corner of this zone is the point $x = 5$ and $y = 7$, thus enabling the failure transition t_4.

The over-approximations that lead to this spurious error are directly due to the need to approximate a subset with a rectangle when the rate changes sign. With octagons, it is not necessary to over-approximate the space with rectangles. In fact, every subset in Figs. 5.4, 5.5, and 5.6 is an octagon including Figs. 5.4c and 5.5c. The sequence of octagons for Fig. 5.3 is shown in Figs. 5.7, 5.8, and 5.9. The initial octagon is shown in Fig. 5.7a, followed by the octagon resulting from advancing time 1 unit (Fig. 5.7b) and firing transition t_0 (Fig. 5.7c). In this case, the assignment of a rate of -1 to y is still a octagon, so there is no need to fill it out to a rectangle, as in the case of zones. Fig. 5.8a shows the result of advancing the octagon 1 time unit. After transition t_1 fires, the octagon in Fig. 5.8b is produced and, again, there is no need to fill out to a rectangle. Time is, again, advanced 1 time unit before t_2 fires (Fig. 5.8c). After t_2 fires, time is advanced until $x = 5$, producing

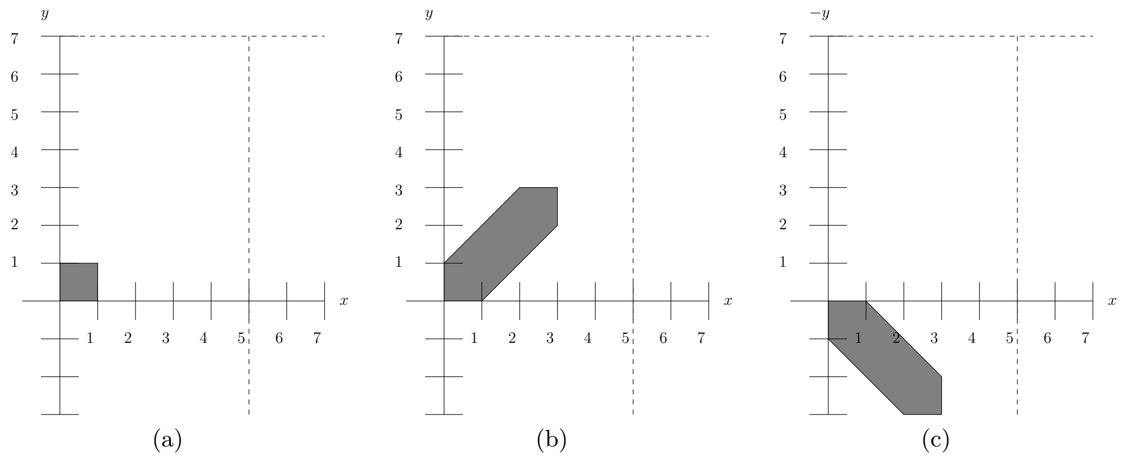


Figure 5.4: The first sequence of zones for Fig. 5.3. a) The initial zone. b) Zone after advancing 2 time units. c) Subset after firing t_0 and assigning y a rate of -1.

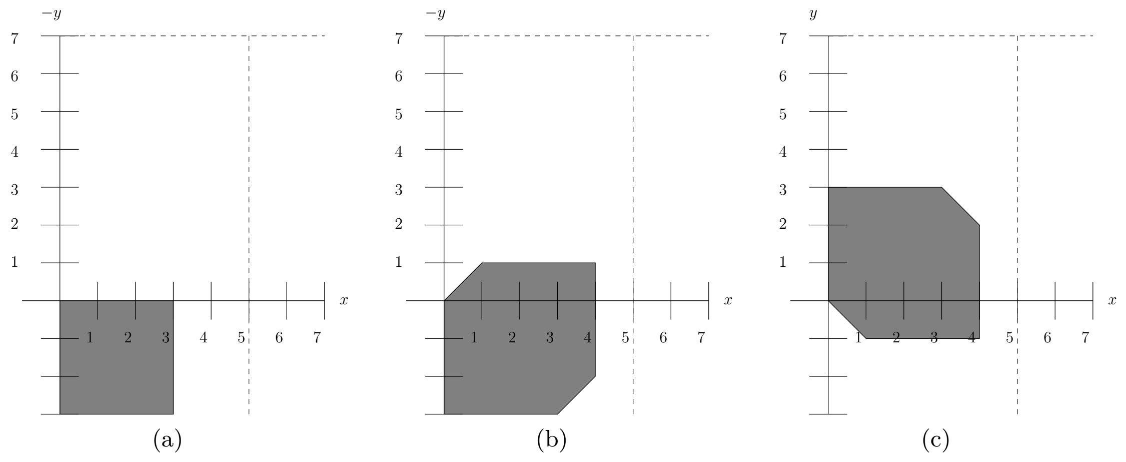


Figure 5.5: The second sequence of zones for Fig. 5.3. a) Zone after performing warping on Fig. 5.4c. b) Zone after advancing time 1 unit. c) Subset after firing transition t_1 and assigning y a rate of 1.

the octagon in Fig. 5.9a. This time, when $x = 5$, y is less than 7 and transition t_3 fires instead of t_4. After t_3 fires, the octagon in Fig. 5.9b is produced, which is the same as the initial octagon in Fig. 5.7a, and the process repeats.

5.2 Theory

This section introduces the necessary background for octagons, as well as the theory behind the required algorithmic changes. In changing from zones to octagons, the basic structure of the algorithms remain the same. The major differences come from interacting

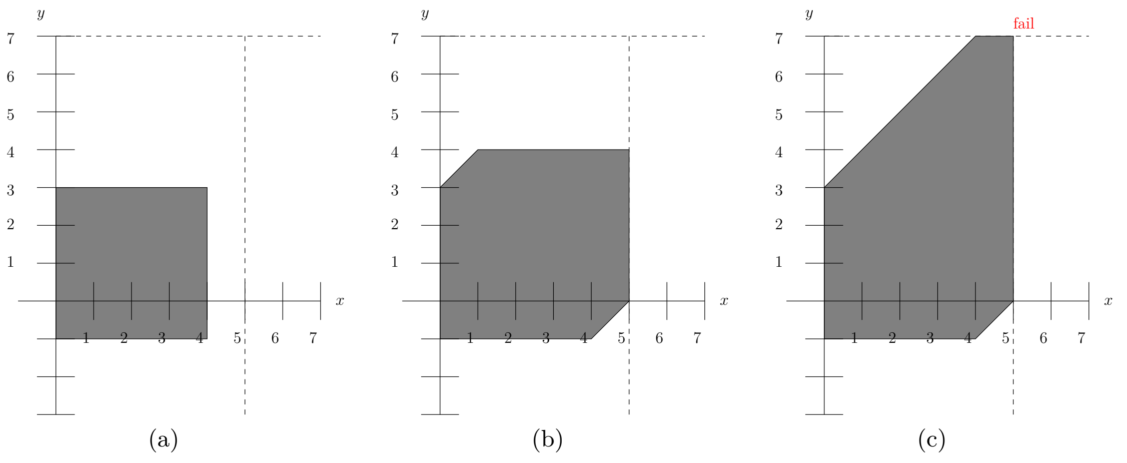


Figure 5.6: The third sequence of zones for Fig. 5.3. a) Zone after performing warping on Fig. 5.5c. b) Zone after advancing 1 time unit. c) Zone after firing transition t_2 and advancing time τ_{max} units.

with the new representation. Specifically, it must be explored how new variables are added to the octagon, how time advances, and how warping is performed.

Accordingly, this section starts with discussing the DBM representation in Section 5.2.1. Then, Section 5.2.2 describes how new variables are added, Section 5.2.3 describes the new method for time advancements, and Section 5.2.4 describes warping.

5.2.1 DBM Representation

Let $V_1, \ldots, V_n$ be continuous variables. An octagon is a subset of $\mathbb{R}^n$ formed by intersecting the hyperplanes of the form $\pm V_i \pm V_j \leq c_{i,j}$ for some constants $c_{i,j}$. Just like for zones, octagons can be represented as DBMs [81]. The first key step to the DBM representation is to introduce a positive variable V_i^+ and a negative variable V_i^- for each continuous variable V_i. The positive variables satisfy $+V_i = V_i^+$ and the negative variables satisfy $-V_i = V_i^-$. With these variables, every inequality $\pm V_i \pm V_j \leq c$ can be written in the form $Y - X \leq c$:

$$\begin{aligned}
V_j - V_i \leq c &\leftrightarrow V_j^+ - V_i^+ \leq c & -V_j + V_i \leq c &\leftrightarrow V_j^- - V_i^- \leq c \\
&\leftrightarrow V_i^- - V_j^- \leq c & &\leftrightarrow V_i^+ - V_j^+ \leq c \\
V_j + V_i \leq c &\leftrightarrow V_j^+ - V_i^- \leq c & -V_j - V_i \leq &\leftrightarrow V_j^- - V_i^+ \leq c \\
&\leftrightarrow V_i^+ - V_j^- \leq c & &\leftrightarrow V_i^- - V_j^+ \leq c \\
V_i \leq M_{V_i} &\leftrightarrow V_i^+ - V_i^- \leq 2M_{V_i} & V_i \geq m_{V_i} &\leftrightarrow V_i^- - V_i^+ \leq -2m_{V_i},
\end{aligned}$$

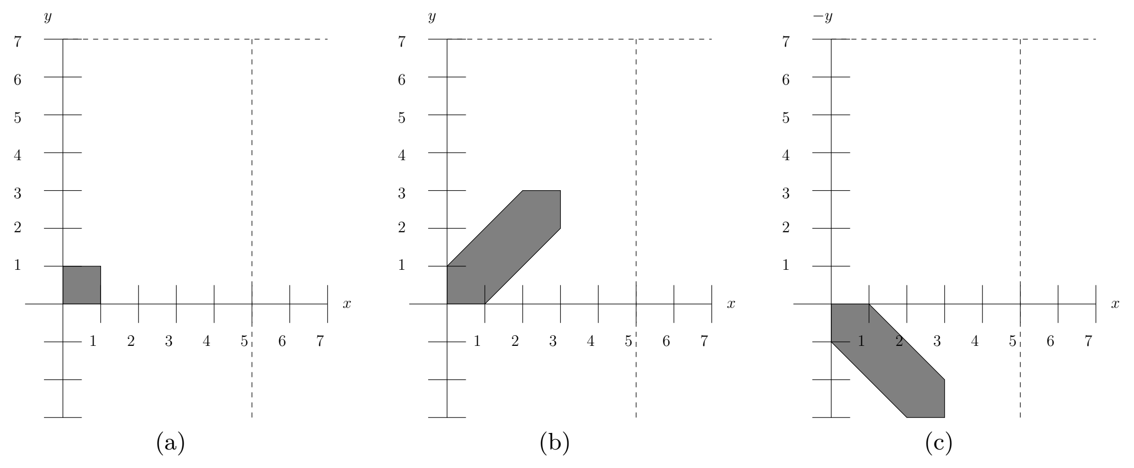


Figure 5.7: The initial sequence of octagons for Fig. 5.3. a) Initial octagon. b) Octagon after advancing time 1 time unit. c) Octagon after firing transition t_0 and assigning a rate of -1 to y.

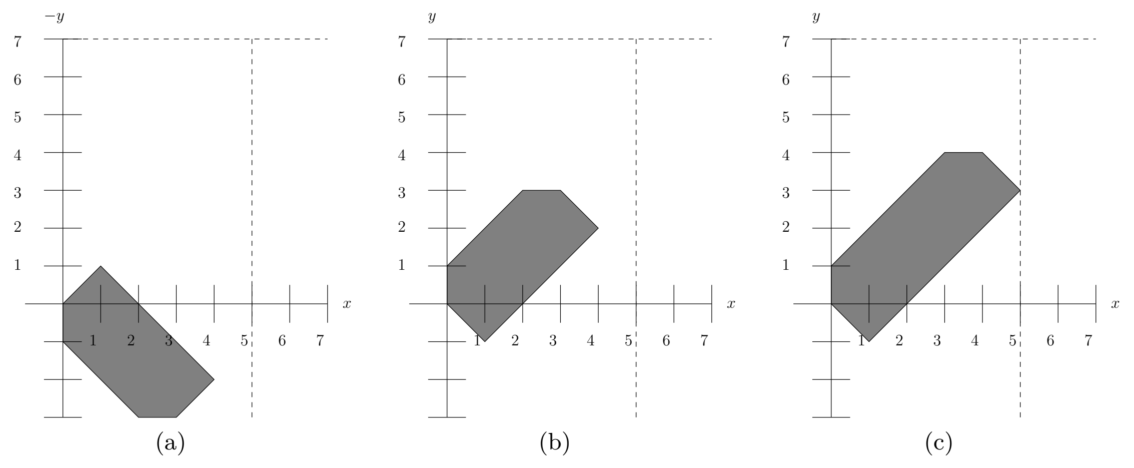
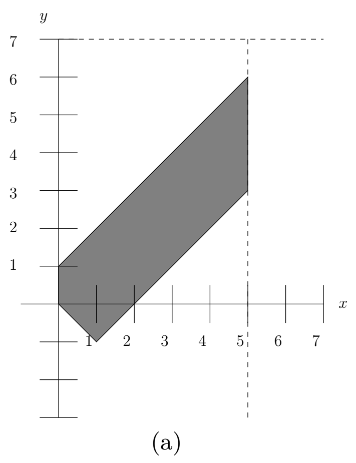


Figure 5.8: The second sequence of octagons for Fig. 5.3. a) Octagon after advancing time 1 time unit from the octagon in Fig. 5.7c. b) Octagon after firing transition t_1 and assigning a rate of 1 to y. c) Octagon after advancing time 1 time unit.

where m_{V_i} is the minimum of V_i and M_{V_i} is the maximum value of V_i. Since the minimum and maximum values can be written in the same for as the rest of the constraints, there is no need for a zero timer like with zones.

For N variables, $V_1, \ldots V_n$, there are $2N$ variables in the DBM representation: the variables $V_1^+, V_1^-, V_2^+, V_2^-, \ldots, V_n^+, V_n^-$. Thus, the DBM is a $2N \times 2N$ matrix M. Given an index i for a variable V_i, the $2i$ row/column index corresponds to V_i^+ and the $2i+1$ row/column index corresponds to V_i^-. So, the positive variables are the even indices in the DBM matrix and the negative variables are the odd indices in the DBM matrix. Given



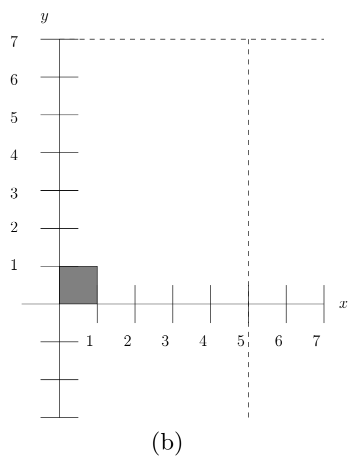


Figure 5.9: The third sequence of octagons for Fig. 5.3. a) Octagon after firing transition t_2. Notice that transition t_4 is not enabled. b) Octagon after transition t_3 fires.

an index i for the DBM, one can convert between the positive and negative indices via the function $\bullet \Longmapsto \bar{\bullet}$ given by $\bar{i} = i \oplus 1$, where $\oplus$ is the *bit-wise exclusive or* operator. Literally, this function flips the last bit of the binary representation for i. Thus, if i is even, $\bar{i} = i + 1$ and if $\bar{i}$ is odd, $\bar{i} = i - 1$. Hence, if i is the index in the DBM of V_j^+, that is if $i = 2j$, then $\bar{i}$ is the index of V_j^-, that is, $i = 2j + 1$. Similarly, if i is the index of V_j^-, then $\bar{i}$ is the index of V_j^+.

As with zones, the DBM collects together the constants of the constraints $Y - X \leq c$. If the entries of the matrix are $m_{i,j}$, the connection is established by:

$$V_i^+ - V_j^+ \leq m_{2i,2j} \qquad V_i^- - V_j^+ \leq m_{2i+1,2j}$$
$$V_i^+ - V_j^- \leq m_{2i,2j+1} \qquad V_i^- - V_j^- \leq m_{2i+1,2j+1}.$$

In addition, the entries $m_{2i,2i}$ and $m_{2i+1,2i+1}$ should always be 0 since they correspond to $V_i^+ - V_i^+ \leq 0$ and $V_i^- - V_i^- \leq 0$, respectively. The DBM has some redundancy since almost every equation corresponds to another equation that conveys the same information. For example, $V_j^+ - V_i^+ \leq c$ imposes the same constraint on $V_j - V_i$ as $V_i^- - V_j^- \leq c$. A DBM is called *coherent*, if these redundant entries are equal. Specifically, a DBM is consistent, if and only if, for all DBM indices i and j, the entries satisfy:

$$m_{i,j} = m_{\bar{j},\bar{i}}.$$

As a concrete example, consider the octagon in Fig. 5.10, repeated from Fig. 5.8b. The variables x and y are replaced with V_0 and V_1, respectively, to aid in following the indices. The inequalities that bound this octagon are as follows:

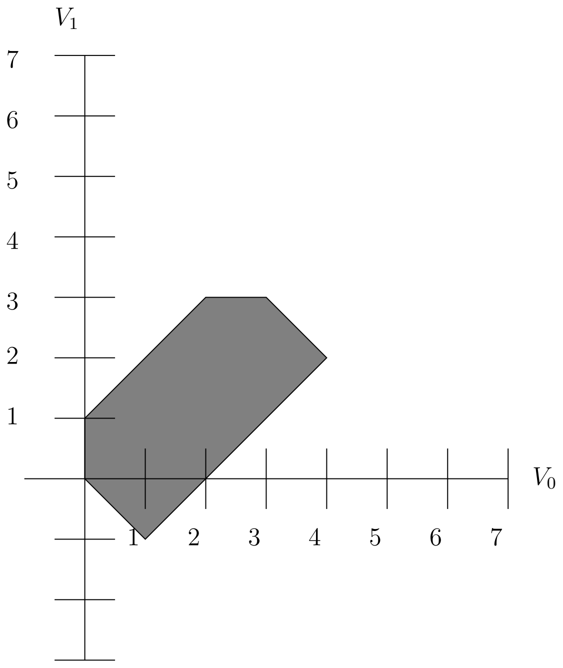


Figure 5.10: A generic octagon.

$$\begin{aligned} V_1 - V_0 &\le 1 & -V_1 + V_0 &\le 2 \\ V_1 + V_0 &\le 8 & -V_1 - V_0 &\le 0 \\ 0 \le V_0 &\le 5 & -1 \le V_1 &\le 4. \end{aligned}$$

After changing these equations into their equivalent forms using the positive and negative variables, the equations become:

$$\begin{array}{llll} V_0^+ - V_0^+ \le 0 & V_0^- - V_0^+ \le 0 & V_1^+ - V_0^+ \le 1 & V_1^- - V_0^+ \le 0 \\ V_0^+ - V_0^- \le 10 & V_0^- - V_0^- \le 0 & V_1^+ - V_0^- \le 8 & V_1^- - V_0^- \le 2 \\ V_0^+ - V_1^+ \le 2 & V_0^- - V_1^+ \le 0 & V_1^+ - V_1^+ \le 0 & V_1^- - V_1^+ \le 2 \\ V_0^+ - V_1^- \le 8 & V_0^- - V_1^- \le 1 & V_1^+ - V_1^- \le 8 & V_1^- - V_1^- \le 0. \end{array}$$

Collecting these values into a matrix yields the DBM:

$$D_1 = \begin{array}{c c} & \begin{array}{cccc} V_0^+ & V_0^- & V_1^+ & V_1^- \end{array} \\ \begin{array}{c} V_0^+ \\ V_0^- \\ V_1^+ \\ V_1^- \end{array} & \begin{pmatrix} 0 & 0 & 1 & 0 \\ 10 & 0 & 8 & 2 \\ 2 & 0 & 0 & 2 \\ 8 & 1 & 8 & 0 \end{pmatrix} \end{array}.$$

For this matrix to be coherent, the entries must satisfy $m_{0,0} = m_{1,1}$, $m_{0,1} = m_{0,1}$, $m_{1,0} =$

$m_{1,0}$, $m_{0,2} = m_{3,1}$, $m_{0,3} = m_{4,1}$, $m_{1,2} = m_{3,}$, $m_{1,3} = m_{4,2}$, and $m_{2,2} = m_{3,3}$, which this DBM satisfies.

5.2.2 Adding A New Variable

Adding new continuous variables and timers is simply a matter of reinterpreting the algorithms for zones in the language of octagons. For example, when adding a continuous variable v with rate r, the maximum and minimum values for v are divided by r and added to the DBM (after multiplying by 2 due to the way octagons store these values). Then, the relational entries are set to infinity, indicating no relationship. This section describes the necessary details.

Suppose V_i is a clock or a continuous variable in the octagon, and V_j is a new clock to be added to the Octagon. Since the new clock is initialized to 0, the new timer satisfies:

$$V_j \leq 0 \qquad\qquad -V_j \leq 0,$$

and thus, V_j^+ and V_j^- satisfy:

$$V_j^+ \leq 0 \qquad -V_j^+ \leq 0 \qquad V_j^- \leq 0 \qquad -V_j^- \leq 0.$$

Furthermore, the old variable satisfies:

$$V_i^+ - V_i^- \leq 2M_{V_i} \qquad\qquad V_i^- - V_i^+ \leq -2m_{V_i},$$

and so:

$$V_i^+ \leq M_{V_i} \qquad V_i^- \leq -m_{V_i} \qquad -V_i^+ \leq -m_{V_i} \qquad -V_i^- \leq M_{V_i}.$$

Combining these inequalities gives:

$$\begin{array}{llll} V_j^- - V_i^- \leq M_{V_i} & V_j^+ - V_i^- \leq M_{V_i} & V_i^+ - V_j^- \leq M_{V_i} & V_i^+ - V_j^+ \leq M_{V_i} \\ V_j^- - V_i^+ \leq -m_{V_i} & V_j^+ - V_i^+ \leq -m_{V_i} & V_i^- - V_j^- \leq -m_{V_i} & V_i^- - V_j^+ \leq -m_{V_i}. \end{array}$$

These equations define the required relationships between a new clock, V_j, and an old clock or continuous variable, V_i. If V_i and V_j are both new timers added at the same time, then all relations are 0, as can be seen by noting that, in this case, $m_{V_i} = 0$ and $M_{V_j} = 0$. Next, suppose V_i is a clock or a continuous variable already in the octagon, and V_j is a new continuous variable. Since V_j is a continuous variable, the minimum and maximum values are not required to be 0. Thus, suppose V_j satisfies the following conditions for the upper and lower bounds:

$$V_j \leq M_{V_j} \qquad\qquad -V_j \leq -m_{V_j},$$

where M_{V_j} and m_{V_j} are not necessarily 0, and so:

$$V_j^+ \leq M_{V_j} \qquad -V_j^+ \leq -m_{V_j} \qquad V_j^- \leq -m_{V_j} \qquad -V_j^- \leq M_{V_j}.$$

It follows that:

$$V_j^+ - V_j^- \leq 2M_{V_j} \qquad\qquad V_j^- - V_j^+ \leq -2m_{V_j}.$$

For the relations between V_i and V_j, no constraints are assumed, just like in the case for zones. Accordingly, the constraints are:

$$\begin{array}{llll} V_j^- - V_i^- \leq \infty & V_j^+ - V_i^- \leq \infty & V_i^+ - V_j^- \leq \infty & V_i^+ - V_j^+ \leq \infty \\ V_j^- - V_i^+ \leq \infty & V_j^+ - V_i^+ \leq \infty & V_i^- - V_j^- \leq \infty & V_i^- - V_j^+ \leq \infty. \end{array}$$

These constraints assume no relation between the variables. Usually, the tightening routine is used next to find the best constraints.

5.2.3 Time Advancement

Time advancement for octagons is not quite as simple as it is for zones. With a zone, one sets the upper bounds for the timers to the largest values possible given their ranges of delays, and the continuous bounds are set to the largest possible values that can be obtained without an inequality changing truth value. Furthermore, if the original zone is exact, then the time advancement is exact, ignoring over-approximations due to some constants not being evenly divisible by the current rate. The main point is that performing time advancement on a zone produces a zone. With octagons, one can still set the timers to (twice) their upper bounds and set the continuous variables' upper bounds to (twice) the largest possible values that can be obtained without changing the truth value of an inequality. However, unlike with zones, an octagon must have one of the intercepts values adjusted as well, namely, the intercept associated with the upper right diagonal. Moreover, the time advancement process itself may introduce an over-approximation due to the exact subset no longer being an octagon.

As a concrete example, consider the three continuous variables V_1, V_2, and V_3, all with a rate of 1. Let $\mathcal{O}$ be the octagon consisting of the line segment joining $(V_1, V_2, V_3) = (1, 0, 0)$ and $(V_1, V_2, V_3) = (0, 1, 0)$ (Fig. 5.11a). This octagon is defined by:

$$\begin{array}{rrr}
0 \leq V_1 \leq 1 & 0 \leq V_2 \leq 1 & 0 \leq V_3 \leq 0 \\
-V_2 + V_1 \leq 1 & -V_3 + V_2 \leq 1 & -V_3 + V_1 \leq 1 \\
V_2 - V_1 \leq 1 & V_3 - V_2 \leq 0 & V_3 - V_1 \leq 0 \\
V_2 + V_1 \leq 1 & V_3 + V_2 \leq 1 & V_3 + V_1 \leq 1 \\
-V_2 - V_1 \leq -1 & -V_3 - V_2 \leq 0 & -V_3 - V_1 \leq 0.
\end{array}$$

Fig. 5.11b shows the exact set of states reachable from this octagon by allowing time to advance up to 4 time units. Mathematically, this subset can be described as $\mathcal{S} = \mathcal{O} \oplus \{(t, t, t) \mid t \in [0, 4]\}$, where $\oplus$ is the Minkowski sum $A \oplus B = \{a + b \mid a \in A \wedge b \in B\}$. Intuitively, the effect of this sum is to allow the points in $\mathcal{O}$ to flow forward between 0 and 4 time units in the direction of the vector $\langle 1, 1, 1 \rangle$, that is, the vector that forms a positive $45°$ angle with each axis. Although the initial subset is an octagon, the subset in Fig. 5.11b is not an octagon. Before presenting a proof of this fact, note that $\mathcal{S}$ consists of the points:

$$\mathcal{S} = \{(1 - s + t, s + t, t) \mid s \in [0, 1] \wedge t \in [0, 4]\}.$$

The projection of this set onto the V_1, V_3-plane is:

$$\mathcal{S}_{V_1,V_3} = \{(1 - s + t, t) \mid s \in [0, 1] \wedge t \in [0, 4]\}.$$

From this set, it is seen that the difference $V_3 - V_1 = t - (1 - s + t) = s - 1$ is maximized over $s \in [0, 1]$ when $s = 1$, yielding $V_3 - V_1 \leq 0$. Similarly, the difference $-V_3 + V_1 = -t + (1 - s + t) = 1 - s$ is maximized over $s \in [0, 1]$ when $s = 0$, yielding $V_1 - V_3 \leq 1$. By a similar analysis, $V_3 + V_1$ and $-V_3 - V_1$ must satisfy $V_3 + V_1 \leq 9$ and $-V_3 - V_1 \leq 0$. Thus, any octagon containing $\mathcal{S}$ cannot have constraints tighter than:

$$V_3 - V_1 \leq 0 \qquad -V_3 - V_1 \leq 1 \qquad V_3 + V_1 \leq 9 \qquad -V_3 - V_1 \leq 0.$$

When the same processes is applied to the projections onto the V_2, V_3-plane and V_1, V_2-plane, the tightest constraints for the other pairs of continuous variables are:

$$\begin{array}{llll}
V_3 - V_2 \leq 0 & -V_3 - V_2 \leq 1 & V_3 + V_2 \leq 9 & -V_3 - V_2 \leq 0 \\
V_2 - V_1 \leq 1 & -V_2 + V_1 \leq 1 & V_2 + V_1 \leq 9 & -V_2 - V_1 \leq -1.
\end{array}$$

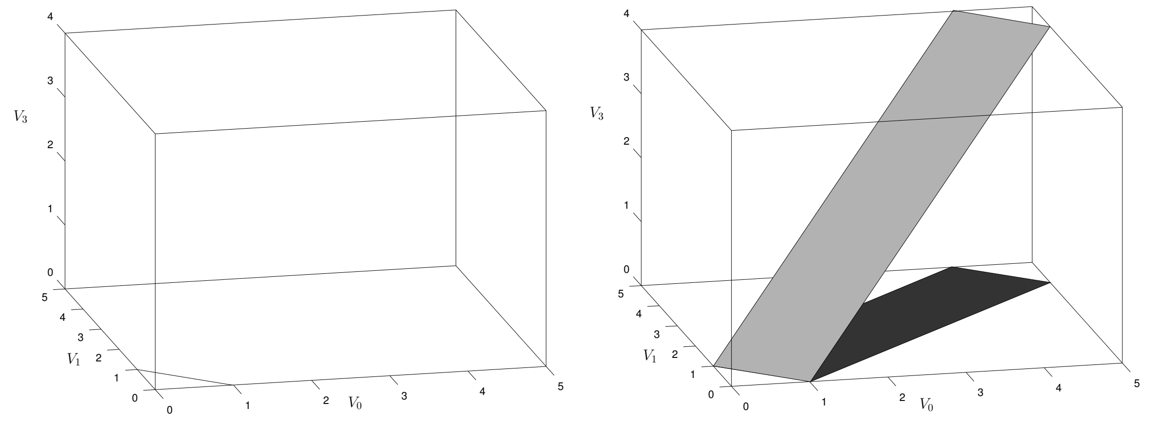


Figure 5.11: Exact space obtained by advancing the octagon consisting of the line joining $(1, 0, 0)$ and $(0, 1, 0)$ forward 4 time units.

Finally, the extreme values for each variable are:

$$0 \leq V_1 \leq 5 \qquad\qquad 0 \leq V_2 \leq 5 \qquad\qquad 0 \leq V_3 \leq 4.$$

Combining these inequalities gives the smallest octagon that contains $\mathcal{S}$. The octagon that they describe is shown in Fig. 5.12, which contains more than the original plane depicted in Fig. 5.11b. In other words, the octagon contains the subset $\mathcal{S}$, but it is not equal to $\mathcal{S}$. To be explicit, the point $(1, 1, 1)$ belongs to the octagon, but it does not belong to $\mathcal{S}$. If $(1, 1, 1)$ is in $\mathcal{S}$, then t would be 1, since $1 = V_3 = t$. This fact, in turn, implies that $s = 0$, since $1 = V_2 = t + s = 1 + s$. However, then V_1 is given by $1 - s + t = 1 - 0 + 1 = 2$, contradicting $V_1 = 1$.

Although it is sometimes necessary to over-approximate the subset obtained by advancing time, the over-approximation is no worse than what would be necessary for zones. In fact, the best over-approximating zone is found by taking the constraints of the form $Y - X \leq c$ from the octagon constraints. Furthermore, the algorithm for producing the time advanced octagon is as simple as for zones. Similar to zones, advancing time nearly amounts to setting the upper bound on each variable to the largest permissible value and then recanonicalizing. The only exception is that the constraints of the form $X + Y \leq c$ have to be adjusted as well, since they limit the forward progress of the variables involved. In two dimensions, these constraints correspond to the $-45°$ lines on the upper right corner of the octagon. If these constraints were not adjusted, then the octagon would be prevented from advancing. For example, the octagon of Fig. 5.12 can be obtained from $\mathcal{O}$ by setting the upper bounds on the variables V_1, V_2, and V_3 to 5, 5, and 4 and by adjusting the constraints $V_3 + V_1 \leq 1$, $V_2 + V_1 \leq 1$, and $V_2 + V_1 \leq 1$ to $V_3 + V_1 \leq 9$, $V_2 + V_1 \leq 9$, and $V_2 + V_1 \leq 9$.

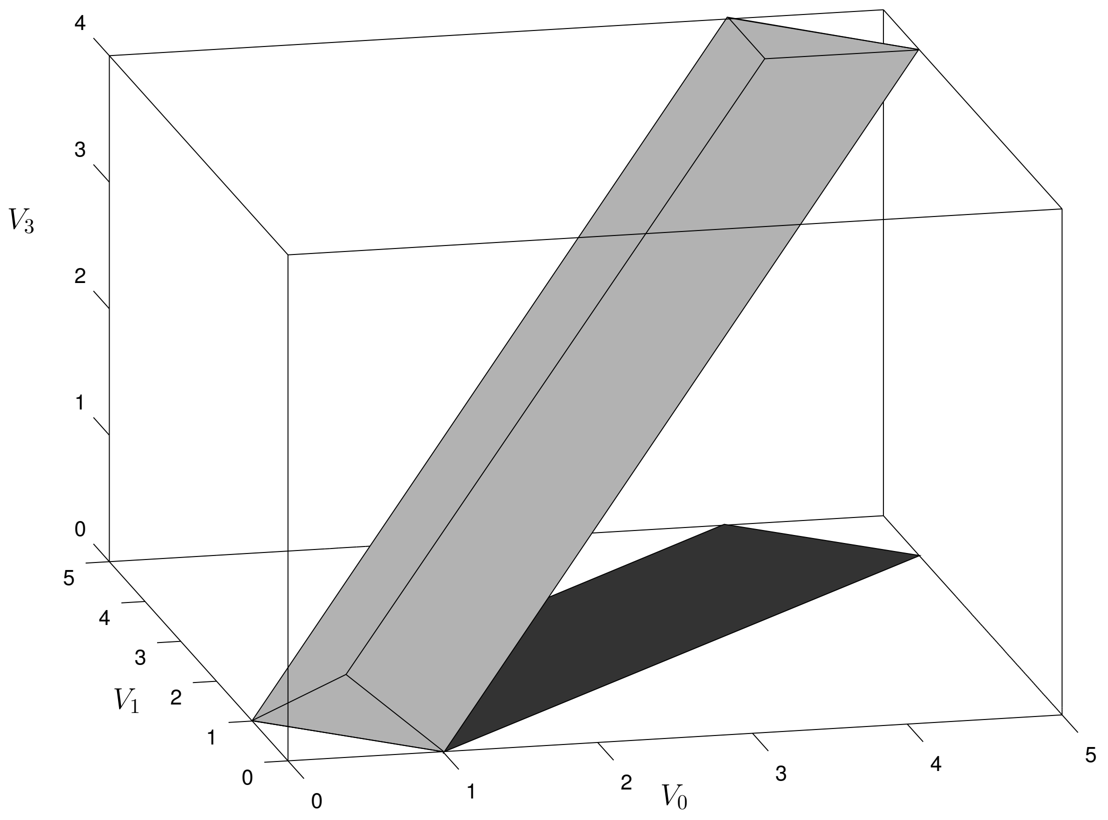


Figure 5.12: Over-approximation required for advancing the octagon consisting of the line joining $(1, 0, 0)$ and $(0, 1, 0)$ forward 4 time units.

Notice that if these constraints remain $V_3 + V_1 \leq 1$, $V_2 + V_1 \leq 1$, and $V_2 + V_1 \leq 1$, then the upper bounds can be tightened back to $V_1 \leq 5$, $V_2 \leq 5$, and $V_3 \leq 4$. For example, adding the inequalities $V_2 + V_1 \leq 1$ and $-V_2 + V_1 \leq 1$ yields $2V_1 \leq 2$ or $V_1 \leq 1$, which is the same as for the original octagon $\mathcal{O}$, thus preventing the variable V_1 from advancing.

Although, nonexact time advancement is a problem that arises when three or more variables are present, the upper right constraint is present in two dimensions. Consider the octagon in Fig. 5.13a. Suppose the upper bound of x is set to 5 for the octagon and the upper right negative 45° line is not moved. Then, this negative sloped line limits the growth of x to no more than 4, resulting in the octagon not changing at all. However, if the upper bound of x is set to 5, the upper bound of y is set to 4, and the upper right constraint, $y + x \leq 6$, is moved to $y + x \leq 7$, then Fig. 5.13b is produced. With the upper right constraint adjusted time is allowed to move 1 unit. Thus, when advancing time, not only do the upper bounds on the variables have to be set to their maximum allowed values, but the entries associated with the inequalities $y + x \leq c$ must be adjusted.

So, to advance time, one sets the upper bounds to the maximum allowable value and

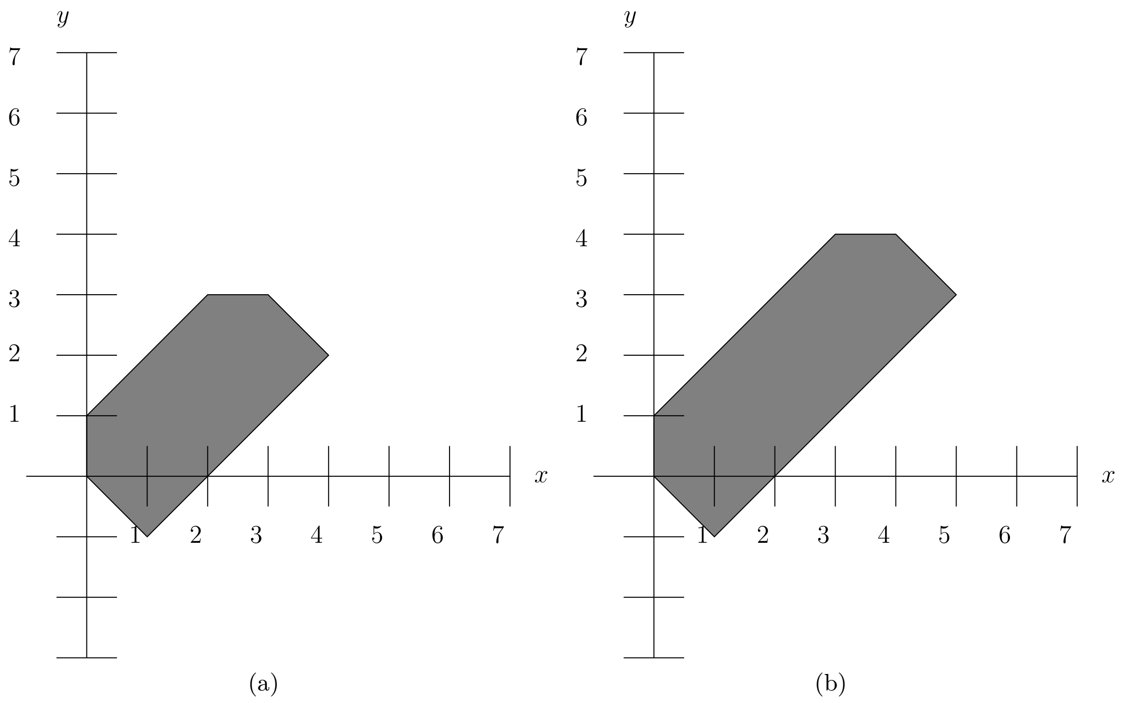


Figure 5.13: Two dimensional time advancement of an octagon. (a) Octagon before time advancement. (b) One unit time advancement of the octagon in (a).

adjusts the $Y + X \leq c$ constraints. For this chapter, the constraints $Y + X \leq c$ are set to $Y + X \leq \infty$, that is, the constraints are effectively removed. By removing the constraint, time advancement is not restricted. In general, removing the constraint is an additional over-approximation; however, it is no worse of an over-approximation than if zones are used alone, since a zone does not contain this type of constraint.

5.2.4 Warping

This section describes how warping is applied to octagons. Recall that warping is the method used to find the best approximating zone after a rate for a continuous variable has changed. Conceptually, determining how warping should be done for octagons is straightforward. Start by replacing every variable v by the scaled quantity $\frac{v}{r}$, where r is the rate of v, just as with zones. The resulting subset is, in general, not an octagon, so replace the scaled subset with the smallest octagon in which it is contained. Finding the over-approximating octagon amounts to solving a few algebraic equations that determine where the new axis intercepts are in terms of the old intercept values. This procedure is much the same as presented in [72] for warping zones, though the derivation used by this chapter is slightly different. Half the equations involved for the positive rate case are the same as for zones. These equations are the ones that handle the upper and lower bounds, as well as the positive 45° constraints. The other equations are introduced to handle the negative 45° lines.

Warping only involves two variables, so let V_i and V_j be two continuous variables with rates r_i and r_j. Since the values in the octagon are scaled, the variables used for the DBM are $x = \frac{V_i}{r_i}$ and $y = \frac{V_j}{r_j}$. Fig. 5.14a shows an arbitrary octagon in the x, y-plane. Now, suppose that V_i is assigned a rate of r'_i and V_j is a assigned a rate of r'_j. Then, the new scaling is $u = \frac{V_i}{r'_i}$ and $v = \frac{V_j}{r'_j}$, and the new figure is shown in Fig. 5.14b. Let $\alpha = \frac{r_i}{r'_i}$ and $\beta = \frac{r_j}{r'_j}$, then α and β transform the x, y-plane into the u, v-plane, that is,

$$u = \alpha x \qquad\qquad v = \beta y.$$

As described in Sectionoct:sec:dbm, the octagon can be described as a DBM

$$D = \begin{array}{c} \\ x^+ \\ x^- \\ y^+ \\ y^- \end{array} \begin{array}{c} \begin{array}{cccc} x^+ & x^- & y^+ & y^- \end{array} \\ \begin{pmatrix} 0 & -2m_x & b_1 & -b_4 \\ 2M_x & 0 & b_3 & -b_2 \\ -b_2 & -b_4 & 0 & -2m_y \\ b_3 & b_1 & 2M_y & 0 \end{pmatrix} \end{array}$$

for Fig. 5.14a and

$$D' = \begin{array}{c} \\ u^+ \\ u^- \\ v^+ \\ v^- \end{array} \begin{array}{c} \begin{array}{cccc} u^+ & u^- & v^+ & v^- \end{array} \\ \begin{pmatrix} 0 & -2m_u & b'_1 & -b'_4 \\ 2M_u & 0 & b'_3 & -b'_2 \\ -b'_2 & -b'_4 & 0 & -2m_v \\ b_3 & b_1 & 2M_v & 0 \end{pmatrix} \end{array}$$

for Fig. 5.14b. In the DBM representation, the constants b_1, b_2, b_3, and b_4 are the y intercepts of the bounding lines:

$$y - x \leq b_1 \qquad -y + x \leq -b_2 \qquad y - x \leq b_3 \qquad -y - x \leq -b_4.$$

and the constants b'_1, b'_2, b'_3, and b'_4 are the u intercepts of the bounding lines:

$$v - u \leq b'_1 \qquad -v + u \leq -b'_2 \qquad v - u \leq b'_3 \qquad -v - u \leq -b'_4.$$

The constant with a minus sign are those that are defining lower bounds. The goal of warping is to determine m_u, m_v, M_u, M_v, b'_1, b'_2, b'_3 and b'_4 in terms of m_x, m_y, M_x, M_y, b'_1, b'_2, b'_3 and b'_4.

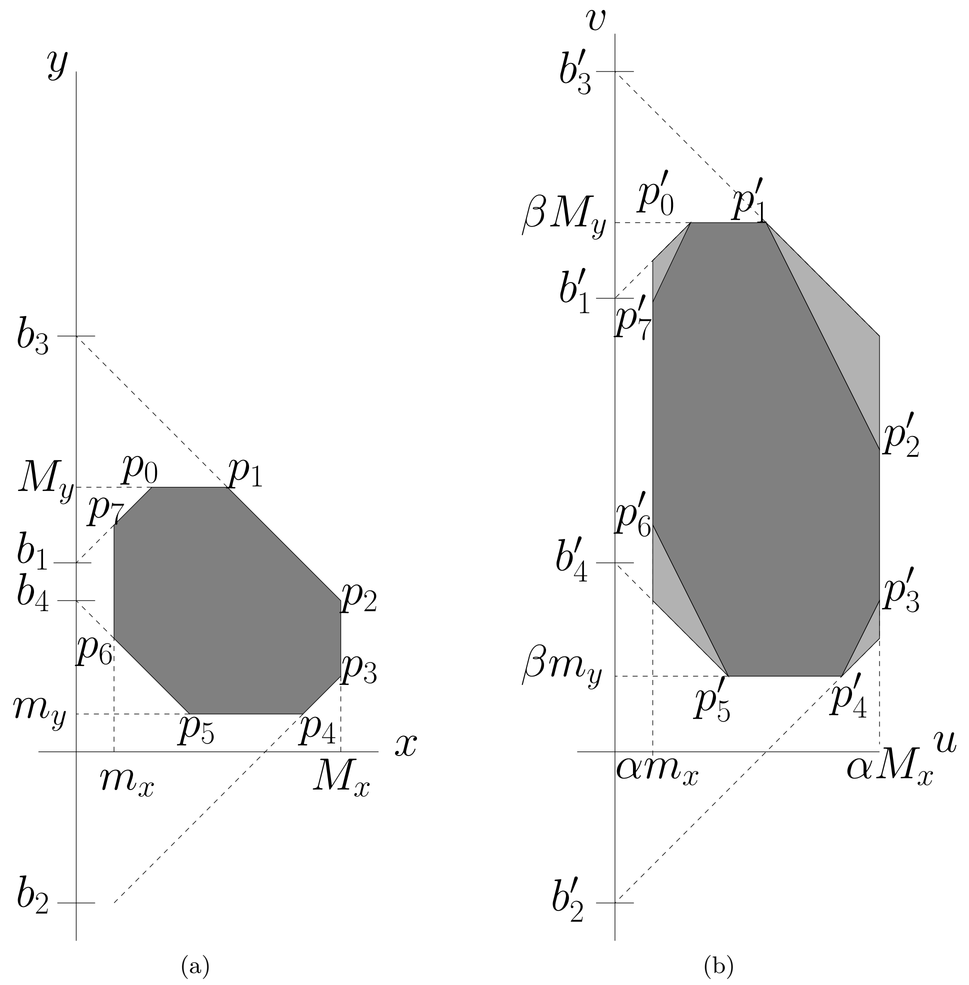


Figure 5.14: Labeled octagon. (a) An octagon with y-intercepts labeled b_1, b_2, b_3, and b_4, and vertices labeled $p_0, \ldots, p_7$. (b) Warped octagon with $\frac{\beta}{\alpha} > 1$.

The easiest values to determine are the new minimum and maximum values m_u, m_v, M_u, and M_v. If α is positive, then

$$m_u = \alpha m_x$$
$$M_u = \alpha M_x,$$

and if α is negative

$$M_u = \alpha m_x$$
$$m_u = \alpha M_x.$$

That is, the new minimum and maximum values are obtained by undoing the previous scaling and introducing the new scaling. Additionally, if α is negative, then the minimum and maximum values are swapped. The situation is analogous for m_v and M_v. If β is negative, then

$$m_v = \alpha m_y$$
$$M_v = \alpha M_y,$$

and if β is negative

$$M_v = \alpha m_y$$
$$m_v = \alpha M_y.$$

The intercepts require a little more care. One could simply do the same idea; however, this results in a larger octagon than necessary. Instead the new relations are given by the following possible two possible sets of equations. First assume that $\alpha > 0$ and $\beta > 0$. When $\frac{\beta}{\alpha} > 1$, the possible equations are:

$$b_1' = (\beta - \alpha)M_y + \alpha b_1$$
$$b_2' = (\beta - \alpha)m_y + \alpha b_2$$
$$b_3' = (\beta - \alpha)M_y + \alpha b_3$$
$$b_4' = (\beta - \alpha)m_y + \alpha b_4,$$

and, when $\frac{\beta}{\alpha} < 1$, the equations are:

$$b_1' = (\beta - \alpha)m_x + \beta b_1$$
$$b_2' = (\beta - \alpha)M_x + \beta b_2$$
$$b_3' = (\alpha - \beta)M_x + \beta b_3$$
$$b_4' = (\alpha - \beta)m_x + \beta b_4.$$

These equations are called the warping equations. When α is negative and β is positive, the equations are identical except the constants are interchanged according the following correspondence:

$$b_1 \mapsto b_3 \qquad b_2 \mapsto b_4 \qquad b_3 \mapsto b_1 \qquad b_4 \mapsto b_2.$$

Similarly, if α is positive and β is negative, the equations are identical except the constants are interchanged according the following correspondence:

$$b_1 \mapsto -b_4 \qquad b_2 \mapsto -b_3 \qquad b_3 \mapsto -b_2 \qquad b_4 \mapsto -b_1.$$

Finally, if α is negative and β is negative, the equations are identical except the constants are interchanged according the following correspondence:

$$b_1 \mapsto -b_2 \qquad b_2 \mapsto -b_1 \qquad b_3 \mapsto -b_4 \qquad b_4 \mapsto -b_3.$$

The full derivation of these equations is given in Appendix C; however, the idea is relatively straightforward. The basic idea is to find the x and y coordinates of each vertex using the boundary lines. Find the transformed u and v coordinates of the new subset by multiplying the x coordinates by α and the y coordinates by β. Use the new boundary lines and the coordinates of the vertex to solve for the new intercept. This scheme gives the warping equations. For determining how changing the signs affects these questions, one can perform the coordinate change on the the equations and then compare new the equations with the old equations.

5.3 Reachability Algorithm

The basic structure of the reachability algorithm remains the same as the depth-first search algorithm used for zones (Algorithm 4.1). Throughout the rest of the algorithms, minor differences are introduced based on how items are stored in the DBM for an octagon versus the DBM for zones. For example, in accessing the upper bound for a zone, one looks at the DBM entry $m_{i,0}$ associated with $V_i - t_0$, where t_0 is the zero clock, which always has a value of 0. For octagons, the upper bound entry is associated with $V_i^+ + V_i^+$ and is given by the DBM entry $m_{2i,2i}$. In this section, the algorithms that require a more substantial modification are presented. These algorithms include `AddT`, `AddV`, `advanceTime`, `dbmWarp`, and `recanonicalize`.

The algorithm `addT` shown in Algorithm 5.1 is simply a direct translation of the inequalities in Section 5.2.2. The upper and lower bounds for the newly enabled transitions, En_{new}, are set to zero and all relationships between the newly enabled transitions are set to zero. Finally, the relationships of Section 5.2.2 are set between the newly enabled transitions, the previously enabled transitions, and the continuous variables.

In this algorithm, $ub(O, x)$ is the entry of the octagon associated with the inequality $x^+ - x^- \leq c$ and gives twice the upper bound, while $nlb(O, x)$ is the entry of the octagon associated with the inequality $x^- - x^+ \leq c$ and gives twice the negative of the lower bound. The functions $O(x^\pm, y^\pm)$ give the entry associated with $y^\pm - x^{pm} \leq c$. With this function, $ub(O, x) = O(x^-, x^+)$ and $nlb(O, x) = O(x^+, x^-)$.

The algorithm for adding a continuous variable v is similar and is shown in Algorithm 5.2. Just like in the case of `addT`, the algorithm is a direct translation of the necessary inequalities

Algorithm 5.1: `addT`(O, En_{new})

```
forall the $t \in En_{new}$ do
    $ub(O,t) := 0$;
    $nlb(O,t) := 0$;
    forall the $s \in O$ do
        if $s \in En_{new}$ then
            $O(t^+, s^+) := 0$;
            $O(t^+, s^-) := 0$;
            $O(t^-, s^+) := 0$;
            $O(t^-, s^-) := 0$;
            $O(s^+, t^+) := 0$;
            $O(s^+, t^-) := 0$;
            $O(s^-, t^+) := 0$;
            $O(s^-, t^-) := 0$;
        else
            $O(t^+, s^+) := nlb(O,s)$;
            $O(t^+, s^-) := ub(O,s)$;
            $O(t^-, s^+) := nlb(O,s)$;
            $O(t^-, s^-) := ub(O,s)$;
            $O(s^+, t^+) := ub(O,s)$;
            $O(s^+, t^-) := ub(O,s)$;
            $O(s^-, t^+) := nlb(O,s)$;
            $O(s^-, t^-) := nlb(O,s)$;
```

in Section 5.2.2. The upper and lower bounds are scaled according to their rate. Since a change in sign flips the upper and lower bounds for a variable, when the rate is negative, the variables are scaled and the bounds are swapped. Then, the relationships between the newly added continuous variables and every variable in the octagon are set to ∞.

The algorithm for advancing time is `advanceTime` and is shown in Algorithm 5.3. The algorithm starts by setting the upper bound of every transitions t to the upper bound on the delay given by $d_u(t)$. Next, the upper bounds for each continuous variable v are set to the largest possible value before changing an inequality. This value is found by the function `checkIneq` as described in Appendix B. Finally, the inequalities $v^+ + v'^+ \leq b_3$ are removed for each pair of continuous variables by setting the bound to ∞. The assignment is performed by using the function `uc`(O, v, v') to access the element of O corresponding the inequality $v^+ + v'^+ \leq b_3$.

The DBM warping procedure for octagons is shown in Algorithm 5.4. The algorithm is in 2 stages. The first stage handles the warping according to the signs and the second stage handles the warping according to the rates. In the first stage, if the rate of a variable changes sign, then the upper and lower bounds must be swapped. This step is handled by

Algorithm 5.2: addV(Q, R, O, v)

```
O := O ∪ v;
if R(v) then
    ub(O, v) := 2 * cdiv(q_u(v), R(v));
    nlb(O, v) := 2 * cdiv(q_l(v), R(v));
else
    ub(O, v) := 2 * cdiv(q_l(v), R(v));
    nlb(O, v) := 2 * cdiv(q_u(v), R(v));
forall the s ∈ O do
    O(v^+, s^+) := ∞;
    O(v^+, s^-) := ∞;
    O(v^-, s^+) := ∞;
    O(v^-, s^-) := ∞;
    O(s^+, v^+) := ∞;
    O(s^+, v^-) := ∞;
    O(s^-, v^+) := ∞;
    O(s^-, v^-) := ∞;
```

Algorithm 5.3: advanceTime(En, D, R, I, O)

```
forall the t ∈ O do
    ub(O, t) := d_u(t);
forall the v ∈ O do
    ub(Z, v) := checkIneq(En, R, I, O, v);
    forall the v' ∈ O do
        uc(O, v, v') := ∞;
```

the function swapBounds(O, x). In addition to the bounds, changing the sign of the rates affects the relationships between pairs of variables. Thus, the second for-loop considers the ordered-pair of relations $y^{\pm} - x^{\pm}$. The constraints b_1, $-b_2$, b_3, and $-b_4$ are swapped according to which rates have changed. Thus, there is a case for whether the rate of x became negative, the rate of y became negative, or both rates became negative. The function swap(O, x, y, b_i, b_j) swaps the b_i and b_j constraints for the variables $y^{\pm} - x^{\pm}$. Which bounds need to be switched is explained in Section 5.2.1. The next for-loop considers pairs of variables x and y and handles the warping of the constraints according to the rates. As explained in Section 5.2.1, the equations are different depending on which ratio of rates is larger. To be succinct, the conditions are written according to the function warp given by:

$$\texttt{warp}(z_1, z_2, r_1, r_2) = (r_2 - r_1)z_1 + r_1 z_2. \tag{5.1}$$

Finally, the bounds are warped according to their new rates and the new octagon is returned.

Algorithm 5.4: `dbmWarp`(R, R', O)

```
forall the $x \in O$ do
    if $R(x)/R'(x) < 0$ then
        $O :=$ swapBounds$(O, x)$;
    forall the $y \in O \wedge y \neq x$ do
        if $R(x)/R'(x) < 0 \wedge R(y)/R'(y) > 0$ then
            $O :=$ swap$(O, x, y, b1, b3)$;
            $O :=$ swap$(O, x, y, b2, b4)$;
        else if $R(x)/R'(x) > 0 \wedge R(y)/R'(y) < 0$ then
            $O :=$ swap$(O, x, y, b1, b4)$;
            $O :=$ swap$(O, x, y, b1, b3)$;
        else if $R(x)/R'(x) < 0 \wedge R(y)/R'(y) < 0$ then
            $O :=$ swap$(O, x, y, b1, b2)$;
            $O :=$ swap$(O, x, y, b3, b4)$;
forall the $\{x, y\} \mid x \in O, y \in O, x \neq y$ do
    $\alpha := |$fdiv$(R(x), R'(x))|$;
    $\beta := |$fdiv$(R(y), R'(y))|$;
    if $\alpha > \beta$ then
        $O(x^+, y^+) :=$ warp$(ub(O, y), O(x^+, y^+), \alpha, \beta)$;
        $O(x^-, y^-) :=$ warp$(nlb(O, y)), O(x^-, y^-), \alpha, \beta)$;
        $O(x^-, y^+) :=$ warp$(ub(O, y), O(x^-, y^+), \alpha, \beta)$;
        $O(x^+, y^-) :=$ warp$(nlb(O, y), O(x^+, y^-), \alpha, \beta)$;
        $O(y^-, x^-) := O(x^+, y^+)$;
        $O(y^+, x^+) := O(x^-, y^-)$;
        $O(y^-, x^+) := O(x^-, y^+)$;
        $O(y^+, x^-) := O(x^+, y^-)$;
    else
        $O(x^+, y^+) :=$ warp$(nlb(O, x), O(x^+, y^+), \beta, \alpha)$;
        $O(x^-, y^-) :=$ warp$(ub(O, x)), O(x^-, y^-), \beta, \alpha)$;
        $O(x^-, y^+) :=$ warp$(ub(O, x), O(x^-, y^+), \beta, \alpha)$;
        $O(x^+, y^-) :=$ warp$(nlb(O, x), O(x^+, y^-), \beta, \alpha)$;
        $O(y^-, x^-) := O(x^+, y^+)$;
        $O(y^+, x^+) := O(x^-, y^-)$;
        $O(y^-, x^+) := O(x^-, y^+)$;
        $O(y^+, x^-) := O(x^+, y^-)$;
forall the $x \in O$ do
    nlb$(O, x) :=$ cdiv$(|R(x)|, |R'(x)|) * nlb(O, x)$;
    ub$(O, x) :=$ cdiv$(|R(x)|, |R'(x)|) * ub(O, x)$;
return $O$;
```

The recanonicalization routine is described in Algorithm 5.5. This routine is a direct translation of the algorithm presented in [12] for finding the tightest constraints. The algorithm is a simple modification of the version for zones. Indeed, the first set of for-loops is the Floyd's all-pair algorithm, the same as is used for zones. The second set of for-loops handles adjusting for the fact that the bounds are stored as twice the actual bound, while the rest of the constraints are not.

5.4 Experimental Results

As described in Section 5.1, zones over-approximate the exact reachable state space of an LPN. Since there are states that are not present in the actual system, it is possible that a system can erroneously fail verification due to these additional states violating the property instead of any actual states reachable by the system. By using octagons, some of these false reachable states can be removed. The LPN in Fig. 5.3 provides a concrete example where using zones results in a false negative, while using octagons provides the correct result (see Table 5.1).

After implementing the octagon algorithm in LEMA (Section 2.2), the zone and octagon model checkers are ran on the LPN in Fig. 5.3. The results are shown in Fig. 5.1. As expected, the zone-based method indicates that the system fails while the octagon-based method correctly indicates that the system passes verification.

Even though octagons are more accurate, the additional overhead for the representation is not substantial. The space requirements of an octagon are about twice that for a zone: $n+1$ versus $2n$ where n is the number of active timers and nonrate zero continuous variables. Furthermore, the algorithm with the highest complexity, warping, has the same complexity for octagons as for zones. So, although one expects octagons to be more costly, the cost is not unreasonable. To explore the additional overhead incurred by using octagons instead of zones, the verification scenarios of Chapter 4 are revisited below.

Recall that in Chapter 4, zones were applied to a sequence of capacitor models with each

Algorithm 5.5: `recanonicalize`(O)

```
1 for $k := 0$ to $2n-1$ do
2     for $i := 0$ to $2n-1$ do
3         for $j := 0$ to $2n-1$ do
4             $O(i,j) := \min(O(i,j), O(i,k) + O(k,j))$
5 for $i := 0$ to $2n-1$ do
6     for $j := 0$ to $2n-1$ do
7         $O(i,j) := \min(O(i,j), \mathrm{floor}(O(i,\bar{i})/2) + \mathrm{floor}(O(\bar{j},j)/2))$
```

Table 5.1: Results of running LEMA on the LPN in Fig. 5.3. The experiment was run on a 64-bit machine with a 2.90 GHz Intel Core i7-4910MQ CPU with 4 cores and 32GB of memory.

Method	States	Time (s)	Verifies?
Zones	6	0.013	no
Octagons	6	0.007	yes

stage modeled by the LPN in Fig. 4.2. Verification is preformed using three different failure conditions for the property in Fig. 4.5. In the case of Chapter 4, the point of these models is that the previous translational approach is not sound in general and that the algorithmic approach can sometimes find bugs faster, even with the extra overhead. In this section, the point of the examples is to show the cost of using octagons versus zones. Note: the octagon-based method is built on the zone-based and so it has the same ability to handle ranges of rates and, consequently, is also sound.

The first case that is considered is when the failure condition on $tFail$ in Fig. 4.5 is $\neg(V_i \geq 15)$. In this case, the failure condition never fires, so the algorithms perform a full state space exploration. The results of running the zones and octagons with this condition are shown in Table 5.2. For this example, it turns out that the same number of states is found by both algorithms and they both time-out at the same number of capacitor stages. Also, in each case verification does complete, the octagon-based method takes no more than 2 times as long to finish.

When the rate optimization is applied, the results are quite different, as seen in Table 5.3. In this example, the state counts are different with the octagons state count growing more rapidly than for zones. The runtimes are still comparable for these small examples; however, with the rising state counts, it is likely that the octagons runtime will rapidly outpace that for zones.

The number of events fired during the state exploration process provides additional insight into the cost of octagons versus zones. Table 5.4 shows the number of events fired for zones and octagons, with and without optimization. For the zones and octagons without the rate optimization, the number of events fired is the same; however, the runtime for the octagons is increasing more rapidly than for zones. Thus, the octagon approach is more costly than for zones. A similar result is suggested by comparing zones with rate optimization to octagons with rate optimization. Although, with octagons, the number events fired is less, the runtime for the octagon approach is still more than for zones.

For the second example, the failure condition is set to $V_i \geq 30$. In this case, a failure is

Table 5.2: Comparison of zones and octagons with a tFail enabling condition of $\neg(V_i \geq 18)$. All cases verify as correct. TIMEOUT indicates a runtime of more than 12 hours.

	Zones		Octagons	
# Caps	Time (s)	States	Time (s)	States
1	0.188	59	0.204	59
2	2.01	144	3.081	144
3	40.085	279	73.829	279
4	15311.948	1148	26858	1148
5	TIMEOUT	-	TIMEOUT	-

Table 5.3: Comparison of zones and octagons with a tFail enabling condition of $\neg(V_i \geq 18)$. All cases verify as correct.

	Zones (Optimization)		Octagons (Optimization)	
# Caps	Time (s)	States	Time (s)	States
1	0.108	35	0.118	55
2	0.457	56	0.512	140
3	0.941	65	1.042	262
4	2.954	105	4.809	1498
5	4.081	207	6.497	2122

Table 5.4: The total number of event firings for the property with tFail $V_i \geq 18$.

	Event Count ($V_i \geq 18$)			
# Caps	Zones	Zones (Opt)	Octagons	Octagons (opt)
1	313	106	313	106
2	29378	6584	29378	1576
3	805024	44372	805024	5997
4	174330848	15395081	174330848	59993
5	-	414627973	-	78603

expected. The results are shown in Table 5.5. As before, the zones find a failure, and the octagons find a failure. As with the first example, the state counts are the same; however, the runtime for octagons increases more rapidly, resulting in the last example not completing before reaching 5 hours. Again, the increase in the runtime is due to the cost of running the octagons algorithm and not due to an increase in the number of event firings since the number of event firings between the two approaches is the same (Table 5.6).

The final example has the failure condition on tFail as $\neg(V_i \geq 30)$. This case is the one which the translational approach indicated that the system passed verification, when the

Table 5.5: Comparison of zones and octagons with a tFail enabling condition of $V_i \geq 30$ that should not verify to be correct. TIMEOUT indicates a runtime of more than 5 hours.

	Zones			Octagons		
# Caps	Time (s)	States	Correct?	Time (s)	States	Correct?
100	6.504	233	no	269.648	233	no
200	88.599	723	no	4131.728	723	no
300	287.089	875	no	18127.973	875	no
400	710.162	1127	no	65278.836	1127	no
500	3418.39	1967	no	TIMEOUT	-	-

Table 5.6: The total number of event firings for the property with tFail $V_i \geq 30$.

	Event Count ($V_i \geq 30$)	
# Caps	Zones	Octagons
100	232	232
200	722	722
300	874	874
400	1126	1126
500	1924	-

system does not. As is seen in Table 5.7, both the zones and the octagons get the correct verification result. The octagon-based model checker is able to get the same result, since it uses the same reset rates methodology as described for zones in Chapter 4. When finding the error, this time it is the octagons that have the lower state counts and better runtimes. For 8 capacitor stages, the zone-based method completes in more than 8 hours, while the octagons complete is less than a minute. The difference in the runtimes is due to the error lying much sooner on the search path of the octagon algorithm than for zones. Further evidence of this fact is seen in Table 5.8 where the number of event firings is much less for octagons than for zones for 2-5 stages.

5.5 Conclusion

Zones provide efficient methods for finding the state space of LPNs. Indeed, most of the major algorithms have a complexity of $O(n^2)$ or $O(n^3)$. Although zones were originally used to verify systems with continuous variables that have a rate of 1 (that is, timers), zones have been successfully extended to LPNs where the continuous variables have rates other than

Table 5.7: Comparison of zones and octagons for the property shown in Fig. 4.5 that should not verify to be correct.

	Zones			Octagons		
# Caps	Time (s)	States	Correct?	Time (s)	States	Correct?
1	0.146	52	no	0.134	49	no
2	0.534	143	no	0.211	35	no
3	2.00	280	no	0.449	122	no
4	13.6	481	no	0.866	222	no
5	130	877	no	1.427	418	no
6	1047	1649	no	2.413	806	no
7	860	3798	no	5.343	1574	no
8	29709	7489	no	14.811	3116	no

Table 5.8: The total number of event firings for the property with tFail $\neg(V_i \geq 30)$.

	Event Count ($\neg(V_i \geq 30)$)	
# Caps	Zones	Octagons
1	169	165
2	1724	359
3	25597	1091
4	288859	3097
5	2159222	8397

1. Furthermore, Chapter 4 of this dissertation extends zones to LPN with ranges of rates. However, due to the rigidity of zones, states must be added during the state exploration process that are not actually reachable. The primary source of these additional states is the warping process. The necessary over-approximations are greatest when the rates of two variables differ in sign, so that one is positive and the other is negative. With octagons, the negative approximation can be improved so that it is no worse than for positive rates, though over-approximations are not eliminated.

In addition to over-approximations being necessary, time advancement for octagons also requires a degree of over-approximation. The source of this over-approximation is related to the presences of the negative 45° lines. The crux of the matter is that the advancement in three dimensions of one of these negative 45° line segments belongs to a plane of the form $ax + by + cz = d$, where none of the constants a, b, or c is zero. However, the bounding hyperplanes for octagons are of the form $\pm v_i \pm v_j \leq c$ and are not able to capture this plane. Although this over-approximation is not necessary for zones, the over-approximation for octagons is not worse than for zones, since, in the case of zones, the original $45°$ line would be over-approximated by a rectangle, leading to additional states in the timed-advanced zone.

Although the methods of warping and time advancement are new, the basics methods of using octagons have been studied before in the context of software state exploration, such as how to represent octagons as DBMs, how to access the extreme rates, and how to restrict the octagon. Thus, the contribution of this chapter is to add suitable extensions to the method of zones to fill out the necessary algorithms for applying octagons to AMS model checking. With these extensions, it is concretely demonstrated that zones can lead to false negatives that octagons can avoid.

CHAPTER 6

CASE STUDIES

To formally verify an AMS circuit, one needs to have an ability to create a model of the circuit, specify properties, and have a method to check the model against the property. This dissertation improves on the last two of these requirements. With Chapter 3, the ability to express properties in LAMP is increased, while Chapter 4 extends zones to verify models with ranges of rates, and Chapter 5 introduces the use of octagons in the formal verification of AMS circuits. Armed with the ability to verify models with ranges of rates, LEMA's zone-based model checker can now be applied to every model that LEMA's model generator can produce. Previously, the zone-based method was not able to verify every model, since some models are not amenable to translational approach discussed in Section 4.5. The octagon-based model checker is built on the same concepts as the zone-based model checker, and so it is also applicable to models with ranges of rates. In this chapter, the zone and octagon-based verification flows are demonstrated using two case studies.

This chapter demonstrates the new methodology with the help of two circuits: a switched capacitor integrator and a digital C-element whose inputs are driven by RC networks. In the case of the switched capacitor integrator, the original property of avoiding saturation is added by directly creating a simple assertion LPN, and the model that is learned is translated to avoid the ranges of rates which are produced. With the current methodology, the property is written in LAMP and the translational approach becomes unnecessary. In the case of the C-element, the SPICE simulations were provided by a third party, as well as the property that the inputs are ordered. It is shown how to codify this property in LAMP, learn the model, and verify the property against the model using the zone-based and octagon-based model checkers.

The chapter is laid out as follows: Section 6.1 focuses on a switched capacitor integrator. In [74], this circuit is analyzed with the use of LEMA's zone-based model checker and translation of the model. Section 6.1 shows the how the new flow is applied in the case of the switched capacitor integrator. Section 6.2 demonstrates the verification of an AMS circuit built from a digital C-element and some RC networks presented in [34]. Section 6.3 provides the conclusion.

6.1 Switched Capacitor Integrator

A switched capacitor integrator is a particular type of integrator circuit that uses a pair of transistors and a capacitor to implement a type of resistor. A schematic diagram of a switch capacitor is shown in Fig. 6.1 (Fig. 2.1 of [74]). The basic operation of any integrator is to take an input signal, V_{in}, and provide the integral of the signal at V_{out}. A typical application of a switched capacitor is in discrete-time integrators where they are used to accumulate charge. One particular difficulty with switch capacitors is that they can accumulate more charge than desired and end up in the saturation bands of the amplifier. In [74], the switch capacitor integrator of Fig. 6.1 is considered under the environment of an input square-wave signal running at 5 kHz with a low of -1000 mV and a high of 1000 mV. Since the integral of a square wave is a triangle wave, one expects the output of the switched capacitor integrator to be a triangle wave that has a rate of ± 20 mV/μs. To analyze the circuit, two simulations are run with different values for C_2. In the first case, C_2 is given a value of 23 pF and 27 pF, resulting in slew rates of ± 22 mV/μs and ± 18 mV/μs, respectively.

Even though these rates are quite different, under simulation, neither set of conditions lead to saturation, which is seen in the simulation traces of Figs. 6.2 and 6.3. However, as is indicated by [74], an experienced analog designer would know that this circuit has a potential problem for excessive charge build up. To find this potential flaw, the simulation

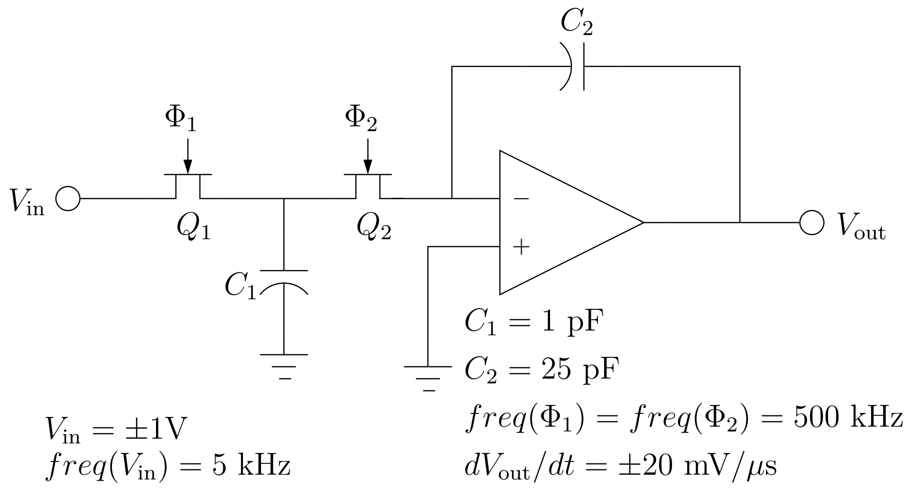


Figure 6.1: A schematic drawing of a switched capacitor integrator.

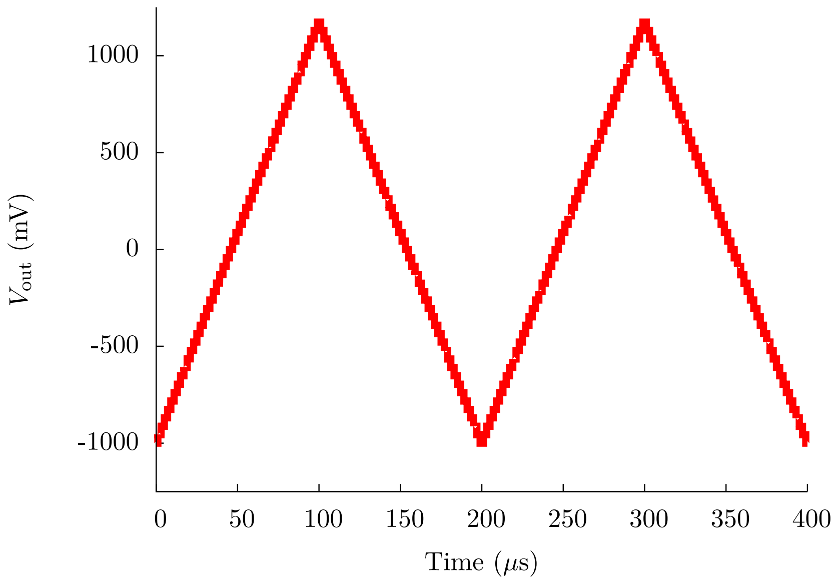


Figure 6.2: SPICE simulation of the switched capacitor integrator with $C_2 = 23$ pF.

traces are run through LEMA's model generation procedure, and the LPN in Fig. 6.4 is produced. As observed in Chapter 4, LEMA's model generation process often produces LPNs with ranges of rates, which is the case with the switched capacitor integrator circuit. The property is added as a single transition that checks that the output voltage does not exceed 2000 mV or -2000 mV. In [74], this property is added by hand; however, it is also possible to encode the property in LAMP, as is shown in Fig. 6.5. The assertUntil(A, B) statement ensures that A remains **true** until B becomes **true**. If B is **false**, this statement requires that A is always **true**, since B cannot become **true**.

To verify this model,[74] relies on the translational approach referred to in Chapter 4, whereby the range of rate assignments are replaced with single rate assignments to the lower bounds and new transitions are added that optionally set the rates to their upper bounds. Using zones on the newly produced LPN does find an error trace where the charge continues to build. However, with the approach of Chapter 4, this additional translation step of the LPN is no longer necessary. LEMA can run on the LPN produced by the model generation process directly. Indeed, when the new version of the zone-based model checker is run on the LPN in Fig. 6.4, the error is also found.

The common method of removing the excessive charge problem is to add a resistor to the feedback loop. A schematic of the new circuit is shown in Fig. 6.6 (Fig. 6.11 of [74]). By running simulation traces (Fig. 6.7) from this new circuit through LEMA's model generator, one obtains the LPN in Fig. 6.8. As before, the approach in [74] is to translate the model

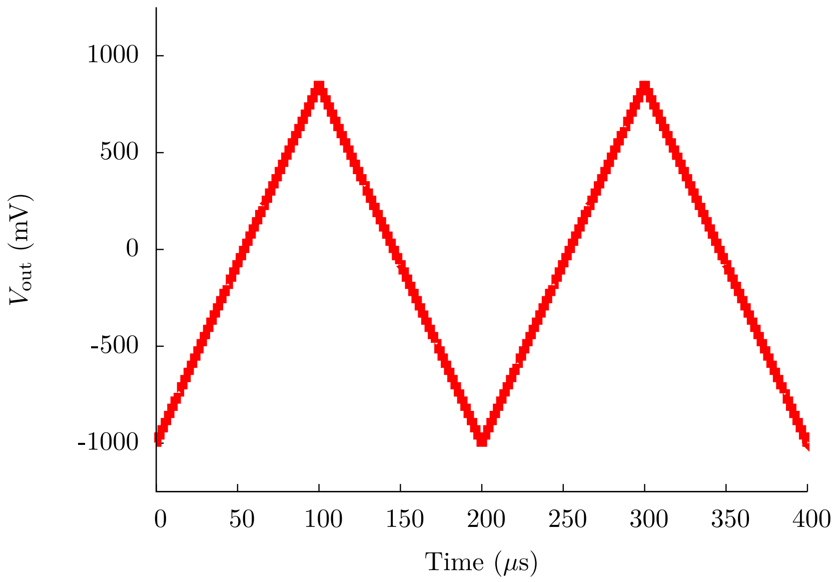


Figure 6.3: SPICE simulation of the switched capacitor integrator with $C_2 = 27$ pF.

to remove any ranges of rates. After running verification, it is found that the model, again, fails verification. Similarly, with the approach of this dissertation, the model of Fig. 6.8 does not pass verification. So, in both cases it is found that the model does not satisfy the property, even with the fix; however, the approach of this dissertation, again, is able to run on the model produced and does not require an additional translation step. The first row in Table 6.1 shows a comparison of the approach in [74] (labeled Translational), the zone-based approach of this dissertation (labeled Algorithmic), and the octagon-based approach of this dissertation (labeled Octagons). All three approaches finish in less than a second. The approaches of this dissertation find the error in one-fifth the number of states.

The error in this LPN model is due to the model not capturing the new circuits behavior accurately enough. To correct this, the thresholds are changed in the model generation process producing the LPN in Fig. 6.9. In this model, more states are added to divide up the continuous state space of the continuous variables. After changing the model to single rate assignments, [74] notes that the model now satisfies the property. Although it is known that, in general, the translation approach can lead to false positives (Section 4.5 and Section 2.6 of [74]), under certain conditions, the translation is sound, which is the case for this particular LPN model. However, with the approach of this dissertation, one does not need to check if the LPN satisfies any conditions to know if the verification results are correct. Running the approach of Chapter 4 on the model in Fig. 6.9 it is verified that the model does satisfy the property; there is no need to confirm that the model satisfies any

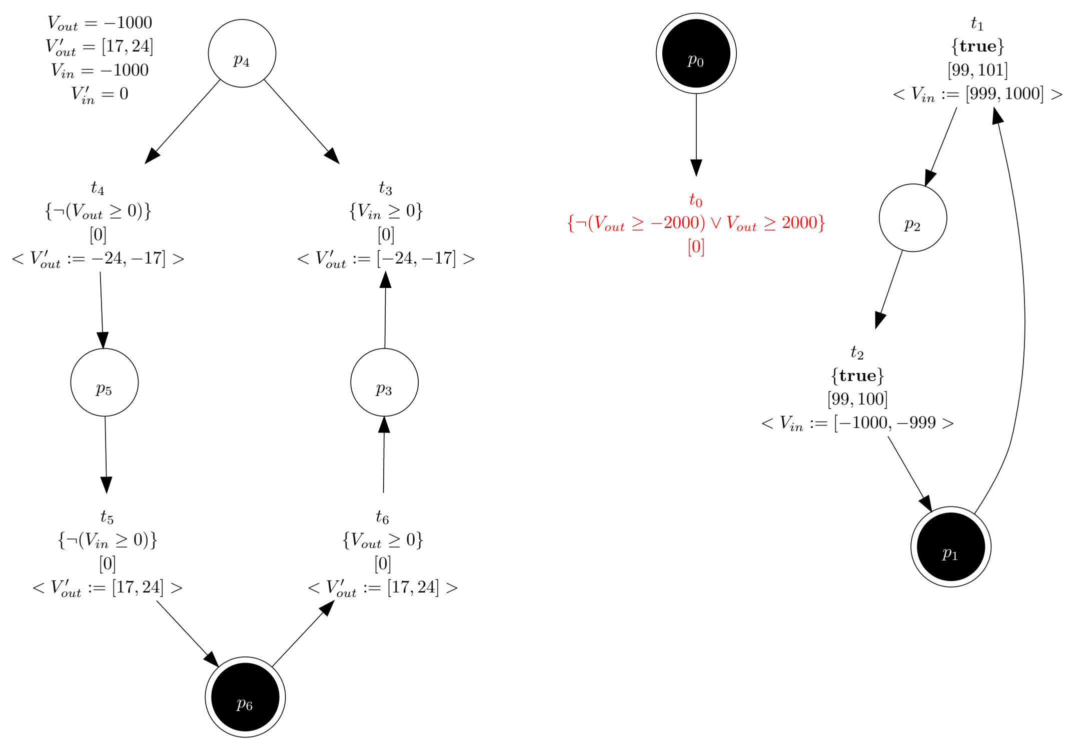

$V_{out} = -1000$
$V'_{out} = [17, 24]$
$V_{in} = -1000$
$V'_{in} = 0$
p_4
p_0
t_1
{**true**}
$[99, 101]$
$< V_{in} := [999, 1000] >$
t_4
$\{\neg(V_{out} \geq 0)\}$
$[0]$
$< V'_{out} := -24, -17] >$
t_3
$\{V_{in} \geq 0\}$
$[0]$
$< V'_{out} := [-24, -17] >$
t_0
$\{\neg(V_{out} \geq -2000) \vee V_{out} \geq 2000\}$
$[0]$
p_2
p_5
p_3
t_2
{**true**}
$[99, 100]$
$< V_{in} := [-1000, -999 >$
t_5
$\{\neg(V_{in} \geq 0)\}$
$[0]$
$< V'_{out} := [17, 24] >$
t_6
$\{V_{out} \geq 0\}$
$[0]$
$< V'_{out} := [17, 24] >$
p_1
p_6

Figure 6.4: Generated LPN model of a switched capacitor integrator.

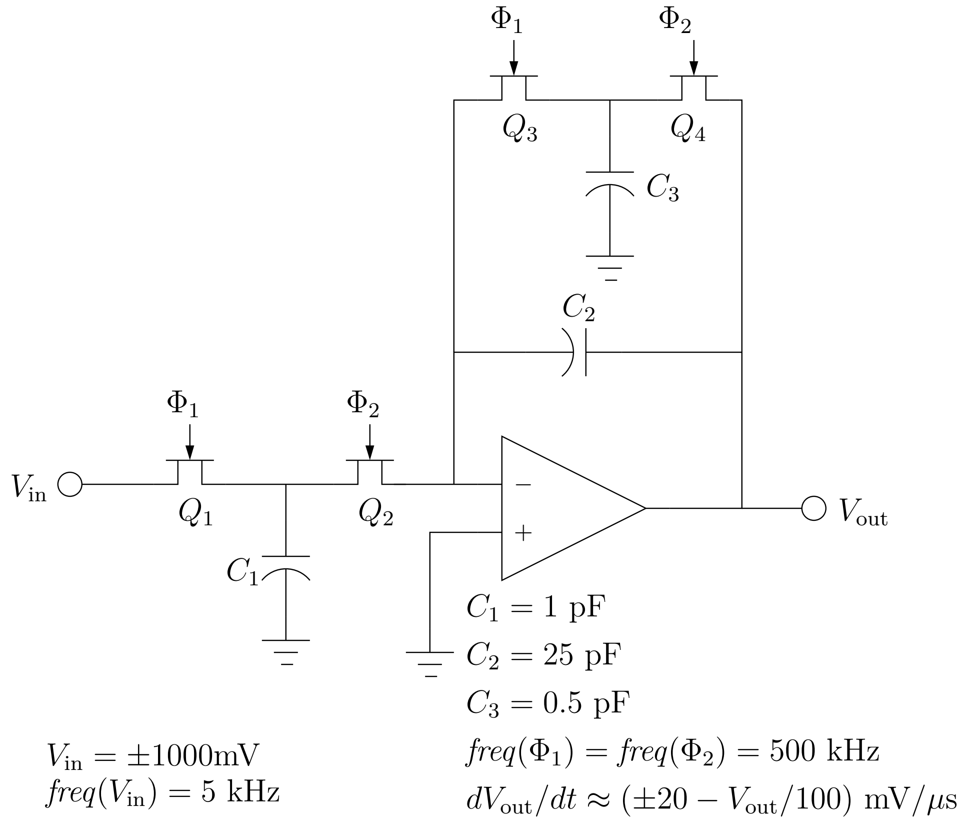
```
property saturation {
  real Vout;
  assertUntil((Vout >= -2000)&(Vout >= 2000), false);
}
```

Figure 6.5: Saturation property for the switch capacitor integrator. The false keyword is not officially supported but can be constructed using a Boolean expression with its negative. For example, ~(Vout >= 0) & (Vout >= 0).

additional properties. Furthermore, as the second line in Table 6.1 shows, the number of required states is cut in half. Just as before, the runtimes of all the algorithms complete is less than a second.

6.2 C-element

For a second case study, a digital C-element whose inputs are driven by RC-networks is studied [34]. Simulation traces for this circuit are provided by Vladimir Dubikhin from Newcastle, along with a property to verify that one of the two input values changes before the second.



Φ_1
Φ_2
Q_3
Q_4
C_3
C_2
Φ_1
Φ_2
V_{in}
Q_1
Q_2
$-$
$+$
V_{out}
C_1
$C_1 = 1$ pF
$C_2 = 25$ pF
$C_3 = 0.5$ pF
$V_{\text{in}} = \pm 1000$mV
$freq(V_{\text{in}}) = 5$ kHz
$freq(\Phi_1) = freq(\Phi_2) = 500$ kHz
$dV_{\text{out}}/dt \approx (\pm 20 - V_{\text{out}}/100)$ mV/μs

Figure 6.6: The switch capacitor integrator of Fig. 6.1 with a feedback resistor. The resistor is added in the form of transistors Q_3 and Q_4 together with capacitor C_3.

A C-element is a type of digital circuit that has the following behavior: when all the inputs are **false**, the output is **false**; when all the inputs are **true**, the output is **true**; and when the inputs are different, the output retains the previous value. In the case of two inputs, A and B, the output, C, can be described by $C = AB + C(A + B)$. The circuit in Fig. 6.10 creates a simple AMS circuit by driving the inputs of a C-element with two RC networks. The RC circuits take as input the inverted output of the C-element and eventually they produce the same output. So, when C is high, the input into the RC networks is low and each RC network starts to fall. Eventually, both A and B become low, changing the output C to low. The inverter then makes the input into the RC networks high. This high value causes the values A and B to become high. The C-element then outputs high and the cycle repeats. A SPICE simulation trace of the circuit is shown in Fig. 6.11.

The speed at which A and B change between high and low depends on what values are chosen for the resistor and capacitor. In particular, the values can be chosen so that A changes faster than B. Using LAMP, this condition can be described as requiring that A goes high before B and A goes low before B. An example of such a property is shown is

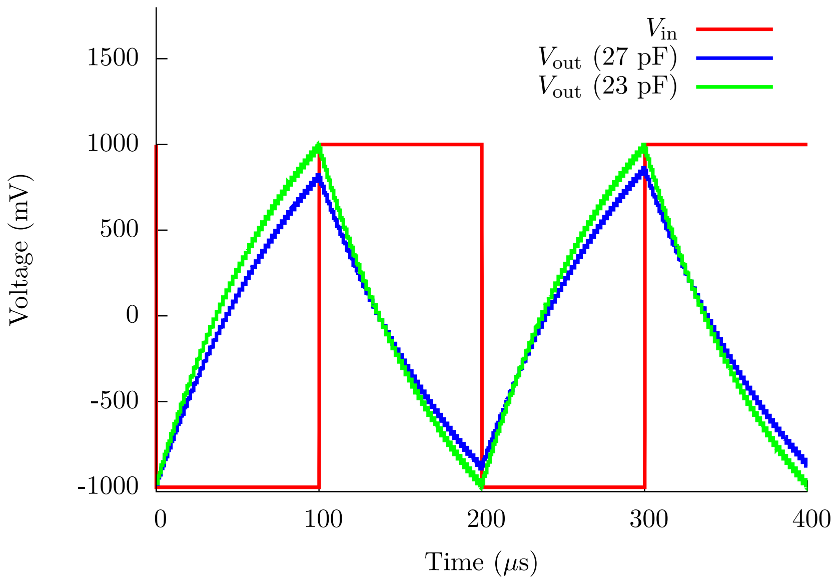


Figure 6.7: SPICE simulations traces for the corrected switched capacitor circuit of Fig. 6.6.

Fig. 6.12. Since the RC circuits are analog circuits, A, B, and C are continuous variables. The high value is considered greater than 5000, and the low value is less than 5000. Thus, the property starts by declaring A, B, and C as real variables. Next, an always loop is added to repeatedly check the property of A changing before B. The **assertUntil** ensures that B cannot go high before A. After A goes high, the property waits for B to go high. Then, the second **assertUntil** ensures that B stays high until A goes low. Finally, after A goes low, the property waits for B to go low and the check repeats. The compiled property LPN for this property is shown in Fig. 6.13.

With the property set, some values are chosen for R and C so that A changes faster than B. The circuit is then simulated using SPICE and the simulation traces are passed through the model generator. The resulting C-element LPN is in Fig. 6.14, the R_1C_1 LPN is in Fig. 6.15, and the R_2C_2 LPN is in Fig. 6.15. To connect the different LPNs together in LEMA a top-level model is created with modules for each of the LPNs, including the model LPNs and the property LPN. A screenshot of the top-level model in LEMA is shown in Fig. 6.16. Each module represents one of the LPN models or the property. Module $C1$ is the C-element LPN depicted in Fig. 6.14. Modules $C2$ and $C4$ are LPNs for the R_1C_1 network in Fig. 6.15. The $C2$ module is the LPN in Fig. 6.15a and sets the rates for charging and discharging the capacitor, while the $C4$ module is the LPN in Fig. 6.15b and set the output A. The modules $C3$ and $C4$ are similar to $C2$ and $C4$, but they handle the R_2C_2 network

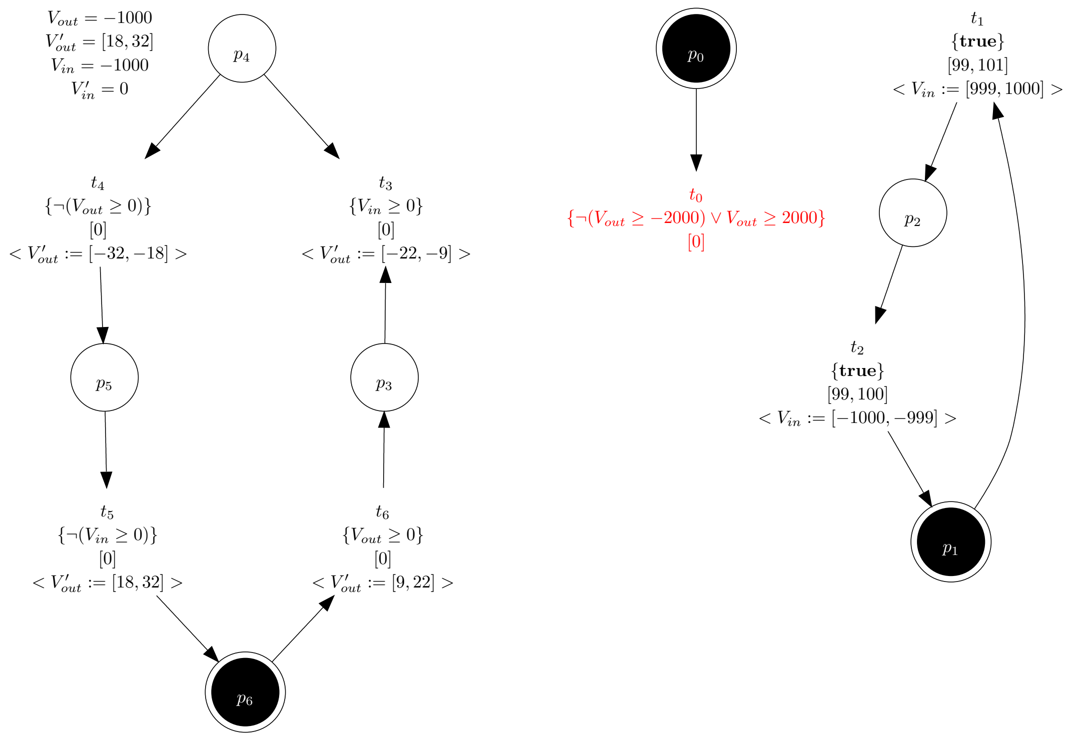


Figure 6.8: Generated LPN model of a switched capacitor integrator with feedback resistor.

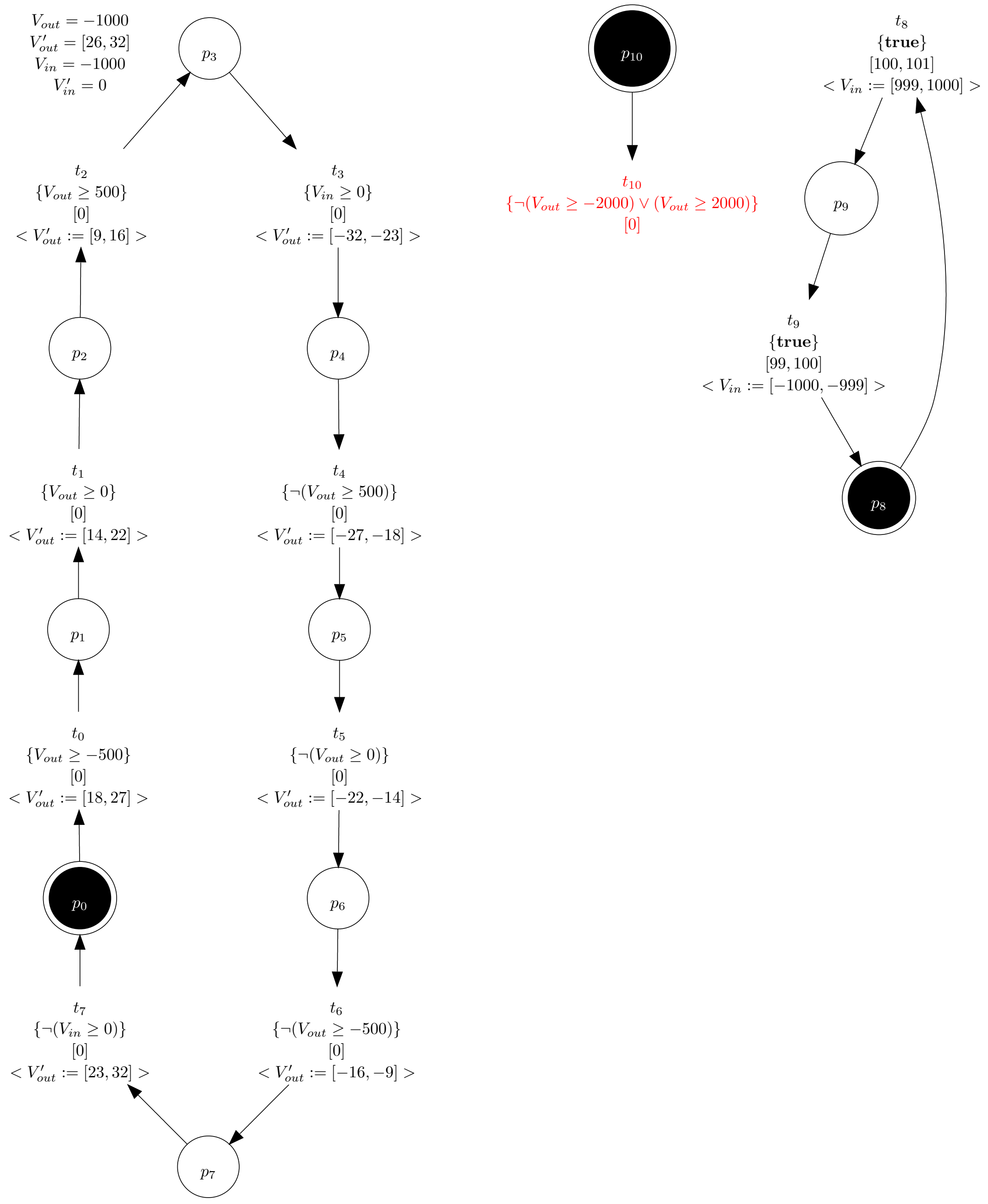
Table 6.1: Comparison of the verification results. for a switched capacitor integrator using the approach of [74], the zone-based approach of Chapter 4, the octagon-based approach of Chapter 5.

	Translational		Algorithmic		Octagons	
Model	Time (s)	States	Time (s)	States	Time (s)	States
Original	< 1	20	< 1	9	< 1	9
Corrected	< 1	73	< 1	44	< 1	42

and correspond to the LPNs in Fig. 6.17. The property LPN of Fig. 6.13 is added as module $C6$. An example simulation for the model portion of the LPN is shown in Fig. 6.18.

After the property LPN in Fig. 6.13 is combined with the LPNs in Figs. 6.14, 6.15, and 6.17, both the zone-based and octagon-based model checkers are run. In both cases, the property is satisfied (first row in Table 6.2). Thus, it is verified that the values of the resistors and capacitor are chosen correctly to ensure that A changes first.

As another check with this property, the values of A and B are reversed. Reversing the signals A and B results in a property that checks for B to change before A. In the top-level model in Fig. 6.16, switching A and B in the property amounts to reversing the input ports



$V_{out} = -1000$
$V'_{out} = [26, 32]$
$V_{in} = -1000$
$V'_{in} = 0$
p_3
t_2 $\{V_{out} \geq 500\}$ $[0]$ $< V'_{out} := [9, 16] >$
t_3 $\{V_{in} \geq 0\}$ $[0]$ $< V'_{out} := [-32, -23] >$
p_2
p_4
t_1 $\{V_{out} \geq 0\}$ $[0]$ $< V'_{out} := [14, 22] >$
t_4 $\{\neg(V_{out} \geq 500)\}$ $[0]$ $< V'_{out} := [-27, -18] >$
p_1
p_5
t_0 $\{V_{out} \geq -500\}$ $[0]$ $< V'_{out} := [18, 27] >$
t_5 $\{\neg(V_{out} \geq 0)\}$ $[0]$ $< V'_{out} := [-22, -14] >$
p_0
p_6
t_7 $\{\neg(V_{in} \geq 0)\}$ $[0]$ $< V'_{out} := [23, 32] >$
t_6 $\{\neg(V_{out} \geq -500)\}$ $[0]$ $< V'_{out} := [-16, -9] >$
p_7
p_{10}
t_{10} $\{\neg(V_{out} \geq -2000) \vee (V_{out} \geq 2000)\}$ $[0]$
t_8 $\{\mathbf{true}\}$ $[100, 101]$ $< V_{in} := [999, 1000] >$
p_9
t_9 $\{\mathbf{true}\}$ $[99, 100]$ $< V_{in} := [-1000, -999] >$
p_8

Figure 6.9: Generated LPN model of a switched capacitor integrator with more thresholds.

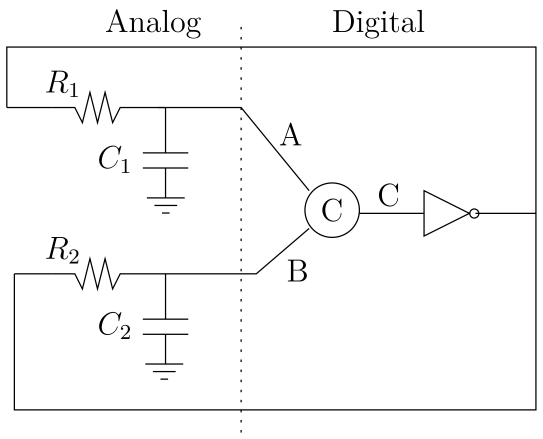


Figure 6.10: Digital C-element with inputs driven by RC circuits.

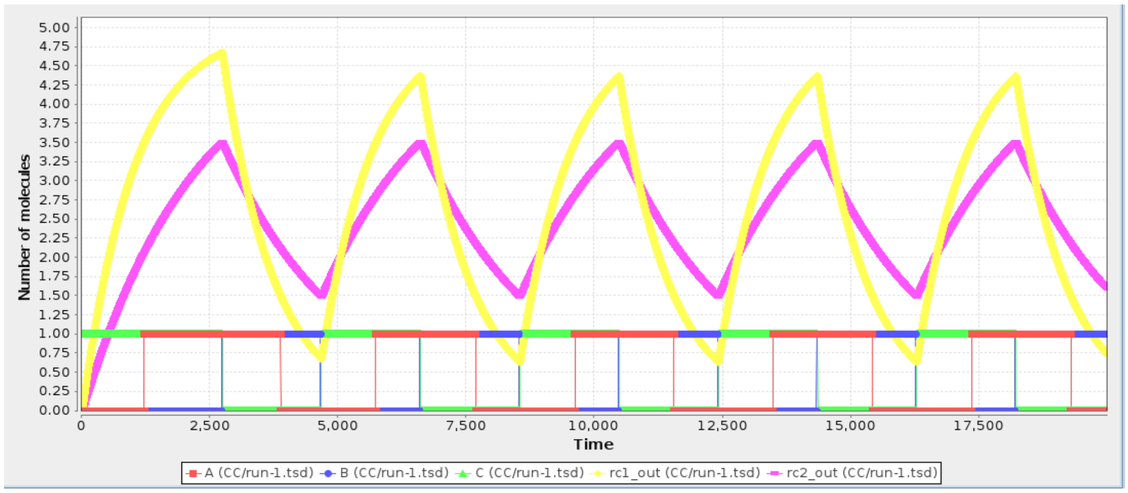


Figure 6.11: SPICE simulation data for Fig. 6.10.

of module $C6$ by assigning the variable A to input B and the variable B to input A. With this change, the zone and octagon-based model checkers both, again, indicate that the C-element circuit fails (second line in Table 6.2). This result provides further the evidence that signal A does, indeed, change before B. A provided error trace for this failure is:

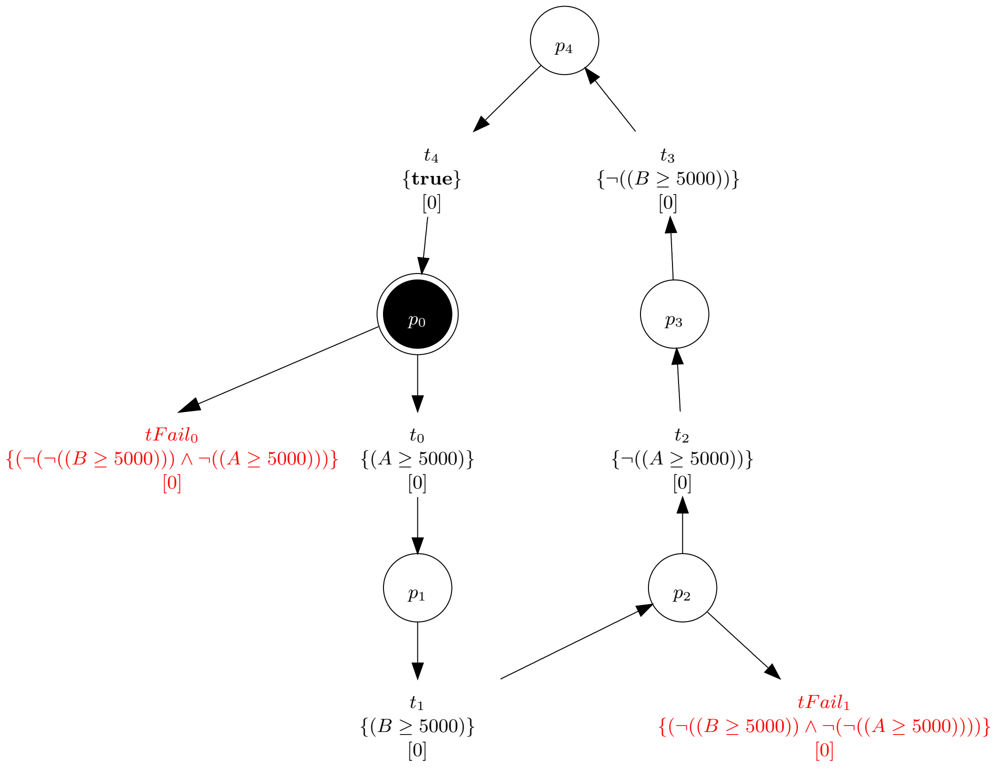
$$s_0 \xrightarrow{C2\_t_8} s_1 \xrightarrow{C4\_t_2} s_2 \xrightarrow{C3\_t_6} s_3 \xrightarrow{C5\_t_2} s_4 \xrightarrow{C1\_t_2} s_5$$
$$\xrightarrow{RC_1 \geq 11683} s_6 \xrightarrow{C2\_t_7} s_7 \xrightarrow{RC_1 \geq 23367} s_8 \xrightarrow{C2\_t_0} s_9 \xrightarrow{RC_2 \geq 17566} s_{10}$$
$$\xrightarrow{C3\_t_0} s_{11} \xrightarrow{C4\_t_1} s_{12} \xrightarrow{C6\_tFail_0}$$

Each transition's label is prefixed with the module it is related to in the top-level model Fig. 6.16. Before following the trace, the initial condition must be known. To determine the initial conditions, one takes the initial value for the variable from the LPN that has

```
property A_faster_B {
  real A;
  real B;
  always {
    assertUntil(~(B>=5000), A>=5000);
    wait(B>=5000);
    assertUntil(B>=5000, ~(A>=5000));
    wait((~B>=5000));
  }
}
```

Figure 6.12: Property for the circuit in Fig. 6.10 requiring that A changes before B.

that variable as an output, that is the LPN that has an assignment to the variable. For example, the initial condition for C is found by looking at the LPN for the C-element (Fig. 6.10), since this LPN is the one that set the value of C. Following this convention, the initial values of the continuous variables are $A = 0$, $B = 0$, $C = 10000$, $RC_1 =$, $RC_2 = 0$, $RC_1' = 32$, and $RC_2' = 16$. The trace starts by firing the t_8 transition in Fig. 6.15a, which is enabled since the initial condition of C is 10000, that is, C is high. Next, the transition t_2 in Fig. 6.15b fires, setting A low. These two transitions setup the initial conditions for the R_1C_1 network. Similarly, the transition t_6 of Fig. 6.17a fires since C is high, followed by transition t_2 in Fig. 6.17b firing, setting B low. These two transitions setup the initial conditions for the R_2C_2 network. Since A and B are initially low, the assignments to these variables are vacuous. The model generator includes them in case of multiple simulation traces that may have different initial conditions. Now, the transition t_2 of Fig. 6.10 fires, setting C low. Next, RC_1 increases enough to cross the 11683 boundary, so the value of the inequality $RC_1 \geq 11683$ changes from **false** to **true**, enabling transition t_7 of Fig. 6.15a. This transition then fires and sets the rate of RC_1 to 32. This assignment is vacuous in this case; however, it is possible for the rate to be changed to 33 by a rate change event since RC_1 is given an initial range of rates $[32, 33]$. After t_7 fires, RC_1 increases enough to cross the 23367 boundary and the inequality $RC_1 \geq 23367$ changes from **false** to **true**, enabling the transition t_0 in Fig. 6.15a and transition t_1 in Fig. 6.15b. The transition t_0 in Fig. 6.15a fires first, since its delay is zero, and sets the rate of RC_1 to 20. The minimum delay on t_1 in Fig. 6.15b is large enough that RC_2 crosses the boundary 17566 before the delay is reached. Thus, the inequality $RC_2 \geq 17566$ changes from **false** to **true**, enabling the transition t_0 in Fig. 6.17a. This t_0 transition has a zero delay and so, it fires immediately and sets the rate of RC_2 to 12. Finally, enough time elapses for the transition t_1 in Fig. 6.15b to fire and



p_4
t_4
$\{\mathbf{true}\}$
$[0]$
t_3
$\{\neg((B \geq 5000))\}$
$[0]$
p_0
p_3
$tFail_0$
$\{(\neg(\neg((B \geq 5000))) \wedge \neg((A \geq 5000)))\}$
$[0]$
t_0
$\{(A \geq 5000)\}$
$[0]$
t_2
$\{\neg((A \geq 5000))\}$
$[0]$
p_1
p_2
t_1
$\{(B \geq 5000)\}$
$[0]$
$tFail_1$
$\{(\neg((B \geq 5000)) \wedge \neg(\neg((A \geq 5000))))\}$
$[0]$

Figure 6.13: The property LPN associated with Fig. 6.12.

the value of A is set to 10000, that is, A is set high. This change of A enables the failure transition $tfail_0$ in Fig. 6.12, which fires immediately and signals the failure. To understand why this failure transition is enabled, recall that the inputs of the property module $C6$ are reversed, effectively switching A and B in the property LPN of Fig. 6.12. Thus, the enabling condition of $tfail_0$ is $\neg(\neg(A \geq 5000)) \wedge \neg(B \geq 5000)$ or $(A \geq 5000) \wedge \neg(B \geq 5000)$. Since A is set high by t_1 and B is still low, that is, $B = 0$, this condition is satisfied. On a higher level, A changes before B, while the property is designed to check that B changes before A, when the senses of A and B are swapped.

6.3 Conclusion

This chapter applies the techniques of this dissertation on two case studies. The first case study is a switched capacitor integrator. In [74], the switched capacitor is analyzed with the aid of LEMA's model generator and zone-based model checker. Since the model generator produces an LPN with ranges of rates, the zone-based model checker does not directly apply. Thus, the LPN model is translated into single rate assignments that first set the rate to the lower bound and then to the upper bound rate once. The verification results are used to detect an error in the circuit design and to verify the corrected circuit. This

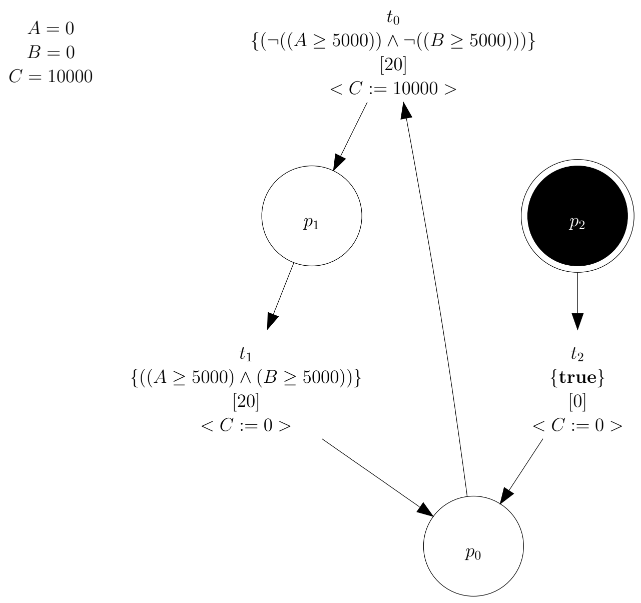


Figure 6.14: An LPN model of a C-element.

approach is sufficient for the switched integrator, but it requires a check to ensure that the translation is sound before accepting any positive verification results. Using LAMP and the extensions to the zone-based model checker, the same conclusions are recovered without the need to translate the model. Furthermore, since the method of this dissertation applies to every model produced by the model generator, there is no need to determine whether the method applies in order to accept a positive verification result. The results of the extended zone-based model checker are also backed up by the new octagon-based model checker.

The second case study is an AMS circuit comprised of a digital C-element whose inputs are driven by RC networks. Simulation traces were provided by Vladimir Dubikhin of Newcastle along with the property that one input signal should change before the other. This property is implemented in LAMP by means of ensuring that A goes high before B and then, that A goes low before B. It is verified that the property holds. The signals are also switched in the property, which corresponds to B going high before A and then, B going low before A. With the new property, the system fails, strengthening the results that A switches first.

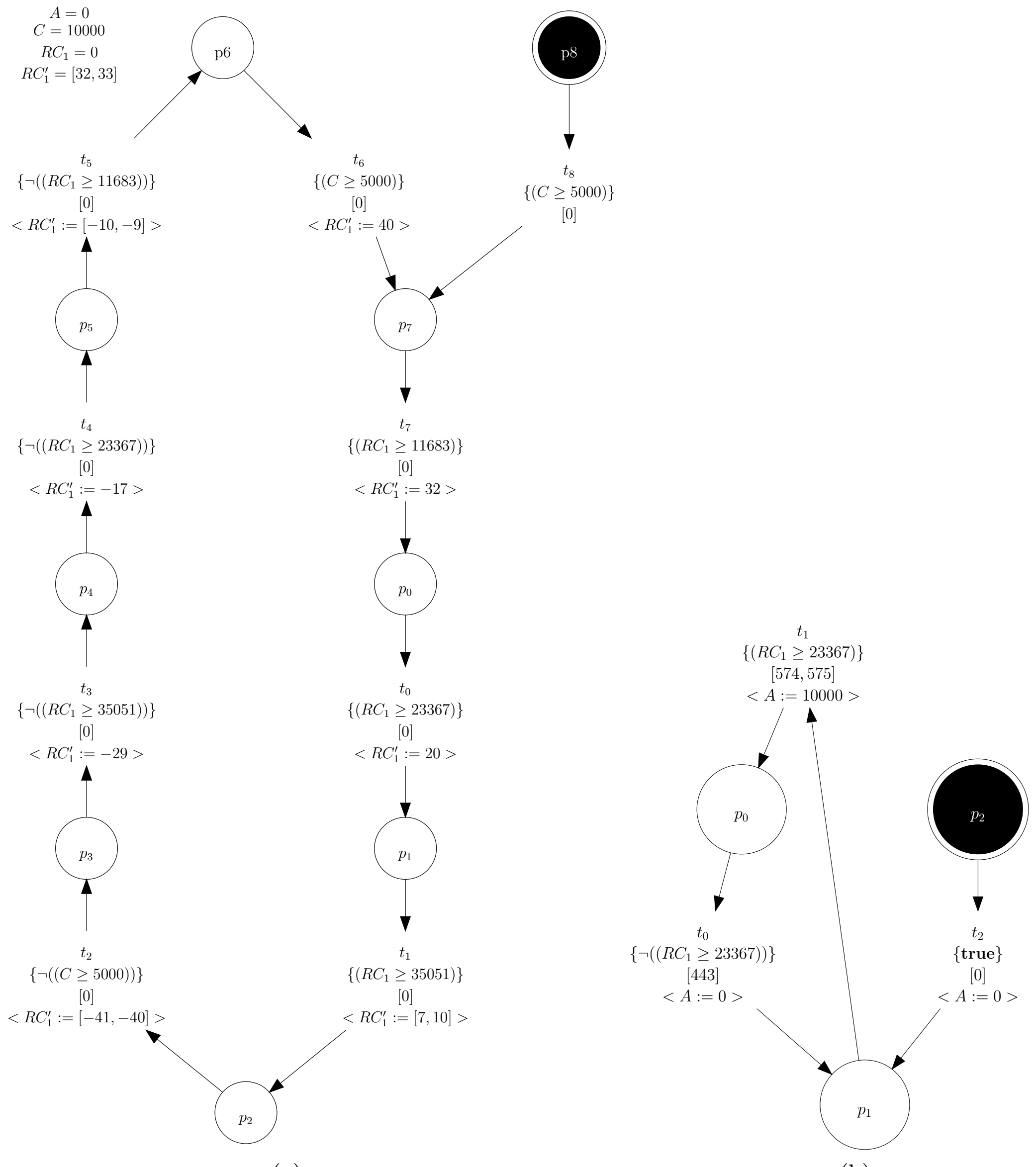
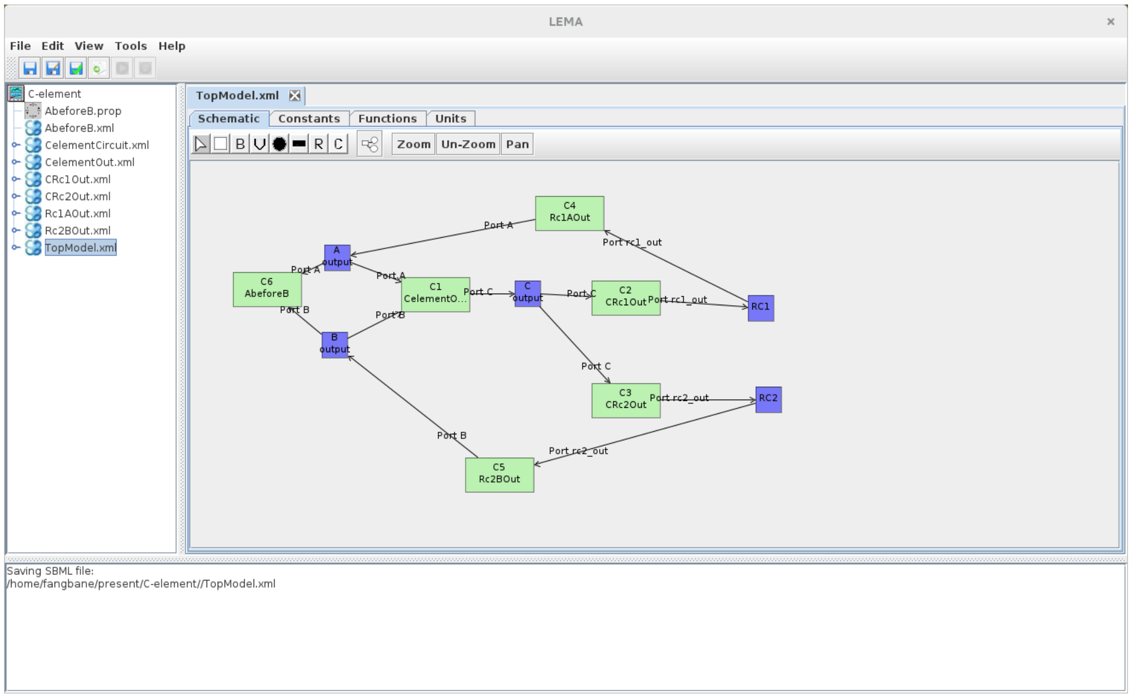
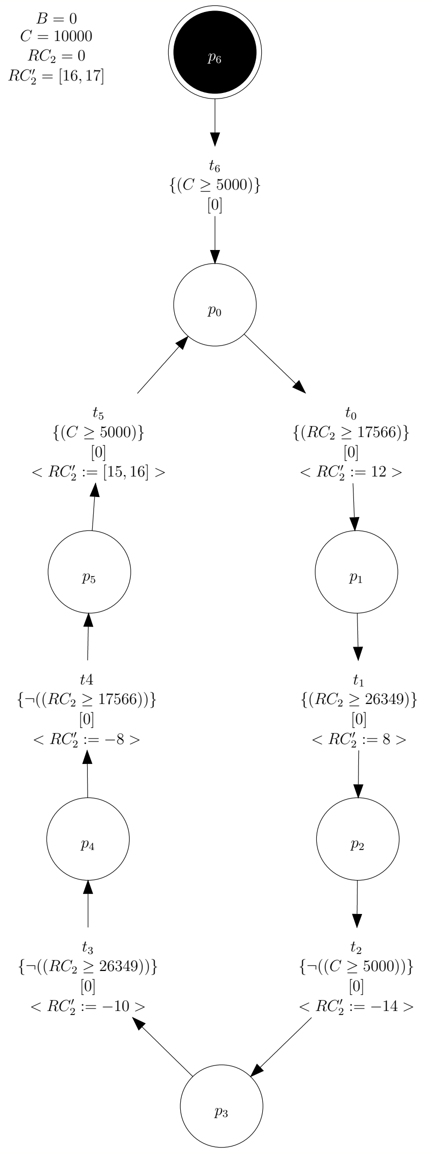


Figure 6.15: An LPN model of the R_1C_1 network.

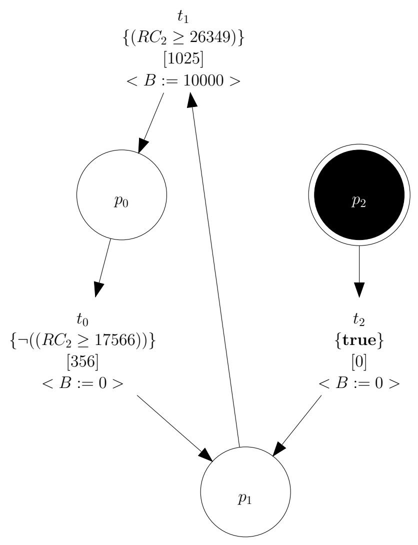


```
Saving SBML file:
/home/fangbane/present/C-element//TopModel.xml
```

Figure 6.16: `LEMA` screenshot of a top-level model connecting the LPNs for each RC-network, the C-element, and the property LPN. The modules are: $C1$ – the C-element LPN (Fig. 6.14), $C2\&C4$ – the R_1C_1 network LPN (Fig. 6.15), $C3\&C5$ – the R_2C_2 network LPN (Fig. 6.17), and $C6$ – the property LPN (Fig. 6.12).



(a)



(b)

Figure 6.17: An LPN model of the R_2C_2 network.

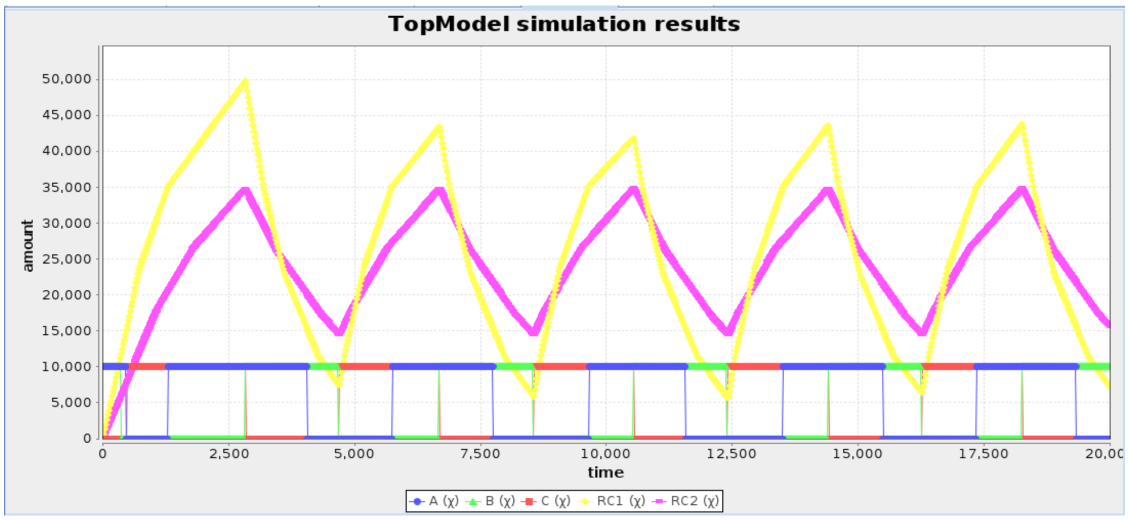


Figure 6.18: Simulation of the C-element models.

Table 6.2: Results for verifying the C-element network.

	Zones			Octagons		
Property	Time (s)	States	Verifies?	Time (s)	States	Verifies?
Fig. 6.12 delayed.	1.9	243	yes	2.6	246	yes
Reversed Fig. 6.12	< 1	13	no	< 1	13	no

CHAPTER 7

CONCLUSION

As devices have scaled, there has been a disproportionate benefit for digital circuits versus analog circuits. The performance of digital designs doubles about every two years, while for analog designs, it take around five to six years for performance to double. The difference can be traced to the different demands of transistors between the two methodologies and the difference that scaling has on these demands. With the digital abstraction, transistors operate in the saturation region and an emphasis is placed on threshold voltages. Scaling a device naturally decreases the threshold voltage, which leads to lower power consumption due to switching. Though the negative effects, such as increased leakage currents and more apparent small channel effects, have started to plague the digital world, the performance gap still remains large.

In contrast to the positive effects of scaling for digital designs, scaling often has negative effects for analog. With analog designs, transistors are often required to operate in the linear region. With smaller sizes, and corresponding smaller voltages, this region decreases, making the allowable ranges smaller. Moreover, as the smaller voltages start approaching the ambient noise levels, it becomes increasingly difficult to distinguish the signal from the noise.

With scaling benefiting digital more than analog, it is tempting to only consider digital designs for small devices. However, most applications requiring small designs also need to interface with the real world, which is inherently analog. So, although a design may use digital circuitry to scale, a portion of the design must be reserved for the analog domain, leading to digitally-intensive AMS circuits.

The combination of digital processing power with analog interfacing helps to scale devices; however, it makes the verification problem more difficult. The common analog verification method of SPICE does not scale well due to the numerous transistors introduced by the digital design, while the various digital methods do not natively support the continuous nature of analog circuitry.

This dissertation improves the verification of digitally-intensive AMS circuits by improving the ability to specify properties and developing sound, efficient extensions for the formal verification of AMS designs.

7.1 Summary

This dissertation improves the verification of AMS circuits by generalizing the LAMP property language, extending the method of zones to systems with ranges of rates, improving the accuracy of zones by introducing octagons to the verification domain, and applying these methods to some case studies.

In order to perform any verification task, one needs a way to specify what is the correct behavior for the circuit. In order to define this specification in a way that is amenable to formal methods, one usually uses a type of formal specification language. These languages are often either based on assertion languages or inspired by temporal logics. An example of the former is RT-SVA, which is inspired by SVA, while STL is an example of the latter. Though powerful, these languages tend to be difficult to use and to convince analog designers to learn. They also tend to be difficult to translate into the property LPNs needed by LEMA for model checking. In contrast, LAMP is a simple intuitive language that is easy to translate; however, it is still a nascent language and cannot specify every property of interest. By adding a delay statement and a conditional always block, one is able to ignore transient periods without requiring a check and enable a check to be aborted if the environment changes.

Properties alone are not very useful; one also needs an ability to check whether an AMS model satisfies the property. Zones provide efficient methods for verifying systems that have discrete events and constant rate continuous variables. Though less accurate than other methods available, zones have algorithms whose complexity are no more than $O(n^3)$, where n is the number of nonzero rate continuous variables and active transitions. Although, through model translation these methods have been adapted to continuous variables with ranges of rates, they do not handle cases where the continuous variables are sampled more than once. One method to soundly extend zones to such cases is to change a range of rates into a set of rate events, where the rate is initially set to the lower bound, is allowed to be set once to the upper bound rate, and is then reset to the lower bound rate after any event. On the face of it, such a method seems to maintain the complexity of a nondeterministic switching point. However, zone-based reachability algorithms can be adapted to handle these rate changes by simply adding a rate change event and using warping. The zone representation itself applies the rate change to every point in the space, effectively capturing all the switching points. This situation is analogous to a zone's ability to collect together states into equivalence classes.

With the extension of zones to models with ranges of rates, the efficient methods of zones are able to verify a large class of models, including all types of models produced by LEMA's model generator. However, the efficiency comes at the cost of over-approximating the actual reachable state space. In particular, by only allowing positive 45° constraints ($x-y \leq c$), zones are often unable to do better than rectangles for representing the reachable state space after a continuous variable has changed from a positive to a negative rate or vice versa. By adding in -45° constraints ($-x-y \leq c$ and $x+y \leq c$), the over-approximations required for switching between positive and negative rates is no worse than the normal over-approximations required for warping. Additionally, octagons have a similar DBM representation that leads to the associated algorithms, again, having a complexity of no more than $O(n^3)$. By increasing the accuracy of the state space representation, one correspondingly removes some potential false negatives.

After adding to the ability to specify properties, extending the zone-based reachability algorithms to properly handle ranges of rates, and increasing the accuracy via octagons, the last step is to apply the methods to more real-world examples. The first example is a switched capacitor integrator. The integrator has been studied before with LEMA's zone-based model checker and model generator. However, since the model generator produces models with ranges of rates, it is necessary to translate the model into single rate assignments. Although the previous translational approach was adequate for the integrator, it was not a general approach for any model that could be generated. With the method of this dissertation, the translational step is unnecessary, and the method applies in general. Furthermore, all verification results obtained by the previous approach are reproduced with the approach presented in this dissertation with at least an order of two reduction in state count. As an additional example, an AMS circuit is analyzed that centers around a digital C-element driven by analog RC networks. It is successfully verified that the changes in the inputs are ordered.

7.2 Future Work

Creating and verifying properties for AMS circuits is a difficult problem that currently has no industrial solution. The benefits are clear: no need to perform tedious checks directly on wave forms, reduction of human error, more automated processes, more assurance of correct behavior, etc. However, no methodology has yet reached a critical mass of usability and interest for it to become more than a sideline interest for companies. Partly, this situation is due to a difficulty in convincing analog designers to learn a new paradigm of design. On the other hand, the lack of serious interest is due to none of the methodologies being mature enough to move into the production setting. Hence, there are many opportunities for future work.

7.2.1 Extensions To LAMP

LAMP has only been in development for a few years and, even with the current extensions, still has places to improve before it becomes a full featured language that is able to specify all properties of interest. LAMP provides a procedural semantics that is different than most temporal logics, but it is currently unclear to what extent such semantics can reproduce the property checks available in temporal logics. Similarly, it should be determined exactly what subset of properties LAMP can verify. With a more thorough evaluation of the properties LAMP can and cannot describe, one can better gauge where the language fits in terms of expressiveness. This knowledge, in turn, will help determine where improvements can be made. Of particular interest is the comparison of LAMP with PSL, RT-SVA, and STL.

As another direction, the language can be extended further. A simple extension is to add support for different units. Currently, the properties have to be added with the appropriate scaling determined by hand. In particular, the scaling has to ensure that the values are integers and match with the scaling introduced by the model generation process [17, 66, 73, 74]. By adding units, context can be provided allowing the tool to determine what the needed scaling should be.

In addition to adding support for units, it is also useful to have the ability to specify parameters. With parameters, one can create properties that can be easily modified by changing a few numbers. For example, if a model is going to be tested for several different parameter values, the property could be defined in terms of these parameters, making it easier to adapt the property to the given version of the model.

7.2.2 Improving Range of Rate Efficiency

Zones provide an efficient means of performing state space reachability analysis; however, with the support of ranges of rates, the runtimes can still be prohibitive. The situation is exacerbated the more continuous variables are allowed to have a range of rates. In particular, the current algorithm requires a full interleaving of all the currently possible rate change events with all the other rate change events, as well as any other currently enabled events, which leads to a potential exponential blowup relative to the number of continuous variables that currently have a range of rates.

Chapter 4 introduces a modest attempt at improving the runtimes by biasing the search towards rate change events. In this way, the rate events only interleave with each other and not with all currently enabled events. This optimization still has an exponential blowup in terms of the continuous variables with ranges of rates, but should reduce the exploration overhead. One detail that remains to resolve, however, is whether the optimization is sound or not. The full algorithm is sound (Chapter 4), but it has not been proven that the reduction is as well.

Even with the rate optimization method of biasing toward rate events, the runtime is still substantial. This fact leaves open the need for even more aggressive optimization strategies. Potential first steps are to consider quicker ways of generating the zones resulting from the rate event interleavings.

7.2.3 Improving the Octagon Representation

Although the octagon representation is able to utilize a DBM representation, there is a lot of redundancy. Every inequality involving two different base variables, that is, any inequality $V_i^{\pm} - V_j^{\pm} \leq c$ such that $i \neq j$, always has two representations in the DBM. Consider again, the standard representation for two variables V_0 and V_1 shown below:

$$D = \begin{array}{c} \\ V_0^+ \\ V_0^- \\ V_1^+ \\ V_1^- \end{array} \begin{array}{c} \begin{array}{cccc} V_0^+ & V_0^- & V_1^+ & V_1^- \end{array} \\ \begin{pmatrix} 0 & -2m_{V_0} & b_1 & -b_4 \\ 2M_{V_0} & 0 & b_3 & -b_2 \\ -b_2 & -b_4 & 0 & -2m_{V_1} \\ b_3 & b_1 & 2M_{V_1} & 0 \end{pmatrix} \end{array}.$$

Every entry involving b_i in the lower left corner is redundant. Thus, one loses no information by eliminating these entries:

$$D = \begin{array}{c} \\ V_0^+ \\ V_0^- \\ V_1^+ \\ V_1^- \end{array} \begin{array}{c} \begin{array}{cccc} V_0^+ & V_0^- & V_1^+ & V_1^- \end{array} \\ \begin{pmatrix} 0 & -2m_{V_0} & b_1 & -b_4 \\ 2M_{V_0} & 0 & b_3 & -b_2 \\ & & 0 & -2m_{V_1} \\ & & 2M_{V_1} & 0 \end{pmatrix} \end{array}.$$

Thus, the DBM is nearly an upper triangular matrix. So, it is likely that a sparse matrix representation will be helpful. A good representation that eliminates these redundant equations will make coherency checks unnecessary.

7.2.4 Counter-Example Guided Abstraction Refinement

A popular method for eliminating false negatives is to use *counter-example guided abstraction-refinement* (CEGAR). The basic idea is to first run verification and get an error trace. Next, determine whether the error trace is real, that is, the original system does exhibit the behavior, or if it an erroneous trace, that is, the system does not really exhibit the behavior. If the trace is not really possible, then the trace is used to inform a refinement routine so that the model no longer exhibits the erroneous behavior. The loop is continued until the system is verified as being correct or a real error trace is found. With a CEGAR loop, if the original model does not capture the desired behavior accurately enough, then the model can be fixed until it does.

In order to implement a CEGAR loop for LEMA, two additional capabilities have to be added. The first is to develop a method for determining whether a trace is a false trace or a real trace. Determining whether a trace is real or not has some challenges. The first is that the state space explored by LEMA is via equivalence classes of states. Thus, in order to produce a traditional trace, one has to start with a violating trace and back-trace it to a set of initial conditions. Since certain steps of the algorithm involve adding additional states (like warping), it may not be possible to finish such a back trace. Presumably, such a case would mean that the trace is not a real trace for the current model. This process is further complicated by the fact that different events could lead to the same state space or a subset, thus, the back trace will require exploring various different possibilities much like the original state space exploration.

Alternatively, one can consider verifying traces using an interval SPICE simulator as suggested by [74]. This avenue still needs to be explored in the context of LEMA. Such a direction would alleviate the need to find a particular trace, though it still needs to be explored how the initial conditions would be identified.

Once a false trace has been identified, one needs a way of adjusting the model. LEMA's model generation algorithm has a few parameters that can be adjusted, such as the number of thresholds that can be adjusted, as well as the scaling of time and the continuous variables can be adjusted. Changing the thresholds leads to tighter bounds on the ranges of possible rates, while changing the scaling increases the accuracy of the zones by reducing rounding errors and tightening the warping bounds. However, more methods should be identified for increasing the faithfulness of the model to the real circuit. The best case would be to identify a method that has the potential of at least asymptotically approaching the actual behavior of the circuit.

7.2.5 Real World Case Studies

Since the methods of this dissertation are aimed at the verification of AMS circuits, it is useful to investigate their use on real case studies. Real case studies serve several purposes for the evaluation of any method of verification. They provide insight into how the methods are utilized in a real setting, while providing designers with real examples on how to apply the methods. This helps to motivate interest, as well as helps provide concrete uses to understand the methods better. They also show that the methods are practical and are not just of theoretical interest.

One particular circuit of interest is the PLL. A PLL is used in several applications including clock recovery, correcting clock skew, frequency synthesis, and wireless communication. They have been the focus of several attempts of formal verification with some success; however, their still does not exist an accepted general method of verifying these circuits. Hence, it would be of great interest to see how far the current methods can reach in solving this problem.

APPENDIX A

TRACES AND RANGES OF RATES

This appendix demonstrates how the zone-based exploration of ranges of rates can be reduced to considering rate change events that only involve the lower bound, upper bound, and zero rates. The focus of this appendix is to show how the same result is true at the level of traces, that is, for the purpose of verification, it is enough to consider traces that only use the extremal rates and rate zero.

A.1 Modeling with Ranges of Rates

Section A.2 motivates using only the lower and upper bound rates together with zero by showing that a piecewise linear function which uses only the lower, upper, and zero rates can be used to provide the same outcome between a given pair of events. This section also presents a more detailed discussion on why only allowing a single rate change is not enough when using LPNs and automata for models. Section A.3 provides a set of traces that do have enough flexibility in the number of times their rates can switch to give a representative of each equivalence class in $\mathbb{T}/\sim_T$ and states the two main theorems of this appendix. Section A.4 provides the proof of the theorems in Section A.3. Section A.5 shows that the method of allowing multiple resets provides the correct results for the model used in this appendix and Section A.6 provides a discussion on why the methods work.

A.2 Linear Approximations of Ranges of Rates

To simplify the presentation, this appendix focuses on LPNs whose transitions have zero delay. The model and property LPNs used in this appendix are shown in Fig. A.1. The model LPN is a simplified model of one of the capacitor stages shown in Fig. 4.2. After the transition t_7 fires, the range of rates for v is $[1, 2]$ and the rate of t is $[1, 1]$. A myriad of trajectories for the continuous variables t and v exist that reach the point $(t, v) = (10, 15)$ allowing the pair of transitions t_0 and t_1 to fire sequentially. One such trajectory is shown as the middle heavy line in Fig. A.2a. This trajectory is given by the trace:

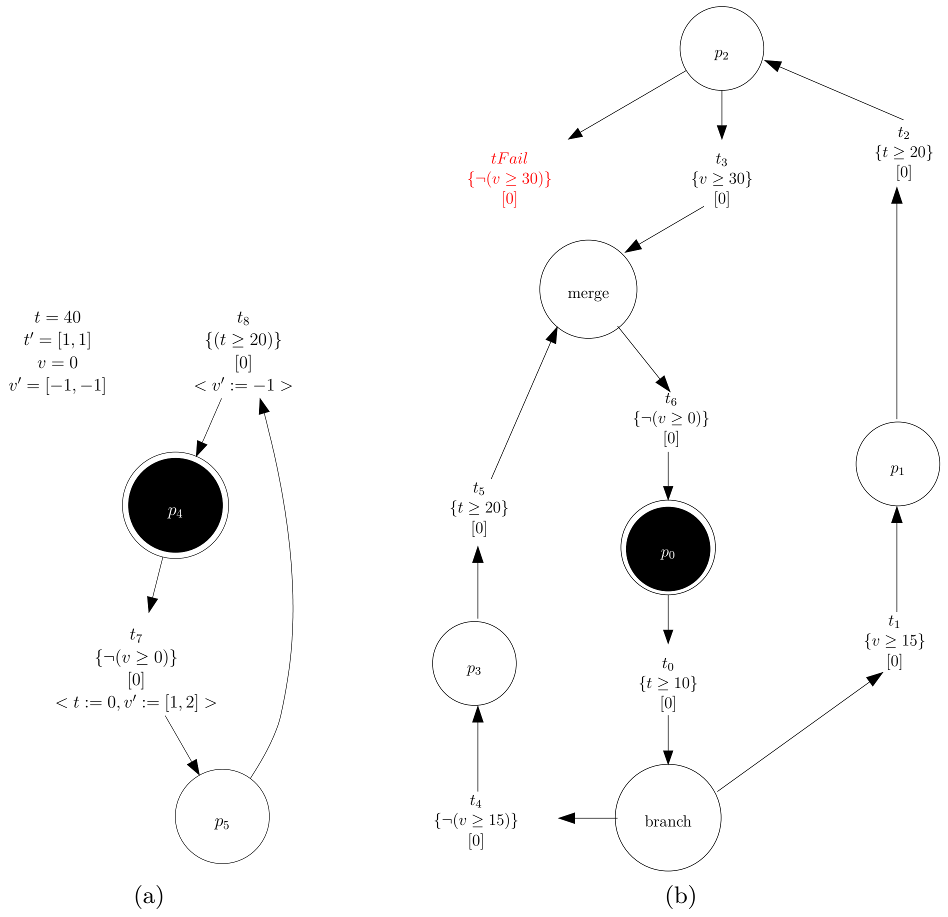


Figure A.1: Example LPN illustrating that resetting a rate once per assignment is not enough to capture all behavior. This model is a simplified version of one of the capacitor stages shown in Fig. 4.2. (a) A model circuit controlling the charging and discharging of a capacitor. (b) A property for the capacitor control circuit that checks that the charge is either below or exceeds 15 mv at 10 μs; if the charge exceeds 15 mv, then the charge must exceed 30 mv after 20 μs.

$$T_H = \sigma_0 \xrightarrow{t_7} \sigma_1 \xrightarrow{R(v)\leftarrow 1.5} \sigma_2 \xrightarrow{4} \sigma_3 \xrightarrow{R(v)\leftarrow 1} \sigma_4 \xrightarrow{2} \sigma_5 \xrightarrow{R(v)\leftarrow 2} \sigma_6 \xrightarrow{3} \sigma_7 \xrightarrow{R(v)\leftarrow 1} \sigma_8 \xrightarrow{1,\{t\geq 10, v\geq 15\}} \sigma_9 \xrightarrow{t_0} \sigma_{10} \xrightarrow{t_1} \sigma_{11}.$$

For this example, the sequence of events $(E_j)_{j\geq i}$ is the sequence of statements above the arrows. Specifically, $E_0 = t_7$, $E_1 = (R(v) \leftarrow 1.5)$ and so on. Then $sub_t(T_H) = \{E_0, E_9, E_{10}\} = \{t_7, t_0, t_1\}$. This same sequence of transitions can also fire by firing the transition t_7 (which sets $v' = 1$), allowing time to flow for 5 time units, setting $v' = 2$, and allowing time to flow for 5 more time units, that is:

$$T_L = \sigma_0 \xrightarrow{t_7} \sigma_1 \xrightarrow{5} \psi_0 \xrightarrow{R(v)\leftarrow 2} \psi_2 \xrightarrow{5,\{t\geq 10, v\geq 15\}} \sigma_9 \xrightarrow{t_0} \sigma_{10} \xrightarrow{t_1} \sigma_{11}.$$

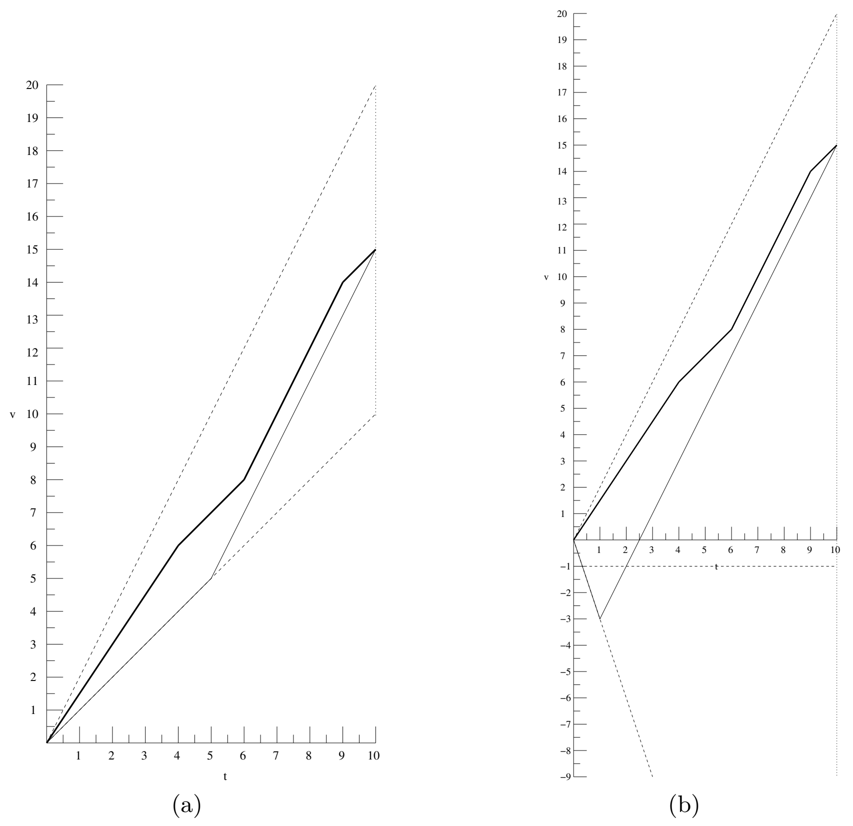


Figure A.2: Piecewise approximation of a ranges of rates. (a) Piecewise approximation of the range of rates [1,2]. The heavy line in the middle is the derived function f_{T_H} and the thinner line is the derived function f_{T_L}. (b) Piecewise approximation of the range of rates $[-3, 2]$ illustrating that an approximating trajectory that uses -3 and then 2 can lead to crossing an addition inequality $v \geq -1$ not crossed by the original trace.

Again, $sub_t(T_L) = \{t_7, t_0, t_1\}$. Since the same sequence of transitions fire (t_7, t_0, and t_1), $T_H \sim_T T_L$ even though trace T_H is much more complicated between the transitions. Thus, it is enough to consider only T_L. This situation is not unique to this pair of traces. Any trajectory for the variable v must terminate at some point $(10, y)$ for $10 \leq y \leq 20$ since the rate is bounded between 1 and 2. Moreover, a function can always be found that starts with rate 1, switches to rate 2 at some point τ', and has this same endpoint $(10, y)$. The following theorem and corollary ensure that such a function can always be found for any trajectory and any range of rates.

Theorem 3. *Let $a, b \in \mathbb{R}$ with $0 \leq a \leq b$ or $a \leq b \leq 0$, $\tau \in \mathbb{R}$ any nonnegative real number, and $q \in \mathbb{R}$ any real number. Then, for any real number v such that $a\tau + q \leq v \leq b\tau + q$, there exists a $\tau' \in [0, \tau]$ such that $f(\tau) = v$ where f is defined by*

$$f(x) = \begin{cases} b(x-\tau') + a\tau' + q & \text{if } \tau' \le x \le \tau \\ ax + q & \text{if } 0 \le x \le \tau' \end{cases}.$$

The point τ' is called the switching point *for the function f. Furthermore, the roles of a and b can be reversed in the function so that f has a slope of b on $[0,\tau']$ and a on $[\tau',\tau]$.*

Proof. The theorem can be reduced to the case when $q = 0$ by subtracting q from the problem and then adding it back at the end. So, all that needs to be proved is the case when $q = 0$. First, if $a = b$, there is nothing really to prove since $v = a\tau = b\tau$ and $f(x) = ax = bx$. So, τ' can be taken to be any value in $[0,\tau]$ and $f(\tau) = a\tau = b\tau = v$. So assume that $a < b$. Then $\tau' = \frac{b\tau - v}{b-a}$ is the value needed. To verify this, first note that by assumption, $a\tau \le v \le b\tau$, from which it follows that $\frac{-a\tau}{b-a} \ge \frac{-v}{b-a} \ge \frac{-b\tau}{b-a}$. After adding $\frac{b\tau}{b-a}$ throughout and simplifying, one obtains $\tau \ge \tau' \ge 0$. Furthermore, with this choice of τ':

$$\begin{aligned} f(\tau) &= b(\tau - \tau') + a\tau' \\ &= b\tau - (b-a)\tau' \\ &= b\tau - (b\tau - v) = v \end{aligned}$$

as required. The proof that the roles of a and b can be reversed is similar. □

For Fig. A.2a, the values are $a = 1$, $b = 2$, $\tau = 10$, $\tau' = 5$, $v = 15$ and $q = 0$. This theorem is not stated in full generality. It, in fact, works with just assuming that $a, b \in \mathbb{R}$ and v is between $a\tau$ and $b\tau$. In particular, it does not really matter whether or not 0 is between a and b. The reason it is stated with excluding zero is for a secondary consideration as to how this theorem is used later in the theory. The main point is that if both a and b are nonnegative or both are nonpositive, then the range of the function f is contained in the range of the function $\ell(t) : [0,\tau] \to \mathbb{R}$ given by $\ell(t) = (1-t)q + tv$, the line connecting the starting value q with the ending value v. This is important since it guarantees that if the original trajectory (which is continuous) does not cross an inequality, then this approximating trajectory does not cross an inequality.

If zero is strictly between a and b, then it is no longer possible to ensure that the approximating trajectory does not introduce new behaviors by crossing additional inequalities. This is illustrated in Fig. A.2b. In this figure, the variable v is now allowed to have a range of rates of $[-3, 2]$. The heavy-line trajectory from Fig. A.2a is still a valid trajectory and is repeated on Fig. A.2b. Also, similar to Fig. A.2a, an approximate trajectory can be created that starts off using -3 for 1 time unit and then switches to 2 for 9 time units. Then the resulting trajectory again hits $(10, 15)$. Now suppose there is an inequality $v \ge -1$ (shown as the horizontal dashed line in Fig. A.2b). In this case, the original trajectory (the heavy line) does not change the value of $v \geq -1$ while the approximating trace (the thin line) changes the truth value of the inequality twice. This could lead to different transitions firing if the inequality $v \geq -1$ enables another transition. This problems can be avoided by first setting the rate to the highest positive rate and then setting the rate to zero. In this case, one would set the rate to 2 for 7.5 time units and then to 0 for 2.5. It is just as possible to set the rate to zero first and then to the appropriate nonzero rate, except when the value of the continuous variable v is equal to the right hand side of the inequality $v \geq a$. In this case, setting the rate to zero first and then to a negative rate changes the value of the inequality $v \geq a$ from **true** to **false**, whereas setting the rate directly to a negative rate keeps the value of the inequality **false**. If a transition depends on the truth value of $v \geq a$, then it is possible the method of setting the rate to zero first will not be sufficient. Thus, when zero is a possible rate, the method adopted in this appendix is to set the rate to the maximum rate or the minimum rate, then set the rate to zero. This notion is formalized in the following corollary to Theorem 3.

Corollary 1. *Let $a, b \in \mathbb{R}$ with $a \leq 0 \leq b$, $\tau \in \mathbb{R}$ a nonnegative number, and $q \in \mathbb{R}$ any real number. Then, for any real number v such that $a\tau + q \leq v \leq b\tau + q$, there exists a $\tau' \in [0, \tau]$ such that $f(\tau) = v$ where f is defined by*

$$f(x) = \begin{cases} [\frac{a-b}{2}(1 - \frac{v}{|v|}) + b]\tau' + q & \text{if } \tau' \leq x \leq \tau \\ \frac{a-b}{2}(1 - \frac{v}{|v|}) + b]x + q & \text{if } 0 \leq x \leq \tau' \end{cases}$$

provided $v \neq 0$ and $f(x) \equiv 0$ otherwise. The expression $\frac{a-b}{2}(1 - \frac{v}{|v|}) + b$ is just a shorthand way of writing b when $v > 0$ and a when $v < 0$.

Proof. The case when $v = 0$ is trivial. The rest of the theorem follows by applying the second part of Theorem 3 with $a = 0$, and $b = b$ when $v > 0$ and applying the first part of Theorem 3 with $a = a$, and $b = 0$ when $v < 0$. □

With Theorem 3 and Corollary 1, it is possible to approximate a sequence of rate changes and time advancements with a trajectory that uses only the lower and upper bound rates together with rate zero. The key lies in how many times the trace is allowed to reset the rates. The situation where the trace is only allowed to be set to the lower bound and then be set to the upper bound once per assignment to the variable's rate is explored in Section 2.6 of [74]. It is shown in [74], using an LPN similar to Fig. A.1, that this is not enough to capture all the important behavior; that is, there is a trace in the original model that fires a failure transition, whereas no traces fire the failure transition when only resetting the rate once. In the terminology of trace equivalence, this means there is a trace in the original model that is not trace equivalent to any trace in the resetting once situation.

As an example, consider the following trace for the LPN in Fig. A.1. First, fire the transition t_7 and assign v a range of rates $[1, 2]$. Change the rate of v to be 1.5, then advance time 10 time units. The value of (t, v) is then $(10, 15)$ so transitions t_0 and t_1 fire. Set the rate to 1.3 and advance time another 10 units. Then the value of (t, v) is $(20, 28)$ so transitions t_2 and $tFail$ fire, signaling a failure. Symbolically, the trace is:

$$\begin{aligned} T_F = \sigma_0 &\xrightarrow{t_7} \sigma_1 \xrightarrow{R(v)\leftarrow 1.5} \sigma_2 \xrightarrow{10,\mathfrak{I}_1} \sigma_3 \xrightarrow{t_0} \sigma_4 \\ &\xrightarrow{t_1} \sigma_5 \xrightarrow{R(v)\leftarrow 1.3} \sigma_6 \xrightarrow{10,\mathfrak{I}_2} \sigma_7 \xrightarrow{t_2} \sigma_8 \xrightarrow{tFail} \textit{failure}, \end{aligned}$$

where $\mathfrak{I}_1 = \{t \geq 10, v \geq 15\}$ and $\mathfrak{I}_2 = \{t \geq 20\}$. The trajectory for t and v is shown in Fig. A.3a. No trace that is obtained by doing one rate change per rate assignment can fire the failure transition $tFail$. To see why, consider a trace starting in the initial state. Since the transition t_7 is enabled, it must fire. So, t_7 fires and assigns the range of rates $[1, 2]$ to v, as well as sets the current rate of v to 1. In order to be able to fire $tFail$, the transitions t_0 and t_1 must fire, so v must be above 15 after 10 time units (when $t \geq 10$). If the rate of v is not changed before 10 time units, then after 10 time units $v = 10$ and the transitions t_0 and t_4 fire. After 10 more time units, (t, v) are $(20, 35)$, so t_5 and t_8 fire. Eventually, t_6 and t_7 fire, returning the LPN to the state already considered. Thus, in order for t_1 to fire, the rate of v must be set to 2 before t_0 fires. Once t_1 fires, $v \geq 15$ and the rate of v must remain 2, since the rate is only allowed to changed once per rate assignment. So after 10 more time units, $t = 20$ and $v \geq 35$, forcing transitions t_2 and t_3 to fire (see Fig. A.3b). Eventually, t_6 and t_7 fire, which returns the LPN back to the state following t_7 firing in the initial state. Thus, the failure transition does not fire.

Similar to the LPN translational approach of [74], the authors of [25] provide a translation from LHAs to *stopwatch automata* (SWAs). The basic idea for the translation of a range of rates $[a, b]$ for a continuous variable v is to replace the place (call it p_0) with the range of rate assignment with 3 stages. The first stage determines how much time the system stays in p_0. Call this time τ. The second stage determines the value of the continuous variable v after τ time units if v has a rate of a, that is, it calculates $x_0 + a\tau$ where x_0 is the initial value of v in p_0. Finally the third stage runs the variable v through the possible values v could have for all the rates $[a, b]$, that is, v goes through the values $[x_0 + a\tau, x_0 + b\tau]$. This process can be illustrated using Fig. A.2a. Again let v be the continuous variable and suppose that the range of rates is $[1, 2]$. The first stage allows time

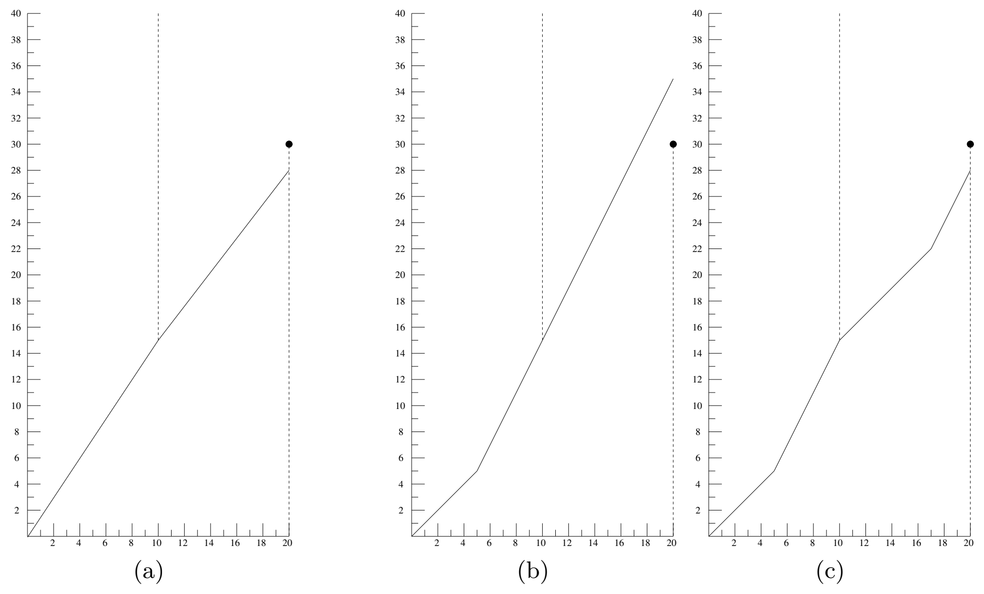


Figure A.3: Example trajectories for the continuous variables t and v in Fig. A.1 starting from when transition t_7 fires. In each diagram, the first dashed line from left to right is the relevant part of the enabling conditions for t_0 and t_1 (combined) in Fig. A.1. The second dashed line in the relevant portions of the enabling conditions for t_2 and $tFail$ (combined). (a) A trajectory where the rate is first set to 1.5 and then set to 1.3 after 10 time units. (b) A trajectory where the rate starts with 1 and then switches to 2 once at 5 time units. (c) A trajectory for a reset trace that is inequality equivalent to the trace giving the trajectory in (a).

to go for 10 time units. Then the second stage uses the rate of 1 and ends with v having the value of 10 at 10 time units. Finally, the third stage considers the possible values of v while v travels along the vertical line from 10 units to 20, that is, the points $(10, v)$ for $10 \leq v \leq 20$.

Fig. A.4 gives an example of Fig. A.1a translated using this idea. In this figure, places p_{5_1}, p_{5_2}, and p_{5_3} handle the first, second, and third stage, respectively. Since a direct application of the method in [25] is not possible due to the differences in the formulation, Fig. A.4 is used to simulate the spirit of the [25] approach. Three main differences should be noted. The first and least of the three is the absence of an extra transition from p_{5_2} to p_{5_2} and similarly from p_{5_3} to p_{5_3}. These transitions are only needed if the lower bound rate is greater than 1 and if the difference between the upper bound and lower bound rates is greater than 1, respectively. So as an optimization, these transitions may be omitted (as they are in Fig. 2 of [25]). The second difference is the use of a delay on transition t_{7_2}. In a direct translation of [25], the time variable would be allowed to be any value from 0 to ∞. In the present circumstance, any value other than 20 is not valid for this particular model

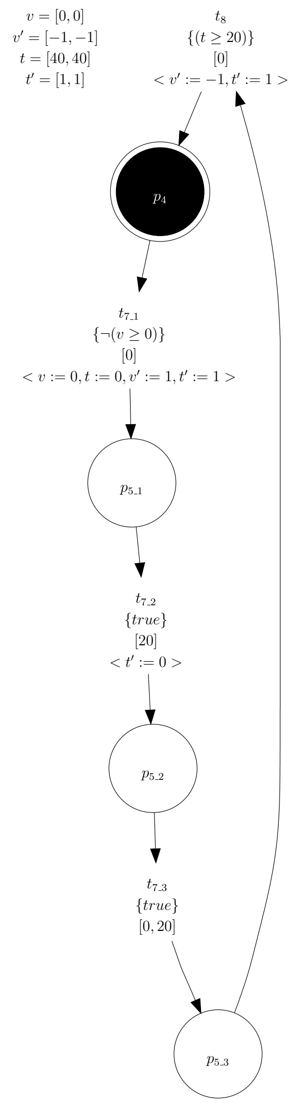


Figure A.4: Translation of Fig. A.1a inspired by [25].

because of the enabling condition of $t \geq 20$ on transition t_8 in Fig. A.1a. In particular, if t is chosen to be a value less than 20, then the model deadlocks since transition t_{7_3} never is enabled. On the other hand, if t is allowed to be a value more than 20, then the new model has a trace that switches the rate of v to -1 after more than 20 time units, which is not possible in the model of Fig. A.1a. Thus, the delay of 20 has been added to remove such cases. The third main difference is the use of a delay in transition t_{7_3}. A direct translation would use an enabling condition on t_{7_2} that allows the transition to fire at any time less than or equal to the value of t in the given state. Since in the current situation, the upper bound on the time for t is known, this same idea can be accomplished by using a delay that allows the transition t_{7_3} to fire any time between 0 and 20 time units. These remarks, of course, do not represent a general method for the adaptation of [25] to LPNs since they rely heavily on the nature of the current model; however, they do provide a reasonable adaptation for comparison purposes.

After obtaining the model in Fig. A.4 and combining it with the LPN from Fig. A.1b, one again finds that the failure transition $tFail$ does not fire. The key observation why this does not work is that the model of Fig. A.4 only allows for v to be equal to 10 when t is 10 and thus t_1 does not fire. This transition can be enabled if t_{7_1} is allowed to have a delay of 10 instead of being forced to 20 since v would then be allowed to consider all possible values from 10 to 20 while t is 10. However, as noted above, the model deadlocks. The deadlock could be avoided by allowing t to progress, but this does not match the spirit of translation in [25], which requires all progress in time to occur in the initial stage p_{5_0}. In any case, even if all variables are allowed to return to the values they had before entering the first stage, then negating the enabling conditions on $tFail$, again, misses the failure. In fact, nothing short of adding an entire second set of stages ensures that the failure transition is not missed no matter how it is altered.

Focusing too much on examples and what is or is not tweaked may belie the true nature of why the approaches of [74] and [25] do not solve the current problem. The crux of the matter (as mentioned in [74]) deals with how many times the variable with a range of rates is sampled. If the variable is sampled as a single time per rate assignment, then Theorem 3 together with Corollary 1 ensure that the method of [74] works. A similar assertion can be made for the method of [25]. The situation is quite different when the variable is sampled twice (or more). With two samplings, the first sampling can force a rate change in the case of [74] or force a particular pass through the calculation stages in [25]. In both cases, the variable is no longer able to use the full range of rates to reach the necessary point for the second sampling, and thus, a second reset needs to be allowed. Section A.3 provides a sufficient number of times for the resetting in order to ensure that no behavior is missed. It should be noted that although [25] does not appear to solve the current problem, this is not an indication that the method is flawed. It simply means that the contexts are more different than they superficially appear.

A.3 Transition Equivalent Traces

This section introduces a larger class of traces that still uses only the lower and upper bound rates together with rate zero; however, they allow for resetting the rates and having another rate change every time an inequality changes or transition fires. Call a trace T a *reset trace* if every time an inequality or transition event occurs, the rate for each continuous variable is reset via the `resetRates` function and is allowed to have another appropriate rate change event. The formal definition is as follows:

Definition 1. *If E_i is a transition event or inequality event and E_j is a transition event or inequality event, call E_i and E_j* time insensitive successors *provided $i < j$ and any event E_k such that $i < k < j$ is either a rate change event or a time advancement that is not an inequality event. Then, a trace $T = \sigma_0 \xrightarrow{E_0}, \sigma_1 \xrightarrow{E_1} \dots$ is a* reset trace*, denoted T_r, if the following condition is satisfied: for any pair of time insensitive events E_i and E_j, there is at most one rate change event E_k such that $i < k < j$ for each continuous variable v. Furthermore, the rate change events E_i must be one of the following types:*

- *$R(v) \leftarrow \max(RR(v))$ if $0 \notin RR(v)$ or $\max(RR(v)) = 0$.*
- *$R(v) \leftarrow 0$ if $0 \in RR(v)$ and either 0 is not one of the bounds or $\min(RR(v)) = 0$.*

The main theorem to prove is:

Theorem 4. *For each trace equivalence class $[T]_T \in \mathbb{T}/\sim_T$, there exists a reset trace T_r such that $[T_r]_T = [T]_T$.*

This theorem implies that for every sequence of transitions that are possible to fire, there is a trace using only the lower and upper bound rates together with rate zero that fires the same sequence of transitions. So, it is enough to just consider the reset traces.

To prove Theorem 4, it turns out to be easier to prove a theorem for a finer set of equivalence classes than transition equivalence. Two traces $T = \sigma_0 \xrightarrow{E_0} \sigma_1 \xrightarrow{E_1} \dots$ and $T' = \sigma'_0 \xrightarrow{E'_0} \sigma'_1 \xrightarrow{E'_1} \dots$ are called *inequality equivalent*, denoted $T \sim_I T'$, if the following conditions are satisfied for $sub_I(T) = (E_{i_k})$ and $sub_I(T') = (E'_{j_k})$ where $sub_I(T)$ is the subsequence of inequality events and transition events:

- E_{i_k} is a transition event if and only if E'_{j_k} is a transition event.
- E_{i_k} is an inequality event if and only if E'_{j_k} is an inequality event.
- If E_{i_k} is an inequality event then $ineq(E_{i_k}) = ineq(E'_{j_k})$ where $ineq(E_i)$ is the set of inequalities that change truth value as a consequence of the time advancement E_{i_k}.

With this definition in place, the theorem to prove becomes:

Theorem 5. *For each inequality trace equivalence class $[T]_I \in \mathbb{T}/\sim_I$, there exists a reset trace T_r such that $[T_r]_I = [T]_I$.*

Notice that $T \sim_I T' \Rightarrow T \sim_T T'$. Thus, Theorem 4 follows directly from Theorem 5. The proof of Theorem 5 is the topic of Section A.4.

A.4 Proof

The proof of Theorem 5 follows the outline of first extracting a function to follow the trajectory of the continuous variables for a trace fragment consisting of only time advancements, then constructing an approximating function, and finally deriving a new trace that is inequality equivalent to the original. The first two steps are accomplished by the next two lemmas.

All the first lemma aims to do is formalize extracting a function from the trajectory of the continuous variables in a trace. For example, the heavy middle line and the thinner line in Fig. A.2a are the derived functions for the traces T_H and T_L described in Section A.1. Similarly, the function shown in Fig. A.3a is the derived function f_{T_F} for the trace T_F also described in Section A.1.

Lemma 1. *Let $k \geq 1$ be an integer and*

$$\hat{T} = \sigma_0^0 \xrightarrow{E_0^0} \sigma_1^0 \xrightarrow{E_1^0} \cdots \xrightarrow{E_{l_0}^0} \sigma_{l_0}^0 \xrightarrow{\tau_1} \tag{A.2}$$

$$\sigma_0^1 \xrightarrow{E_0^1} \sigma_1^1 \xrightarrow{E_1^1} \cdots \xrightarrow{E_{l_1}^1} \sigma_{l_1}^1 \xrightarrow{\tau_2} \tag{A.3}$$

$$\vdots \tag{A.4}$$

$$\sigma_0^{k-1} \xrightarrow{E_0^{k-1}} \sigma_1^{k-1} \xrightarrow{E_1^{k-1}} \cdots \xrightarrow{E_{l_{k-1}}^{k-1}} \sigma_{l_{k-1}}^{k-1} \xrightarrow{\tau_k, \Im} \sigma_0^k \tag{A.5}$$

be a trace fragment where l_i is a natural number for $0 \leq i \leq k-1$ and each E_j^i is a rate change event for $0 \leq i \leq k-1$ and $0 \leq j \leq l_i$. Also, let $V_1, \ldots, V_N$ be an ordering of the continuous variables V, and let $Q_j^i(V_n)$ be the value of V_n in state σ_j^i where $0 \leq n \leq N$, $0 \leq i \leq k-1$, and $0 \leq j \leq l_i$. Then, there exists a function

$$f = (f^1, f^2, \ldots, f^N) : [0, \tau] \to \mathbb{R}^N$$

where N is the number of continuous variables and $\tau = \sum_{i=1}^{k} \tau_i$ that has the following properties for each coordinate function f^i:

1. *$f^n(0) = Q_i^0(V_n)$ for all $0 \leq n \leq N$ and $0 \leq i \leq l_0$,*
2. *$f^n(\sum_{j=1}^{i} \tau_j) = Q_j^i(V_n)$ for all $0 \leq n \leq N$, $1 \leq i \leq k-1$, and $0 \leq j \leq l_i$,*
3. *$f^n(\tau) = Q_0^k(V_n)$ for $0 \leq n \leq N$,*

4. $f^n(t)$ is piecewise linear for each $0 \leq n \leq N$.

The function f is called the derived function for $\hat{T}$ and is denoted $f_{\hat{T}}$.

Proof. Identify the tuple $(a_1, a_2, \ldots, a_n) \in \mathbb{R}^N$ with vectors. For each i such that $1 \leq i \leq k$, define

$$f(t + \sum_{j=1}^{i} \tau_j) = (1-t)(Q^{i-1}_{l_{i-1}}(V_0), \ldots, Q^{i-1}_{l_{i-1}}(V_N)) + t(Q^i_0(V_0), \ldots, Q^i_0(V_N))$$

for $0 \leq t < 1$ and define

$$f(\tau) = (Q^k_0(V_0), \ldots, Q^k_0(V_N)).$$

This function is the linearization between the points

$$(Q^{i-1}_{l_{i-1}}(V_0), \ldots, Q^{i-1}_{l_{i-1}}(V_N))$$

and

$$(Q^i(V_0), \ldots, Q^i_0(V_N)),$$

which are the values of the continuous variables for each of the previous states and successor states involved in the time-advancement events. Then this function satisfies the necessary conditions. One key fact to note, to make this work, is that the rate change events do not change the value of Q so $Q^i_0 = Q^i_j$ for all $0 \leq i \leq k-1$ and $0 \leq j \leq l_i$. □

For the next lemma, it is shown that given a trace fragment that is only rate changes and time advancements, then a reset trace fragment can be created that is inequality equivalent to the given trace fragment. One example of this process is again the traces T_H and T_L of Section A.2. Lemma 2 ensures that, given a trace like T_H (minus the transition events) that has only time advancements, rate changes and one inequality event, then a trace like T_L (minus the transition events) always exists such that $T_H \sim_I T_L$ (see Fig. A.2a).

Lemma 2. *Let $\hat{T} = \sigma_i \xrightarrow{E_i} \sigma_{i+1} \xrightarrow{E_{i+1}} \sigma_{i+2} \ldots \sigma_n \xrightarrow{E_n} \sigma_{n+1} \xrightarrow{\hat{\tau},\mathfrak{I}} \sigma_{n+2}$ be a trace fragment such that for each $0 \leq i \leq n$ the event E_i is a rate change or time advancement that is not an inequality event. Then there exists a reset trace fragment $\hat{T}_r$ such that $\hat{T}_r \sim_I \hat{T}$.*

Proof. Since $\hat{T}$ is of the form in Lemma 1, the derived function $f_{\hat{T}} : [0, \tau] \to \mathbb{R}^N$ exists where τ is the sum of the time advancements in $\hat{T}$. Denote the component functions of $f_{\hat{T}}$ by $f^i_{\hat{T}}$. Note that since the variable v_i has a range of rates $[a_i, b_i]$, $f^i_{\hat{T}}$ has the same range of rates $[a_i, b_i]$ and thus $a_i\tau \leq f^i_{\hat{T}}(\tau) \leq \tau b_i$. So either Theorem 3 or Corollary 1 applies depending on whether $0 \in [a_i, b_i]$. Using Theorem 3 when $0 \notin [a_i, b_i]$ and Corollary 1 when

$0 \in [a_i, b_i]$, there exists a piecewise function f^i with switching point $(\tau')^i \in [0, \tau]$ such that $f^i(0) = f^i_{\hat{T}}(0)$ and $f^i(\tau) = f^i_{\hat{T}}(\tau)$ for each i. A trace fragment $\hat{T}_r$ is constructed such that $f_{\hat{T}_r} = (f^1, f^2, \ldots, f^N)$. Let $\tau'_0, \tau'_1, \ldots, \tau'_n$ be the distinct switching points in increasing order ignoring any $(\tau')^i = \tau$ and define $\tau_0, \tau_1, \ldots, \tau_n$ such that $\tau_0 = \tau'_0$ and for all $0 < i \leq n$, $\tau_i = \tau'_i - \tau'_{i-1}$.

The switching point marks a change of rate on the variable v_i. This change is from the lower bound rate to the upper bound rate if $0 \notin [a_i, b_i]$ and is to 0 if $0 \in [a_i, b_i]$. Thus, at time $(\tau')^i$ the reset trace fragment should have an event $\xrightarrow{R(v_i)\leftarrow r}$ where r is the upper bound rate or 0, respectively. For a fixed i let $E^0_i, E^1_i, \ldots, E^{k_i}_i$ be the rate changes that need to occur at τ'_i, ordered according to the coordinate that the rate change affects. A reset trace fragment can then be constructed as:

$$\begin{aligned}\hat{T}_r := \sigma'_0 \quad &\xrightarrow{\tau_0} \quad \oplus_{j=0}^{k_0}(\sigma'_{1+j} \xrightarrow{E^j_0})\sigma'_{k_0+2}\ldots\sigma'_{\kappa} \\ &\xrightarrow{\tau_n} \quad \oplus_{j=0}^{k_n}(\sigma'_{\kappa+j+1} \xrightarrow{E^j_i})\sigma'_{\kappa+k_n+2} \xrightarrow{\tau-\hat{\tau},\Im} \sigma'_{\kappa+k_n+3} = \sigma_{n+1}\end{aligned}$$

where $\oplus$ is the obvious concatenation of events and states, $\kappa = (\sum_{i=0}^{n-1} k_i) + 2 * n$, σ'_0 is the same as state σ_0 except the rates of the continuous variables. In the case that the range of rates for the continuous variable v_i does not contain zero, then the rate is set to the minimum rate. When 0 is one of the rate bounds, the rate is set to the nonzero rate bound (if zero is not the only rate). Finally, if $[a_i, b_i]$ is the range of rates for v_i and $a_i < 0 < b_i$, then a couple cases need to be consider depending on whether $f^i_{\hat{T}}(\tau)$ is greater than, equal to, or less than $f^i_{\hat{T}}(0)$. If $f^i_{\hat{T}}(\tau) > f^i_{\hat{T}}(0)$, then the rate is set to b. Similarly, if $f^i_{\hat{T}}(\tau) < f^i_{\hat{T}}(0)$, then the rate is set to a. If $f^i_{\hat{T}}(\tau) = f^i_{\hat{T}}(0)$, then the rate is a, provided there exists an inequality $v_i \geq f^i_{\hat{T}}(0)$ and the inequality is **false** in σ_0. Otherwise, the rate of v_i is set to b. Note, in the case that $f^i_{\hat{T}}(\tau) = f^i_{\hat{T}}(0)$, the rate is originally set to a or b to give the correct truth value for any inequality $v_i \geq f^i_{\hat{T}}(0)$. The rate is then immediately set to 0. Finally, $\hat{\tau}$ is the sum of the time advancements τ_i and the rest of the σ'_i are the states resulting from the events. The trace $\hat{T}_r$ is then a reset such that $\hat{T}_r \sim_I \hat{T}$. □

The main idea for the proof of Theorem 5 is illustrated in Fig. A.3a and Fig. A.3c. The basic idea is to fire the same transitions at the same times and to use Lemma 2 to approximate the original trace for the time advancements and inequality event. Consider again the trace:

$$T_F = \sigma_0 \xrightarrow{t_7} \sigma_1 \xrightarrow{R(v)\leftarrow 1.5} \sigma_2 \xrightarrow{10,\Im_1} \sigma_3 \xrightarrow{t_0} \sigma_4 \xrightarrow{t_1} \sigma_5 \xrightarrow{R(v)\leftarrow 1.3} \sigma_6 \xrightarrow{10,\Im_2} \sigma_7 \xrightarrow{t_2} \sigma_8 \xrightarrow{tFail} .$$

To construct the reset trace, the transition events remain the same; however, each sequence of rate changes and time advancements ending with an inequality event are replaced (using Lemma 2) with a trace that starts with the lower rate 1, switches once to the upper rate of 2, and ends with an inequality event that changes the same inequalities. In this case, $\sigma_1 \xrightarrow{R(v)\leftarrow 1.5} \sigma_2 \xrightarrow{10,\mathfrak{I}_1} \sigma_3$ is replaced with $\sigma_1 \xrightarrow{5} \psi_0 \xrightarrow{R(v)\leftarrow 2} \psi_1 \xrightarrow{5,\mathfrak{I}} \sigma_3$ and similarly $\sigma_5 \xrightarrow{R(v)\leftarrow 1.3} \sigma_6 \xrightarrow{10,\mathfrak{I}_2} \sigma_7$ is replaced by $\sigma_5 \xrightarrow{7} \psi_2 \xrightarrow{R(v)\leftarrow 2} \psi_3 \xrightarrow{3,\mathfrak{I}_2} \sigma_7$ to obtain:

$$\begin{aligned} T_r = \sigma_0 \quad &\xrightarrow{t_7} \quad \sigma_1 \xrightarrow{R(v)\leftarrow 1} \psi_0 \xrightarrow{R(v)\leftarrow 2} \psi_1 \xrightarrow{5,\mathfrak{I}_1} \sigma_3 \xrightarrow{t_0} \sigma_4 \xrightarrow{t_1} \\ \sigma_5 \quad &\xrightarrow{7} \psi_2 \xrightarrow{R(v)\leftarrow 2} \psi_3 \xrightarrow{3,\mathfrak{I}_2} \sigma_7 \xrightarrow{t_2} \sigma_8 \xrightarrow{tFail} . \end{aligned}$$

Note that in both states σ_1 and σ_5, the previous event was a transition, so the rate for v is reset to 1. The corresponding derived functions are shown in Fig. A.3(a) and Fig. A.3(c).

With Lemma 2 in place, the proof of Theorem 5 is only a matter of replacing the sequences of events between pairs of inequality events with the results of Lemma 2. The proof is as follows.

Proof of Theorem 5. Let $T = \sigma_0 \xrightarrow{E_0} \sigma_1 \ldots \sigma_n \xrightarrow{E_n} \sigma_{n+1}$ be any trace. Let $sub_I(T) = (E_{i_k})_{k\geq 0}$. The proof proceeds by inductively constructing a sequence $(T_r)_k$ with the property that $sub_I((T_r)_k) = \{E'_k\}$ where for all k, $E'_k = E_{i_k}$ for transition events and E'_k is an inequality event that changes the same set of inequalities as E_{i_k} for inequality events. The base case is so much like the rest of the inductive steps that it is omitted. Suppose $(T_r)_k$ has been constructed, the next step is to construct $(T_r)_{k+1}$. Suppose $E_{i_{k+1}}$ is a transition firing. Since $(E_{i_k})_{k\geq 0}$ is the subsequence of all inequalities changing and transitions firing, it follows that the only events that can occur between E_{i_k} and $E_{i_{k+1}}$ are rate changes and time advancements. Furthermore, the time advancements do not result in a change in the truth value for an inequality. Thus, the truth value of the enabling condition does not change from $S(E_{i_k})$ to the state $P(E_{i_{k+1}})$. Now, since the enabling condition for $E_{i_{k+1}}$ is true in $P(E_{i_{k+1}})$, it must be true in $S(E_{i_k})$. Since transitions fire before any time advancements or rate changes, it is, in fact, the case that the transition $E_{i_{k+1}}$ fires from the state $S(E_{i_k})$. The next part of the sequence is defined as $(T_r)_{k+1} := \sigma_{i_k+1} \xrightarrow{E_{i_{k+1}}} \sigma_{i_{k+1}+1}$.

Now suppose that $E_{i_{k+1}}$ is a time advancement resulting in a set of inequalities changing, that is $\xrightarrow{E_{i_{k+1}}} = \xrightarrow{\tau,\mathfrak{I}}$. In this case, define $(T_r)_{k+1} := \hat{T}_r$ where $\hat{T}_r$ is the trace given in the proof of Lemma 2 based on the trace fragment $\hat{T} = \sigma_{i_k+1} \xrightarrow{E_{i_k+1}} \sigma_{i_k+2} \ldots \sigma_{i_{k+1}} \xrightarrow{E_{i_{k+1}}} \sigma_{i_{k+1}+1}$. It has already been noted that $\hat{T}_r \sim_I \hat{T}$ in the lemma and a check of the trace constructed in Lemma 2 shows that $sub_I(\hat{T}_r) = \{E'_k\}$ where E'_k is an inequality event that changes the same set of inequalities as E_{i_k} as desired.

Once $(T_r)_k$ has been constructed, then define $T_r := (T_r)_0(T_r)_1 \dots$ where juxtaposition is the obvious concatenation of trace fragments. By the construction of the subsequence $(T_r)_k$, $T_r \sim_I T$. □

A.5 Results

The trace based theory presented in this appendix leads to the later development of the zone-based extension presented in this chapter. Thus, it is the zone-based model checking algorithm presented in Section 4.3 that is used in this section to show that the trace based method works.

This section describes two sets of experiments performed using `LEMA`. Each experiment corresponds to a block in Table A.1. The first block deals with methods of allowing the rate to be reset a single time versus allowing multiple resets. In this experiment, the algorithm is not changed, rather the comparison is performed by simulating the single reset method with a translated LPN model. The second set of experiments aims at comparing the complexity of allowing multiple resets of rates versus LPNs that do not assign ranges of rates. In this experiment, the model is changed twice. One version uses only the lower bound rate, the second uses only the upper bound rate. These two examples are then compared with the original model. In all versions, the failure transition is removed from the property LPN so that a full state exploration is performed. Also, in all cases, the algorithm is not changed. Each of these experiments is described in more detail below.

As mentioned above, for the first experiment, a single reset per rate assignment is simulated using a transformed LPN model and is compared against the method of allowing multiple resets as described in Section 4.3. As indicated in Section A.2, if one only performs a single reset for a continuous variable per range of rate assignment then a false positive can occurred. This behavior is explicitly illustrated by transforming the model of Fig. A.1a to that in Fig. A.5 via the procedure described in Chapter 2 of [74]. This transformation replaces the range of rate assignment by an assignment to the lower bound rate of 1. Then, it adds a transition t_9 that sets the rate to the upper bound 2. A Boolean variable r_0 is added to ensure that the transition t_9 is not fired more than once, since setting the rate to the upper bound again after it is already at 2 does nothing. When the model in Fig. A.5 is used with the property LPN in Fig. A.1b, then `LEMA` 'falsely'[1] declares that the model verifies. This is recorded as the 'Reset Once' line of Table A.1. However, when `LEMA` is

[1]Technically the verification is correct since indeed the transformed model does not have an error. However, the transformed model does not provide the same behaviors as the original, which is what is being illustrated.

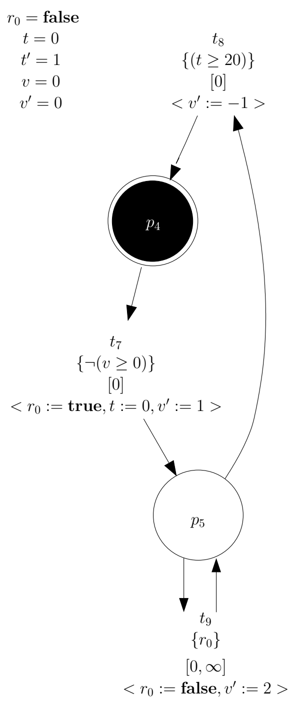


Figure A.5: Fig. A.1a model transformed to only allow a single resetting of the rate for v.

Table A.1: Collected verification results. These results are generated using `LEMA`, a java-based verification tool, on a 64-bit machine running an Intel Core i5 CPU M 480@ 2.67GHz with 4 processors and 4GB of memory.

Property	Time (s)	States	Verifies?	Correct Result?	Figs
Reset Once	0.148	42	yes	no	A.5, A.1b
Resetting Rates	0.170	51	no	yes	A.1a, A.1b
Rate 1 Only	0.147	19	yes	yes	A.1a, A.1b
Rate 2 Only	0.123	34	yes	yes	A.1a, A.1b
No Failure	0.205	91	yes	yes	A.1a, A.1b

applied directly to Fig. A.1, `LEMA` correctly detects the error. This is shown as the second entry in Table A.1. The difference is, of course, that the rate is reset more than once.

For the second experiment, the increase in complexity when allowing a range of rates assignment is investigated by comparing a version of the LPN that uses only a single rate assignment. Suppose a range of rates is assigned to a variable where zero is one of the bounds or is not in the range. Then, every time an inequality changes or a transition fires, the algorithm has to account for two zones instead of one. Thus, one can expect a doubling in the number of states. Similarly, if a second continuous variable is assigned a range of rates, then one has to consider an additional pair of rates independent of the first for every zone that is found using a single rate. This provides an additional factor of 2. Therefore, for each variable assigned to a range of rates, one can expect a doubling in the number of zones that need to be considered. In addition, if zero is strictly between the lower and upper bound rates, then an additional factor of 2 needs to be considered since the rate can either be set to the greatest or lowest rate bound and in each case can then be set to zero. However, in practice, one gets less than this amount since some of the resulting zones may be equal or subsets of the other possibilities allowing the redundant zones to be removed. Of course, this doubling only affects those states where the continuous variable is assigned a range of rates and so the state count is less if the continuous variable is not always assigned a range of rates.

Lines three through four in Table A.1 illustrate the increase in state count for the model of Fig. A.1. The Rate 1 Only line gives the result when running the example of Fig. A.1 with only the lower bound rate being assigned and Rate 2 Only corresponds to only the upper bound rate being assigned. Doubling the state count for each example according to the analysis above, one gets 38 and 68 states moving from having a single rate to a range. Furthermore, the LPN markings that are explored by each example are roughly disjoint since the Rate 1 Only example explores the markings where t_4 fires and the Rate 2 Only example explores the markings where t_1 fires. This leaves a possible 6 markings in common (the markings corresponding to pairing p_4 and p_5 with *branch*, p_0, and *merge*). Since the Resetting Rates line actually explores both sets of markings, a rough estimate for the number of states required would be the sum $38 + 68 - 6 = 100$. To get the number of states for a full state exploration of Fig. A.1 using ranges of rates, the failure transition was removed. The corresponding result is given as the No Failure line of Table A.1. Thus, the number of states is 91, which accords well with the estimate of 100.

A.6 Conclusion

In moving from the theory to an implementation, the first thing one has to account for is that the number of reset traces still remains infinite. For an indication of why, consider again Fig. A.2a where v is a continuous variable with a range of rates of $[1, 2]$, t is a continuous variable with a rate of 1, and time has been allowed to elapse for 10 time units. A reset trace starts at $(0, 0)$ with a rate of 1 and then has the option to set the rate to 2 any time between the 0 to 10 time units. Thus, one has a reset trace corresponding to each point $[0, 10]$ where the switching takes place. When dealing with equivalence classes of traces via subsets of Euclidean space, like polyhedra and zones, this difficulty is in fact an illusion. By grouping the traces together into the subset bounded by the dashed lines in Fig. A.2a, one accounts for all the reset traces, as well as all the other traces. Now, the resulting figure is not a zone since zones only use lines that are horizontal, vertical, or at 45 degrees. However, the figure can be over-approximated by a zone.

Since a zone can be used to collect up all the behaviors in addition to the reset traces, the question arises: why does one need the theory of Section A.1? The true power of the theory presented is not the existence of reset traces, but an explicit representation of the more general idea that all the behaviors of an LPN can be captured as long as one accounts for all possible starting and ending possibilities between the important events (the events required for being inequality equivalent). Putting this into context, consider again the LPN of Fig. A.1. After firing the transition t_7, one has $(v, t) = (0, 0)$, $v' \in [1, 2]$, and $t' = [1, 1]$. The next important events are when time has advanced far enough for either $t \geq 10$ or $v \geq 15$ to change sign. This is illustrated in Fig. A.6. All traces that start at $(0, 0)$ and end along the upper horizontal dashed line are inequality equivalent. Similarly, all traces that start at $(0, 0)$ and end along the vertical dashed line are also inequality equivalent (to each other not to those ending along the horizontal line). The reset traces are able to have a least one member in every inequality equivalence class specifically because they are able to hit these two dashed lines. Then, at the next stage one has to consider every point along these dashed lines as potential new initial points and ensure that the next important events can all be reached.

The above scheme lays out a relatively clear path for an algorithm with a few reservations. Of course, some approximations would have to be made along the way since in general LHAs are undecidable [57] and it is reasonable to assume that LPNs are as well. However, the point of using zones in `LEMA` is to use simple approximating polyhedra that do not have to deal with such general spaces as depicted in Fig. A.6. The desire to use an existing system, such as `LEMA`, brings about a second useful aspect of the theory in Section A.1. As stated before, `LEMA` has the ability to handle rate assignments to continuous variables as long as the rate assignment is to a single rate. The knowledge that all behaviors are captured using only the lower and upper bound rates together with zero provides a straightforward idea for generalizing the algorithms already present in `LEMA`. For example, if zero is not in the range of rates, then at every stage consider the zone produced by using the lower bound rate and consider the zone produced by using the upper bound rate. As an illustration of how this helps, consider a couple zones along an error trace produced when verifying the

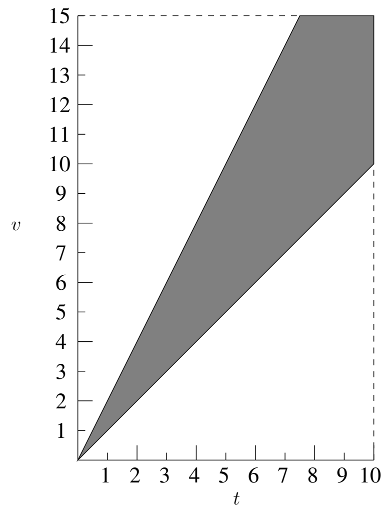


Figure A.6: The set of inequality equivalent traces for Fig. A.1 starting at $(v, t) = (0, 0)$, $v' \in [1, 2]$, and $t' = [1, 1]$.

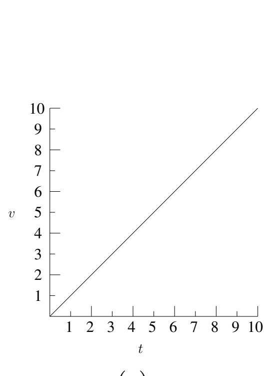
LPN in Fig. A.1. When the analysis reaches the point after firing transition t_7 in Fig. A.6, the next zone formed after assigning the rate of v to 1 is given in Fig. A.7a.

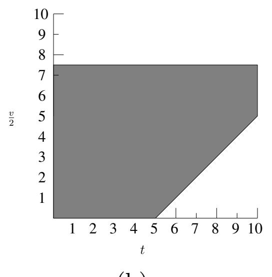
The zone clearly does not capture all the needed behaviors. Since v is 10 when t is 10, the sequence of transitions t_0 and t_1 cannot fire. On the other hand, reset traces suggest that the rate 1 and rate 2 cases should both be considered. Thus, one can consider the rate change event setting the rate of v to 2. After doing this in LEMA, one obtains the zone in Fig. A.7b (in particular, after applying warping and advancing time). In this figure, v has been scaled by a factor of 2. With this zone, the transition sequence t_0 and t_1 is possible since the point $(7.5, 10)$[2] corresponds to v having a value of 15 and t having a value of 10. Notice that this zone encapsulates the polyhedron in Fig. A.6, and so it is able to capture all the possible behaviors for this stage of the analysis.

The next key inspiration from reset traces is that the rate of v should be reset to 1 after the inequalities $v \geq 15$ and $t \geq 10$ change (and after t_0 and t_1 fire). Thus, after firing the inequality changes of $t \geq 10$ and $v \geq 15$ (and resetting the rate of v to 1), the next zone produced is shown in Fig. A.8. The bottom line of this zone does indeed lead to the failure transition being enabled since the bottom right corner is $(v, t) = (24, 20)$.

This example illustrates that the extension of zones by adding additional rate change

[2]LEMA technically stores zones as integers. So this zone is restricted to 7 along the $v/2$ axis. The algorithm realizes that the inequality can change between 7 and 8 and enables the inequality to change.


(a)


(b)

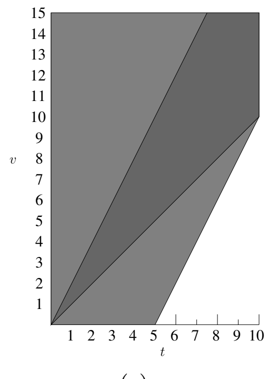

(c)

Figure A.7: Comparison of rate change event with an exact polyhedra. (a) Zone after firing t_7 in Fig. A.1. (b) The zone from (a) after firing the rate change event $R(v) \leftarrow 2$. (c) Comparison of the zone from (b) with the polyhedron in Fig. A.6.

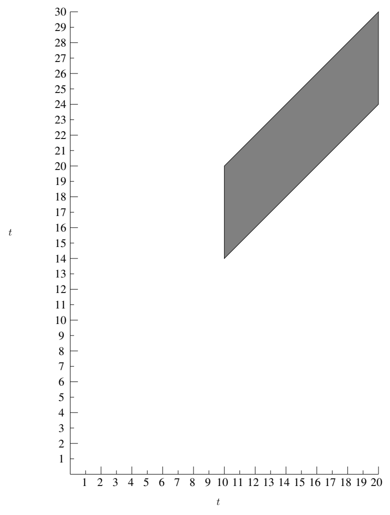


Figure A.8: Zone after starting with Fig. A.7c, firing the inequality events $v \geq 15$, $t \geq 10$ (as well as transitions t_0 and t_1), and advancing time.

events eliminates the false positive that is present in Fig. A.1 when using a method that only allows for one rate change. The argument that this always works is given in Section 4.4.

APPENDIX B

ADDITIONAL REACHABILITY COMPONENTS

This appendix contains the algorithms for the other functions in [72, 74] that have not be explained in Section 4.3. These algorithms are included for completeness and do not require any major modifications in order to implement the range of rates algorithm.

The first algorithm is `initialStateSet` and is shown in Algorithm B.1. The function takes in the set of transitions, the set of continuous variables, the enabling conditions for the transitions, the delay assignment formulas, the initial markings, the initial ranges of values for the continuous variables, and the initial range of rates for the continuous variables. To construct the initial state set, the algorithm needs to provide the marking, the range of values of each rate zero continuous variables, the range of possible values, the current rate of each continuous variable, the truth value of each inequality, and the zone. The markings are provided directly from the initial markings, while the initial rates are provided by the function `resetRates` defined in Section 2.1.2. Next, the current values of the continuous variables are assigned. Additionally, if the rate of the continuous variable is nonzero, then the variable is added to the zone. To fill out the rest of the zone, each transition is tested to determine if it is enable, and if the transition is enabled, a corresponding clock is added to the zone. After the zone is constructed, the algorithm next finds all the inequalities that are involved in the enabling conditions and determines to truth value. Finally, the current delay is evaluated for each of the enabled transitions and the state is returned.

To finish the description of the `initialStateSet`, two additional functions need to be described: `addV` and `addT`. The algorithm `addV` is shown in Algorithm B.2 and handles adding a continuous variable to the zone. The first step is to add the continuous variable to the zones collection of variables. Next, the upper and lower bounds are assigned to the continuous variables in the zone, after being warped by the rate. The function $\texttt{cdiv}(a, b)$ calculates the ceiling of the division $\frac{a}{b}$, while the functions $\texttt{nlb}(Z, v)$ and $\texttt{ub}(Z, v)$ access the negative of the lower bound and upper bound of the zone for the variable v, respectively. Using these functions, the assignment $\texttt{nlb}(Z, v) := -1 * \texttt{cdiv}(q_l(v), R(v))$ assigns the ceiling

Algorithm B.1: `intitialStateSet`$(T, V, En, DA, M_0, Q_0, RR_0)$

```
M := M_0;
R := resetRates(RR_0);
forall the (v ∈ V) do
    if (R(v) ≠ 0) then
        (Q, Z) := addV(Q_0, R, Z, v);
    else
        Q(v) := Q_0(v);
forall the (t ∈ T) do
    if (Eval(En, M_0, R, Q_0, t)) then
        Z := addT(Z, t);
forall the (v ≥ k ∈ ineq(En)) do
    if (Q_0(v) = k) then
        I(v ≥ k) := R(v) ≥ 0)
    else
        I(v ≥ k) := Q_0(v) ≥ k;
forall the (t ∈ Z) do
    D(t) = Eval(DA, R, Q_0, t)
return (M, D, Q, R, RR, I, Z)
```

of the division $\frac{q_l(v)}{R(v)}$ to the negative of the lower bound, which is a conservative estimate of the scaled version of the lower bound for the continuous variable. Similarly, the assignment `ub`(Z, v) := `cdiv`$(q_u(v), R(v))$ assigns a conservative estimate of the upper bound of the continuous variable scaled by the rate. When the rate is negative, the upper and lower bounds are swapped, hence the upper bound value q_u is used to assign `nlb` and q_l is used to assign `ub`. The following forall loop handles assigning the individual relationships between the new variable being added to the zone and the variables already in the zone. These assignments are performed with the aid of the function $Z(x, y)$, which stores the constraint $y - x \leq Z(x, y)$. No relation is assumed between a new continuous variable and the other values in the zone, so the constraints are all assigned to ∞, except the constraint $Z(v, v)$, which is always 0.

Adding a new clock for a transition is a little shorter. The algorithm for `addT` is shown in Algorithm B.3. The first step adds a clock to the zone that records the amount of time the transition has been enabled. When a clock is added, it is initialized to zero, so the next two lines set the negative of the lower bound and the upper bound to zero. Then, the forall loop assigns the new relations for this newly added clock. Similar to adding a variable, $Z(c_t, c_t)$ is always zero, so $Z(c_t, c_t)$ is assigned 0. Next, each upper bound and negative lower bound is assigned to the relations $Z(c_t, x_i)$ and $Z(x_i, c_t)$, respectively. These

Algorithm B.2: `addV`(Q, R, Z, v)

```
1  $Z := Z \cup \{v\}$;
2  if $(R(v) > 0)$ then
3      $\texttt{nlb}(Z, v) := -1 * \texttt{cdiv}(q_l(v), R(v))$;
4      $\texttt{ub}(Z, v) := \texttt{cdiv}(q_u(v), R(v))$;
5  else
6      $\texttt{nlb}(Z, v) := \texttt{cdiv}(q_u(v), R(v))$;
7      $\texttt{ub}(Z, v) := \texttt{cdiv}(q_l(v), R(v))$;
8  forall the $(x_i \in Z)$ do
9      if $(x_i = v)$ then
10         $Z(v, x_i) := 0$;
11     else
12         $Z(v, x_i) := \infty$;
13         $Z(x_i, v) := \infty$;
14 return $(Q, Z)$;
```

Algorithm B.3: `addT`(Z, t)

```
1  $Z = Z \cup \{c_t\}$;
2  $\texttt{nlb}(Z, c_t) := 0$;
3  $\texttt{ub}(Z, c_t) := 0$;
4  forall the $(x_i \in Z)$ do
5      if $(x_i = C_t)$ then
6          $Z(c_t, x_i) := 0$;
7          else
8              $Z(c_t, x_i) := \texttt{ub}(Z, x_i)$;
9              $Z(x_i, c_t) := \texttt{nlb}(Z, x_i)$;
10 return $Z$;
```

assignments have the same effect as assigning the relations to ∞ and then tightening. To see where these assignments originate, consider adding the clock c_t and suppose that x_i is a variable that is already in the zone. When the clock c_t is added, it is initialized as zero. Thus, $0 \leq c_t$. By definition of the function `ub`, $x_i \leq \text{ub}(Z, x_i)$. Subtracting the inequality $0 \leq c_t$ from $x_i \leq \text{ub}(Z, x_i)$, one obtains $Z(c_t, x_i) = x_i - c_t \leq \text{ub}(Z, x_i)$. The justification for assigning $\text{nlb}(Z, x_i)$ to $Z(x_i, c_t)$ is similar.

The next missing function is the `addSetItem` function shown in Algorithm B.4. This function adds events to a set of events, $\mathcal{E}$, to be later returned as the possible next events. The three types of events that can be added to $\mathcal{E}$ are: a set of inequalities that can change at the same time, a transition that can fire, and a variable that can change rates. A rate change event can always occur, so additional processing is only necessary if the event is

Algorithm B.4: addSetItem($T, En, D, R, Z, \mathcal{E}, e_{\text{new}}$)

```
1  if (isNotRateChange($e_{new}$)) then
2      forall the ($E \in \mathcal{E}$) do
3          forall the ($e \in E$) do
4              if ($e_{new} \in ineq(En) \wedge e \in$ ineq($En$)) then
5                  ($E, status$) := happensFirstII($R, Z, e_{\text{new}}, e, E$);
6              else if ($e_{new} \in$ ineq($En$) $\wedge e \in T$) then
7                  ($E, status$) := happensFirstIT($E, R, Z, e_{\text{new}}, e, E$);
8              else if ($e_{new} \in T \wedge e \in$ ineq($En$)) then
9                  ($E, status$) := happensFirstTI($D, R, Z, e_{\text{new}}, e, E$);
10             if ($status = Notpossible \vee status = Possible$) then
11                 return $\mathcal{E}$;
12 $\mathcal{E} = \mathcal{E} \cup \{\{e_{\text{new}}\}\}$;
13 return $\mathcal{E}$;
```

not a rate change. In the case that the event is not a rate change, the rest of the function determines which happensFirst function to call depending on whether the new event is an inequality and the old event is an inequality, the new event is an inequality and the old event is a transition firing, or the new event is a transition and the old event is an inequality. If the result of any of the functions is *Notpossibe* or *Possible*, the result is immediately returned. This statement is bypassed if the two transitions are returned or the result returned by the executed happensFirst function is *undecided*.

The individual happensFirst functions are shown in Algorithms B.5, B.6, B.7. The function happensFirstII (Algorithm B.5) handles comparing two inequality events. It finds how the zone is restricted according to each inequality and then makes the final determination depending on whether the inequalities deal with the same continuous variable or different continuous variables. The function happensFirstIT (Algorithm B.6) determines whether a new inequality event should happen before an old transition by comparing how the inequality restricts the zone to the lower bound delay of the transition. The function happensFirstTI (Algorithm B.7) is nearly the same.

To complete the happensFirst functions, two more functions need to be described: the compareSameV (Algorithm B.8) and the compareDifferentV (Algorithm B.9) functions. These functions determine whether the new inequality must occur before the old inequality. The first function, compareSameV, handles the easy case when the inequalities involve the same continuous variable. If the new inequality restricts the zone to a value after the old inequality, then the inequality does not happen first. Hence, the value returned is *NotPossible*. Conversely, if the old inequality restricts the zone to a value after the new

Algorithm B.5: `happensFirstII`$(R, Z, i_{\text{new}}, i, E)$

```
1 let $i = (v \geq k)$;
2 let $i_{\text{new}} = (v_{\text{new}} \geq k_{\text{new}})$;
3 restrictVal := cdiv($k, R(v)$));
4 restrictVAl$_{\text{new}}$ := cdiv($k_{\text{new}}, R(v_{\text{new}})$);
5 if $(v_{new} = v)$ then
6     return compareSame$V(i_{new}, i, E,$ restrictVAl$_{new}$, restrictVal$)$;
7 else
8     return compareDifferentV$(Z, i_{new}, E,$ restrictVal$_{new}$, restrictVal$)$;
```

Algorithm B.6: `happensFirstIT`$(D, R, Z, i_{\text{new}}, E)$

```
1 let $i_{\text{new}} = (v_{\text{new}} \geq k_{\text{new}})$;
2 restrictVal := $d_l(t)$;
3 restrictVal$_{\text{new}}$;
4 if $(-$restrictVAl$_{new} > -$restrictVal$+ Z(t, v_{new}))$ then
5     $E := E - \{t\}$;
6     return $(E, Undecided)$;
7 else if (restrictVal$_{new} >$ ub$(Z, t) + Z(v_{new}, t))$ then
8     return $(E, NotPossible)$;
```

Algorithm B.7: happensFirstTI$(D, R, Z, t_{\text{new}}, i, E)$

```
1 let $i = (v \geq k)$;
2 restrictVal := cdiv($k, R(v)$);
3 restrictVAl$_{\text{new}}$ := $d_l(t_{\text{new}})$;
4 if $(-$restrictVal $> -$restrictVal$_{new} + Z(t_{new}, v))$ then
5     return $(E, NotPossible)$;
6 else if (restrictVal $>$ restrictVal$_{new} + Z(v, t_{new}))$ then
7     $E := E - \{i\}$;
8     return $(E, Undecided)$;
```

inequality, then the new inequality happens first. The value *Undecided* is returned since the new inequality may happen after another inequality not being compared. Finally, if the two inequalities restrict the zone by the same amount, then the new inequality is added to the same set of inequality events as the old inequality and the result is returned with the status of *Possible*.

The `compareDifferentV` function is similar, however, since the inequalities depend on different variables, the relationship between the two variables has to be taken into account. The first and fourth cases determine if the new inequality would result in a restriction beyond the old inequality, while the second and third cases determine that the new inequality

Algorithm B.8: compareSameV($i_{\text{new}}, i, E,$ restrictVal$_{\text{new}}$, restrictVal)

```
1  if (restrictVal$_{new}$ > restrictVal) then
2      return $(E, NotPossible)$;
3  else if (restrictVal > restrictVal$_{new}$) then
4      $E := E - \{i\}$;
5      return $(E, Undecided)$;
6  else
7      $E := E \cup \{i_{\text{new}}\}$;
8      return $(E, Possible)$;
```

Algorithm B.9: compareDifferentV($Z, i_{\text{new}}, i, E,$ restrictVal$_{\text{new}}$, restrictVal)

```
1  let $i = (v \geq k)$;
2  let $i_{\text{new}} = (v_{\text{new}} \geq k_{\text{new}})$;
3  if ($-$restrictVal > $-$restrictVal$_{new}$ + $Z(v_{new}, v)$) then
4      return $(E, NotPossible)$;
5  else if ($-$restrictVal$_{new}$ > $-$restrictVal + $Z(v, v_{new})$) then
6      $E := E - \{i\}$;
7      return $(E, Undecided)$;
8  else if (restrictVal > nlb($v_{new}$) + $Z(v, v_{new})$) then
9      $E := E - \{i\}$;
10     return $(E, Undecided)$;
11 else if (restrictVal$_{new}$ > nlb($v$) + $Z(v_{new}, v)$) then
12     return $(E, NotPossible)$;
13 else
14     $E := E \cup \{i_{\text{new}}\}$;
15     return $E, Possible)$;
```

happens before the old inequality. Just like for the compareSameV function, the last case handles when the restrict values are the same.

The last function to describe before completing the description of findPossibleEvents is the ineqCanChange function shown in Algorithm B.10. Consider an inequality $v \geq k$. The only way the inequality will change truth value is if the continuous variable crosses the boundary condition c. The variable can cross the boundary in two ways: the value of v is less than the boundary c and the rate is positive or the value of v is greater than the boundary c and the rate is negative. Since the truth value of $v \geq c$ indicates whether the value of v is greater than or less than c, the condition can be equivalently stated as the inequality is **false** and the rate is positive or the inequality is **true** and the rate is negative. If one of these two conditions is met and the value of the inequality is at the boundary, then the inequality is about to change truth value. To check if the inequality is at the bounds,

Algorithm B.10: `ineqCanChange`(R, I, Z, i)

```
if $((R(v) < 0 \wedge I(i)) \vee (R(v) > 0 \wedge \neg I(i)))$ then
    if $(\texttt{ub}(Z, v) \geq \texttt{fdiv}(k, R(v)))$ then
        return $true$;
return $false$;
```

the upper bound of v in the zone Z is compared with the scaled version of k, that is the value $\frac{k}{R(v)}$. Since zones are restricted to integers, the upper bound of v is compared to the floor of the division given by `fdiv`.

The next set of algorithms describe the missing functions for the `updateState` and the functions that they depend on. The first function is the `restrict` function shown in Algorithm B.11. The `restrict` function constrains the zone according to the information provided by which event occurred. If a transition fires, the time has advanced at least as far as the lower bound delay. Thus, the lower bound of the associated clock is set to the lower bound delay. If an inequality is changing truth value, then time has advanced far enough that the associated continuous variable has reached the boundary. Thus, the lower bound of the continuous variable is adjusted to the boundary. However, values in the zone are scaled by the rate, so the lower bound is set to the boundary divided by the rate. If the upper bound is also lower than the boundary, then the upper bound is adjusted to the boundary to keep the zone consistent. An event can consist of more than one inequality, so the restriction is made for each inequality in the event.

The `restrict` function simply changes the lower bound for the transition firing or the lower bounds of the continuous variables involved in a set of inequalities; the function does not change the values of the rest of the zone to reflect the restriction. Thus, the constraints need to be retightened. The function `recanonicalize` performs this retightening by running Floyd's All-Pairs Shortest Path Algorithm. The algorithm is shown in Algorithm B.12.

Restriction and recanonicalization are two steps necessary for firing a transition; however, several other steps are also needed. The function `fireTransition` (see Algorithm B.13) handles the steps for updating the state that are unique to firing a transition. The algorithm updates the markings, removes the fired transition, makes any new assignments to the continuous variables, makes any new assignments to the range of rates, and updates the values in the zone for each continuous variable that is assigned a new range of values.

To update the value of the continuous variable, the function `fireTransition` uses the function `doVarAsgn` shown in Algorithm B.14. The algorithm loops through all the variables v such that the transition t makes an assignment to v. The variable is then removed from the

Algorithm B.11: `restrict`(T, D, Z, E)

```
forall the $(e \in E)$ do
    if $(e \in T)$ then
        let $t = e$;
        nlb$(Z, t) := -1 * d_l(t)$;
    else
        forall the $((v \geq k) \in e)$ do
            nlb$(Z, v) :=$ cdiv$(k, R(v))$;
            if $($ub$(Z, v) <$ cdiv$(k, R(v)))$ then
                ub$(Z, V) :=$ cdiv$(k, R(v))$;
return $Z$;
```

Algorithm B.12: `recanonicalize`(Z)

```
forall the $(x_i \in Z)$ do
    forall the $(x_j \in Z)$ do
        forall the $(x_k \in Z)$ do
            if $(Z(x_j, x_k) > Z(x_j, x_i) + Z(x_i, x_k))$ then
                $Z(x_j, x_k) = Z(x_j, x_i) + Z(x_i, x_k)$;
return $Z$;
```

zone, the assignment is made, and the variable is added back into the zone. The removing and adding the variable has the effect of removing any relations between the continuous variable v and other elements of the zone.

After a transition fires or a set of inequalities change value, time must be advanced. The function `advanceTime` shown in Algorithm B.15 handles this step. Time is advanced by setting the upper bounds on the transitions to the largest value before the transition will fire, that is, the upper bound. The upper bounds for the continuous variables are set to the largest value possible before an inequality changes. The function `checkIneq` shown in Algorithm B.16 finds this value. The algorithm starts by setting the time advancement to ∞. Next, the algorithm considers every inequality that involves the current continuous variable and determines which boundaries will eventually be crossed. Each time a boundary is found that the variable will cross, min is set to the smaller of the current value of min and the time it will take to cross the boundary. To determine if a boundary will be crossed, the algorithm considers whether the rate is positive or negative, whether the inequality is **true** or **false**, and whether the upper bound is less than the boundary or not.

Algorithm B.13: `fireTransition`$(M, F, Q, V, VA, RA, Z, \psi, t)$

```
$M := (M - \bullet t) \cup t\bullet$;
$Z :=$ rmT$(Z, t)$;
$(Q, Z) :=$ doVarAsgn$(Q, R, Z, VA, t)$;
$RR :=$ EvalAssign$(RA(t, v), Q, I, Z)$;
forall the $(v \in V \wedge VA(t) \neq \emptyset)$ do
    if $(v \notin Z \wedge 0 \notin RR(v))$ then
        $(Q, Z) :=$ addV$(Q, R, Z, v)$;
    else if $(v \in Z \wedge 0 \in RR(v))$ then
        $(Q, Z) :=$ rmV$(Q, R, Z, v)$;
$(R, Z) :=$ dbmWarp$(R, R', Z)$;
return $\psi$;
```

Algorithm B.14: `doVarAsgn`(Q, R, Z, VA, t)

```
forall the $(v \in V \wedge VA(t) \neq \emptyset)$ do
    $[a_l, a_u] :=$ EvalAssign$(VA(t, v), Q, I, Z)$;
    $(Q, Z) :=$ rmV$(Q, R, Z, v)$;
    $Q(v) := [a_l, a_u]$;
    $(Q, Z) :=$ addV$(Q, R, Z, v)$;
return $(Q, Z)$;
```

Algorithm B.15: `advanceTime`(En, D, R, I, Z)

```
forall the $(t \in Z)$ do
    ub$(Z, t) := d_u(t)$;
forall the $(v \in Z)$ do
    ub$(Z, v) :=$ checkIneq$(En, R, I, Z, v)$;
return $Z$;
```

Algorithm B.16: `checkIneq`(En, R, I, Z, v)

```
min := $\infty$;
forall the $((v_i \geq k_i) \in \texttt{ineq}(En) \wedge v_i = v)$ do
    if $(R(v) > 0)$ then
        if $(\neg I(v_i \geq k_i))$ then
            if $(\texttt{ub}(Z, v) < \texttt{fdiv}(k_i, R(v)))$ then
                min := min(min, $\texttt{fdiv}(k_i, R(v))$);
            else if $(\texttt{nlb}(Z, p) \leq \texttt{fdiv}(k_i, R(v)))$ then
                min := min(min, $\texttt{ub}(Z, v)$);
    else
        if $(I(v_i \geq k_i))$ then
            if $(\texttt{ub}(Z, v) \leq \texttt{fdiv}(k_i, R(v)))$ then
                min := min(min, $\texttt{fdiv}(k_i, R(v))$);
            else if $(\texttt{nlb}(Z, p) < \texttt{fdiv}(k_i, R(v)))$ then
                min := min(min, $\texttt{ub}(Z, v)$);
return min;
```

APPENDIX C

DERIVING THE WARPING EQUATIONS

For this appendix, recall from Section 5.2.4, that V_i and V_j be continuous variables with rates r_i and r_j, respectively. Since the values in the octagon are scaled, the variables used for the DBM are $x = \frac{V_i}{r_i}$ and $y = \frac{V_j}{r_j}$. Fig. C.1a shows an arbitrary octagon in the x, y-plane. Now, suppose that V_i is assigned a rate of r'_i and V_j is a assigned a rate of r'_j. Then, the new scaling is $u = \frac{V_i}{r'_i}$ and $v = \frac{V_j}{r'_j}$, and the new figure is shown in Fig. 5.14b. Let $\alpha = \frac{r_i}{r'_i}$ and $\beta = \frac{r_j}{r'_j}$, then α and β transform the x, y-plane into the u, v-plane, that is,

$$u = \alpha x \qquad\qquad v = \beta y.$$

Again, the original octagon can be described by:

$$D = \begin{array}{c} \\ x^+ \\ x^- \\ y^+ \\ y^- \end{array} \begin{array}{c} \begin{array}{cccc} x^+ & x^- & y^+ & y^- \end{array} \\ \begin{pmatrix} 0 & -2m_x & b_1 & -b_4 \\ 2M_x & 0 & b_3 & -b_2 \\ -b_2 & -b_4 & 0 & -2m_y \\ b_3 & b_1 & 2M_y & 0 \end{pmatrix} \end{array}$$

for Fig. C.1a and

$$D' = \begin{array}{c} \\ u^+ \\ u^- \\ v^+ \\ v^- \end{array} \begin{array}{c} \begin{array}{cccc} u^+ & u^- & v^+ & v^- \end{array} \\ \begin{pmatrix} 0 & -2m_u & b'_1 & -b'_4 \\ 2M_u & 0 & b'_3 & -b'_2 \\ -b'_2 & -b'_4 & 0 & -2m_v \\ b_3 & b_1 & 2M_v & 0 \end{pmatrix} \end{array}$$

for Fig. C.1b. In the DBM representation, the constants b_1, b_2, b_3, and b_4 are the y intercepts of the bounding lines:

$$y - x \leq b_1 \qquad -y + x \leq -b_2 \qquad y - x \leq b_3 \qquad -y - x \leq -b_4.$$

and the constants b'_1, b'_2, b'_3, and b'_4 are the u intercepts of the bounding lines:

$$v - u \leq b'_1 \qquad -v + u \leq -b'_2 \qquad v - u \leq b'_3 \qquad -v - u \leq -b'_4.$$

For the first set of warping equations, suppose that $\alpha > 0$ and $\beta > 0$. Then, the new

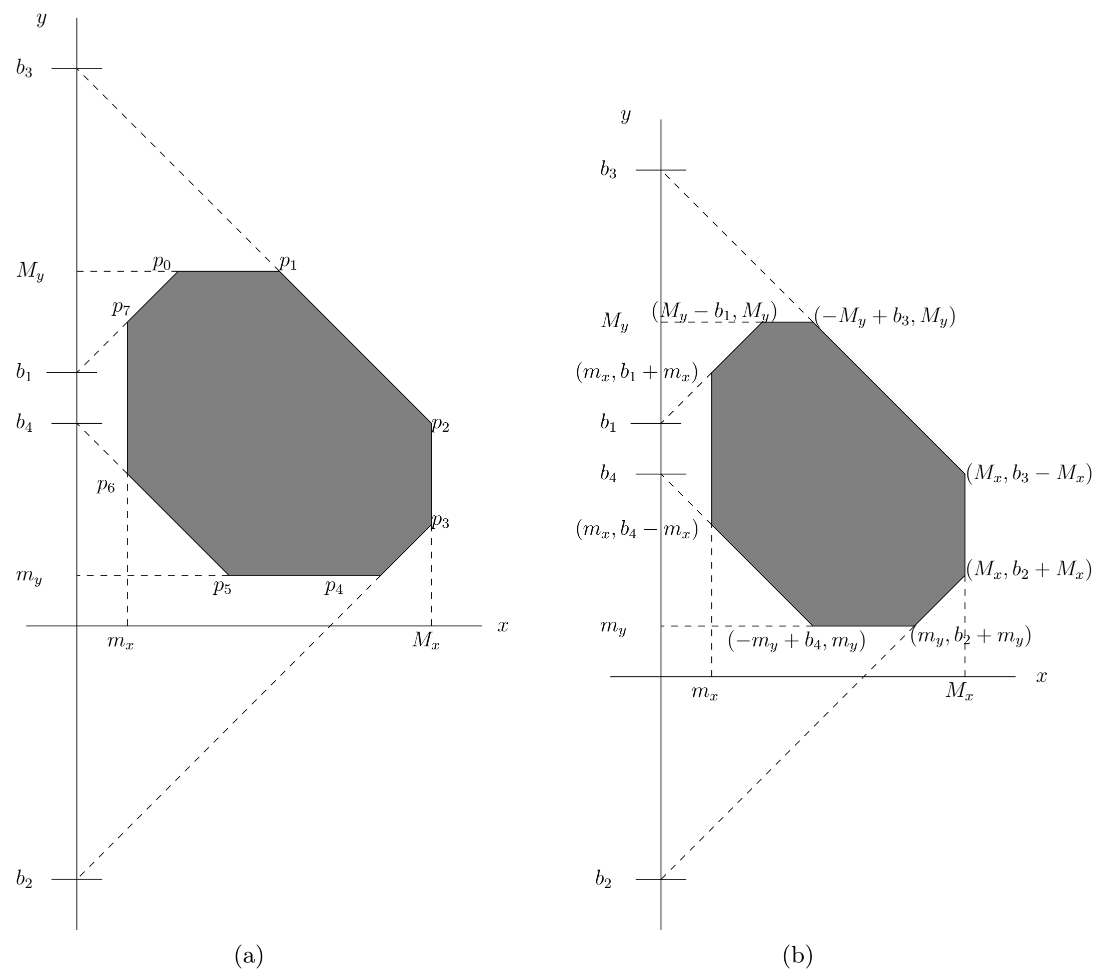


Figure C.1: Labeled octagon. (a) An octagon with y-intercepts labeled b_1, b_2, b_3, and b_4, and vertices labeled $p_0, \ldots, p_7$. (b) An octagon with the vertex labels of (a) replaced with their coordinates in terms of the maximum values, minimum values, and y-intercepts.

minimum and maximum values are given by:

$$m_u = \alpha m_x$$

$$m_v = \alpha m_y$$

$$M_u = \alpha M_x$$

$$M_y = \alpha M_y.$$

Also, in this case, the warping equations are:

$$\begin{aligned} b_1' &= (\beta - \alpha)M_y + \alpha b_1 \\ b_2' &= (\beta - \alpha)m_y + \alpha b_2 \\ b_3' &= (\beta - \alpha)M_y + \alpha b_3 \\ b_4' &= (\beta - \alpha)m_y + \alpha b_4, \end{aligned}$$

when $\frac{\beta}{\alpha} > 1$, and

$$\begin{aligned} b_1' &= (\beta - \alpha)m_x + \beta b_1 \\ b_2' &= (\beta - \alpha)M_x + \beta b_2 \\ b_3' &= (\alpha - \beta)M_x + \beta b_3 \\ b_4' &= (\alpha - \beta)m_x + \beta b_4, \end{aligned}$$

when $\frac{\beta}{\alpha} < 1$.

These equations are not difficult to derive once a couple of simple observations are made. First, label the vertices $p_0, \ldots, p_7$ of the octagon starting with the intersection of M_y and $y - x = b_1$ and labeling clockwise (see Fig. C.1a). The coordinates of each of the vertices can be found by plugging one of the bounds m_x, M_x, m_y, or M_y for either x or y in the bounding equations $y - x = b_1, y - x = b_2, y + x = b_3$, and $y + x = b_4$ and solving for the missing variable. For example, the y coordinate of p_0 is M_y, and since it lies on the line $y - x = b_1$, the x coordinate, p_0^x, of p_0 satisfies $M_y - p_0^x = b_1$. Solving this last equation gives $p_0^x = M_y - b_1$, so the point is $p_0 = (M_y - b_1, M_y)$. For completeness, the whole list is:

$$\begin{aligned}
&p_0 \text{ on line } y - x = b_1 : && y = M_y \implies x = M_y - b_1 \\
&p_1 \text{ on line } y + x = b_3 : && y = M_y \implies x = b_3 - M_y \\
&p_2 \text{ on line } y + x = b_3 : && x = M_x \implies y = b_3 - M_x \\
&p_3 \text{ on line } y - x = b_2 : && x = M_x \implies y = M_x + b_2 \\
&p_4 \text{ on line } y - x = b_2 : && y = m_y \implies x = m_y - b_2 \\
&p_5 \text{ on line } y + x = b_4 : && y = m_y \implies x = b_4 - m_y \\
&p_6 \text{ on line } y + x = b_4 : && x = m_x \implies y = b_4 - m_x \\
&p_7 \text{ on line } y - x = b_1 : && x = m_x \implies y = b_1 + m_x.
\end{aligned}$$

So the points are:

$$\begin{aligned}
&p_0 : (M_y - b_1, M_y) \quad && p_1 : (b_3 - M_y, M_y) \quad && p_2 : (M_x, b_3 - M_x) \quad && p_3 : (M_x, M_x + b_2) \\
&p_4 : (m_y - b_2, m_y) \quad && p_5 : (b_4 - m_y, m_y) \quad && p_6 : (m_x, b_4 - m_x) \quad && p_7 : (m_x, b_1 + m_x),
\end{aligned}$$

as is illustrated in Fig. C.1b. After performing the coordinate change $u = \alpha x$ and $v = \beta y$, the points become $p'_0, \ldots, p'_7$ and are given by:

$$\begin{aligned}
&p'_0 : (\alpha(M_y - b_1), \beta M_y) && p'_1 : (\alpha(b_3 - M_y), \beta M_y)\\
&p'_2 : (\alpha M_x, \beta(b_3 - M_x)) && p'_3 : (\alpha M_x, \beta(M_x + b_2))\\
&p'_4 : (\alpha(m_y - b_2), \beta m_y) && p'_5 : (\alpha(b_4 - m_y), \beta m_y)\\
&p'_6 : (\alpha m_x, \beta(b_4 - m_x)) && p'_7 : (\alpha m_x, \beta(b_1 + m_x)).
\end{aligned}$$

As indicated by the warping equations, two separate conditions need to be addressed, when $\frac{\beta}{\alpha} > 1$ and when $\frac{\beta}{\alpha} < 1$. In both cases, the $\pm 45°$ lines in the xy-coordinate system are no longer $\pm 45°$ after the coordinate transformation. Specifically, the line segments p_7p_0, p_1p_2, p_3p_4, and p_5p_6, where juxtaposition indicates the line segment between the two vertices, are transformed into line segments $p'_7p'_0$, $p'_1p'_2$, $p'_3p'_4$, and $p'_5p'_6$ that no longer belong to any line of the form $\pm y \pm x = b$. To keep a $\pm 45°$ boundary line for the polygon, the best that can be done is to define b'_1, b'_2, b'_3, and b'_4 such that one of the endpoints belongs to the appropriate line $y - x = b'_1$, $x - y = b'_2$, $y + x = b'_3$, or $-y - x = b'_4$ and the other endpoint belongs to the appropriate half-plane $y - x \leq b'_1$, $y - x \leq -b'_2$, $y + x \leq b'_3$, or $-y - x \leq -b'_4$. The case when $\frac{\beta}{\alpha} > 1$ is shown in Fig. C.2. Notice that when transforming the octagon from the xy-plane (Fig. C.2a) to the uv-plane (Fig. C.2b), the slopes of the line segments p_7p_0 and p_3p_4 become more than $45°$ while the slopes of the line segments p_1p_2 and p_5p_6 are less than $-45°$. This fact forces the lines $y - x = b'_1$, $x - y = b'_2$, $y + x = b'_3$, or $-y - x = b'_4$ to be tangent to the polygon at p'_0, p'_1, p'_4, and p'_5. In other words, the best approximating lines of the form $y - x = b'_1$ and $x - y = b'_2$ that lie above the polygon are exactly the lines that contain the points p'_0 and p'_1, respectively. Similarly, the best approximating lines of the form $y + x = b'_3$ and $-y - x = b'_4$ that lie below the polygon contain the points p'_5 and p'_4, respectively.

The intercepts b'_1, b'_2, b'_3, and b'_4 can now be found by plugging in the corresponding point on the line as shown below:

$$\begin{aligned}
&p'_0 = (\alpha(M_y - b_1), \beta M_y) \text{ on line } y - x = b'_1 : && \beta M_y - \alpha(M_y - b_1) = b'_1\\
&p'_4 = (\alpha(m_y - b_2), \beta m_y) \text{ on line } y - x = b'_2 : && \beta m_y - \alpha(m_y - b_2) = b'_2\\
&p'_1 = (\alpha(b_3 - M_y), \beta M_y) \text{ on line } y + x = b'_3 : && \beta M_y + \alpha(b_3 - M_y) = b'_3\\
&p'_5 = (\alpha(b_4 - m_y), \beta m_y) \text{ on line } y + x = b'_4 : && \beta m_y + \alpha(b_4 - m_y) = b'_4.
\end{aligned}$$

Gathering together like terms yields the warping equations for the case when $\frac{\beta}{\alpha} > 1$.

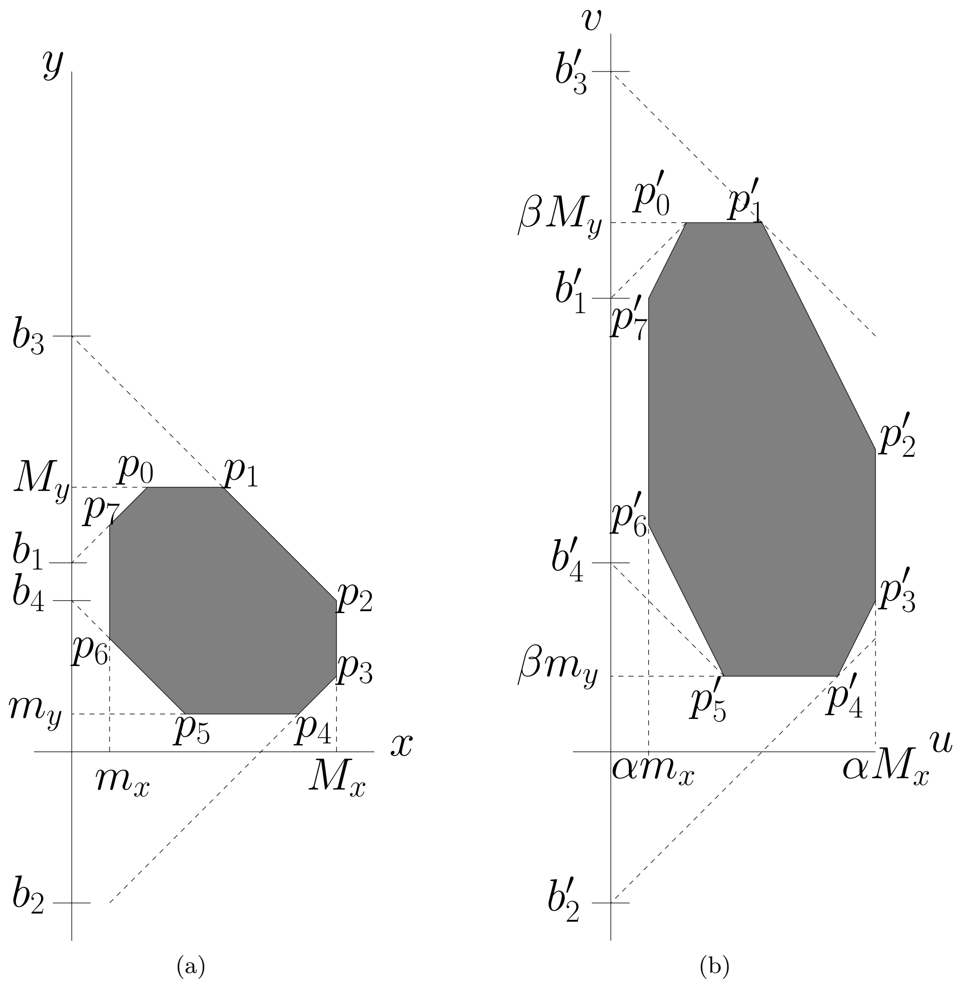


Figure C.2: Warping when $\frac{\beta}{\alpha} > 1$. (a) Original octagon. (b) Warped octagon.

The case when $\frac{\beta}{\alpha} < 1$ can be derived in a similar fashion as above. The only thing that changes is that the line segments $p'_7p'_0$ and $p'_3p'_4$ have slopes less than 45° while the segments $p'_1p'_2$ and $p'_5p'_6$ have slopes greater than $-45°$. This fact translates to the best approximating lines lying above the polygon are tangent to p'_7 and p'_2 instead of p'_0 and p'_1. Similarly, the best approximating lines lying below the polygon are tangent to p'_6 and p'_3 instead of p'_5 and p'_4 (see Fig. C.3). From this point, one can do the exact same procedure of plugging the point into the corresponding lines:

$$p'_7 : (\alpha m_x, \beta(b_1 + m_x)) \text{ on line } y - x = b'_1 : \qquad \beta(b_1 + m_x) - \alpha m_x = b'_1$$
$$p'_3 : (\alpha M_x, \beta(M_x + b_2)) \text{ on line } y - x = b'_2 : \qquad \beta(M_x + b_2) - \alpha M_x = b'_2$$
$$p'_2 : (\alpha M_x, \beta(b_3 - M_x)) \text{ on line } y + x = b'_3 : \qquad \beta(b_3 - M_x) + \alpha M_x = b'_3$$
$$p'_6 : (\alpha m_x, \beta(b_4 - m_x)) \text{ on line } y + x = b'_4 : \qquad \beta(b_4 - m_x) + \alpha m_x = b'_4.$$

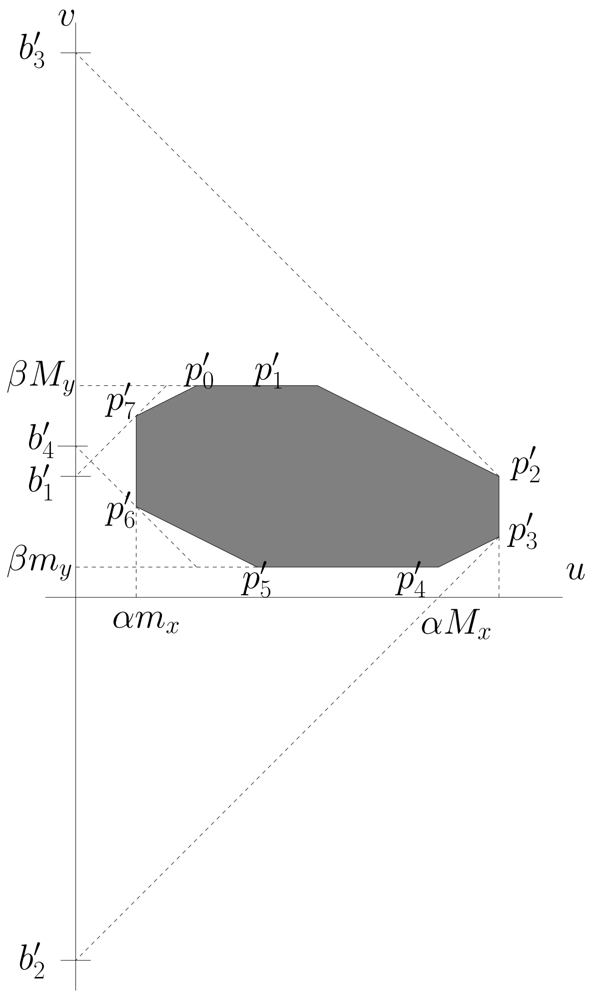


Figure C.3: Warping when $\frac{\beta}{\alpha} < 1$.

As before, the warping equations for $\frac{\beta}{\alpha} < 1$ follow by gathering together like terms.

Between the first set of equations and the second set, there seems to be an asymmetry. With the first set of equations, $(\beta - \alpha)$ is all that is required; however, in the second set of equations both factors $(\beta - \alpha)$ and $(\alpha - \beta)$ are present. This asymmetry is caused by writing all the equations in terms of the y-intercepts. Before illustrating this fact, one needs to know how the intercepts change when considering the subset as being in the xy-plane versus being in the yx-plane, that is, whether it is the x or the y-axis that is horizontal. Regardless of which plane one is in, the set of points is the same; however, when in the xy-plane, one thinks of the boundary lines (except the vertical ones) as functions of x and in the yx-plane, one thinks of the boundary lines (again, except the vertical) as functions of y. In terms of equations, switching between the two planes amounts to interchanging x and y. With this correspondence in mind, consider the boundary conditions $y - x \leq b_1$,

$-y+x \le -b_2$, $y+x \le b_3$, and $-y-x \le -b_4$. Let c_1, c_2, c_3, and c_4 be the corresponding values for the yx-plane, that is the x intercepts. Then c_1, c_2, c_3, and c_4 are x-intercepts for the boundary conditions $x-y \le c_1$, $-x+y \le -c_2$, $x+y \le c_3$, and $-x-y \le -c_4$. By comparing the equations that give the same inequalities, one obtains:

$$c_1 = -b_2 \qquad c_2 = -b_1 \qquad c_3 = b_3 \qquad c_4 = b4.$$

Using this correspondence (and the related version for the primed constants), the equations then become:

$$\begin{aligned}
-c_2' &= (\beta-\alpha)m_x + \beta(-c_2)\\
-c_1' &= (\beta-\alpha)M_x + \beta(-c_1)\\
c_3' &= (\alpha-\beta)M_x + \beta c_3\\
c_4' &= (\alpha-\beta)m_x + \beta c_4,
\end{aligned}$$

which simplifies to the following equations where the first and second equations have been switched:

$$\begin{aligned}
c_1' &= (\alpha-\beta)M_x + \beta c_1\\
c_2' &= (\alpha-\beta)m_x + \beta c_2\\
c_3' &= (\alpha-\beta)M_x + \beta c_3\\
c_4' &= (\alpha-\beta)m_x + \beta c_4.
\end{aligned}$$

This final set of equations has exactly the form as the first set of equations (after noting that reversing x and y also reverse α and β). In fact, this translation provides an alternate way of deriving the warping equations for the $\frac{\beta}{\alpha} < 1$ case. Namely, start with the octagon in the xy-plane, then translate it into the yx-plane (Fig. C.4), using the above correspondence. In the yx-plane, the rate ratio to consider is $\frac{\alpha}{\beta}$. Since $\frac{\beta}{\alpha} > 1$, the inverse ratio gives $\frac{\alpha}{\beta} > 1$, and so, the first version of warping applies with the substitutions:

$$\begin{array}{llll}
m_y \mapsto m_x & M_y \mapsto M_x & \alpha \mapsto \beta & \beta \mapsto \alpha\\
b_1 \mapsto -b_2 & b_2 \mapsto -b_1 & b_3 \mapsto b_3 & b_4 \mapsto b4\\
b_1' \mapsto -b_2' & b_2' \mapsto -b_1' & b_3' \mapsto b_3 & b_4' \mapsto b4,
\end{array}$$

yielding the equations:

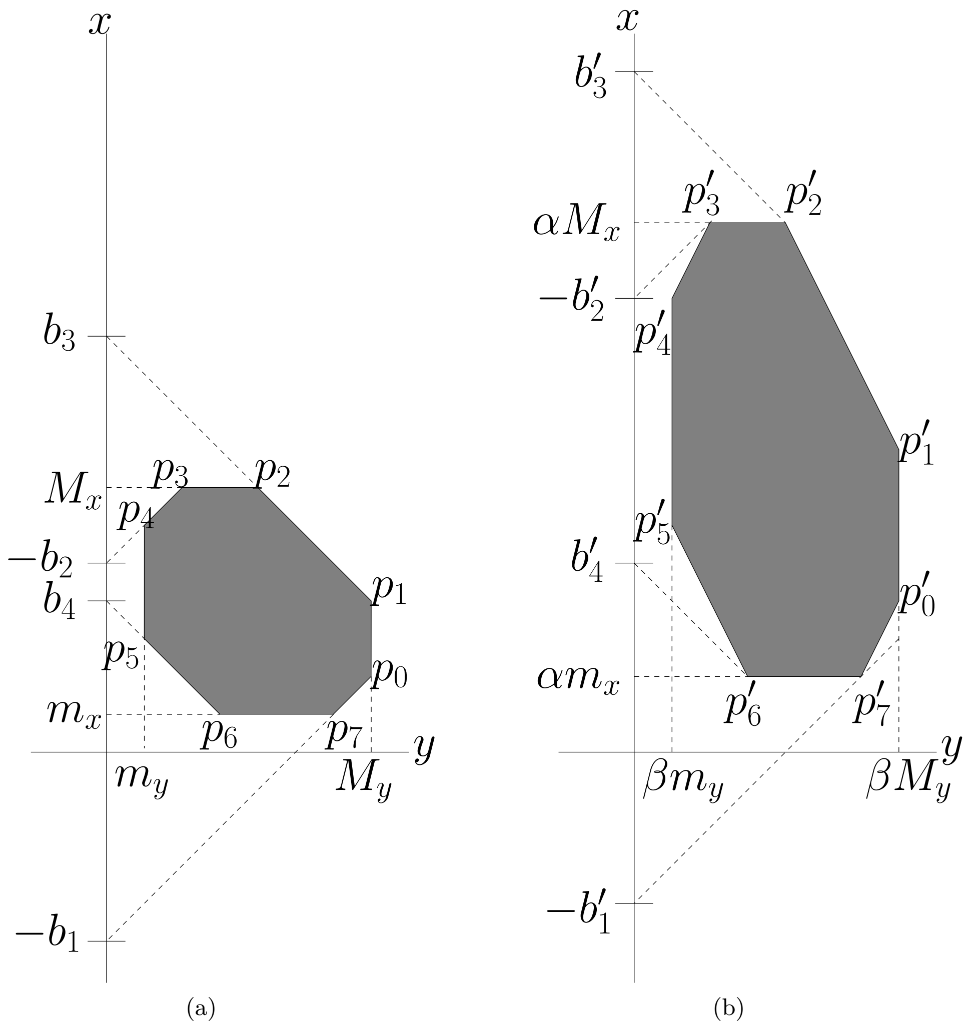


Figure C.4: Warping in the yx-plane when $\frac{\beta}{\alpha} < 1$.

$$
\begin{aligned}
-b'_2 &= (\alpha - \beta)M_x + \beta(-b_2) \\
-b'_1 &= (\alpha - \beta)m_x + \beta(-b_1) \\
b'_3 &= (\alpha - \beta)M_x + \beta b_3 \\
b'_4 &= (\alpha - \beta)m_x + \beta b_4,
\end{aligned}
$$

which matches the second form of the warping equations after a few simplifications.

So far, the only cases that have been consider are the ratios of α and β when $\alpha, \beta > 0$. The cases when at least one of the rate ratios is negative[1]. Before moving onto the general

[1]The cases when at least one of α or β is zero are not considered since rate zero variables are not stored in the octagon.

case, first consider the rate ± 1 cases, that is, the cases when (α, β) is $(1, -1)$, $(-1, 1)$, and $(-1, -1)$.

When $\alpha = -1$ and $\beta = 1$, then the coordinate transformation $u = -x$, $v = y$ amounts to a reflection about the y-axis. Furthermore, since $m_x \leq x \leq M_x$, after the coordinate change $-M_x \leq u \leq -m_x$. So, this inequality gives the first part of the transformation:

$$m_x \mapsto -M_x \qquad\qquad M_x \mapsto -m_x.$$

For the intercepts, recall that x and y satisfy the equations:

$$y - x \leq b_1 \qquad -y + x \leq -b_2 \qquad y + x \leq b_3 \qquad -y - x \leq -b_4.$$

After applying the coordinate transformation $u = -x$ and $v = y$, these equations become:

$$v + u \leq b_1 \qquad -v - u \leq -b_2 \qquad v - u \leq b_3 \qquad -v + u \leq -b_4.$$

By comparing the corresponding equations, one obtains the intercept correspondence:

$$b_1 \mapsto b_3 \qquad b_2 \mapsto b_4 \qquad b_3 \mapsto b_1 \qquad b_4 \mapsto b_2.$$

The first and third equations say that b_1 and b_3 are swapped, while the second and fourth equations say that b_2 and b_4 are swapped. Geometrically, this is clear. The two diagonal lines above the octagon, remain above the octagon; however, they switch sides under a reflection across the y-axis. Similarly, the two diagonal lines below the octagon, remain below the octagon, but they switch sides. The effect of the reflection across the y-axis is shown in Fig. C.5. After reflecting (Fig. C.5b), the upper right bounding diagonal ($y + x \leq b_3$) becomes the upper left bounding diagonal ($y - x \leq b_3$) and vice versa. Since the upper left bounding line gives the b_1 intercept in the uv-plane and the upper right bounding line gives the b_3 intercept, it follows that the intercepts b_1 and b_3 swap values in the uv-plane. This observation matches the mapping $b_1 \mapsto b_3$ and $b_3 \mapsto b_1$. The similar swapping $b_2 \mapsto b_4$ and $b_4 \mapsto b_2$ is shown Fig. C.5 by considering the effect of the transformation on lower left and right bounding diagonals.

The coordinate transformation $u = x$ and $v = -y$ is nearly as simple. This time the upper and lower bounds for y switch places with a minus sign yielding:

$$m_y \mapsto -M_y \qquad\qquad M_y \mapsto -m_y.$$

Before describing the intercept change, again recall that the intercepts satisfy:

$$y - x \leq b_1 \qquad -y + x \leq -b_2 \qquad y + x \leq b_3 \qquad -y - x \leq -b_4.$$

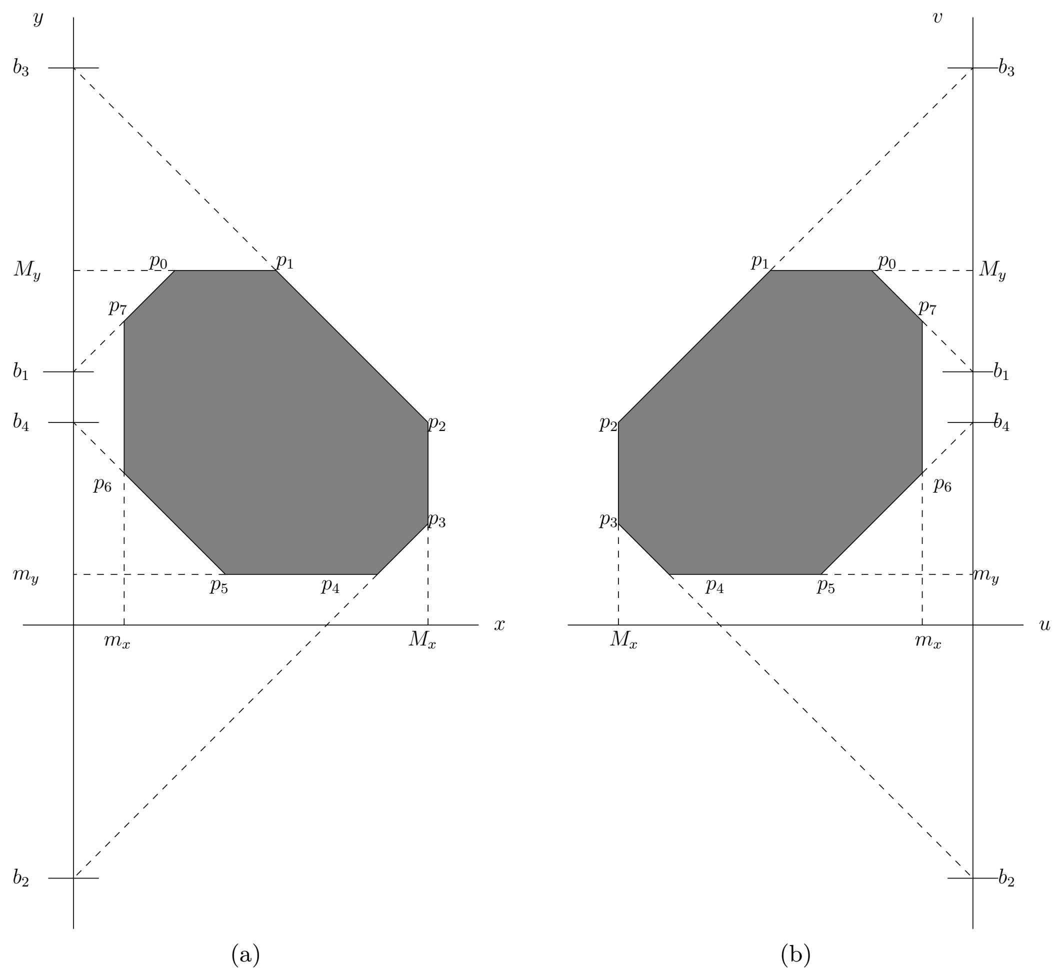


Figure C.5: The effect of the coordinate transformation $u = -x$ and $v = y$. (a) The original octagon. (b) The octagon after the coordinate change.

After the coordinate transformation, the equations become:

$$-v - u \le b_1 \qquad v + u \le -b_2 \qquad -v + u \le b_3 \qquad v - u \le -b_4.$$

By comparing the corresponding inequalities, one gets the following correspondence:

$$b_1 \mapsto -b_4 \qquad b_2 \mapsto -b_3 \qquad b_3 \mapsto -b_2 \qquad b_4 \mapsto -b_1.$$

Similar to the previous case, this coordinate change results in a reflection. This time it is about the x-axis. In this case, the upper left diagonal and the lower left diagonal switch roles, as well as the upper right diagonal and the lower right diagonal. This swapping is again reflected in the pairs of mappings above. Unlike the previous case, though, the transformation introduces a set of minus signs on the intercepts. This additional sign is another effect of having the intercepts defined as y-intercepts. Since the coordinate change flips the sign of the y values, the sign of the intercepts are similarly changed. The coordinate change is illustrated in Fig C.6.

The coordinate change $u = -x$ and $v = -y$ is simply the combination of the previous two coordinate changes. It does not matter which transformation is done first, which can be verified by realizing that reflecting across the x-axis and then the y-axis or reflecting across the y-axis and then the x-axis both result in the mapping:

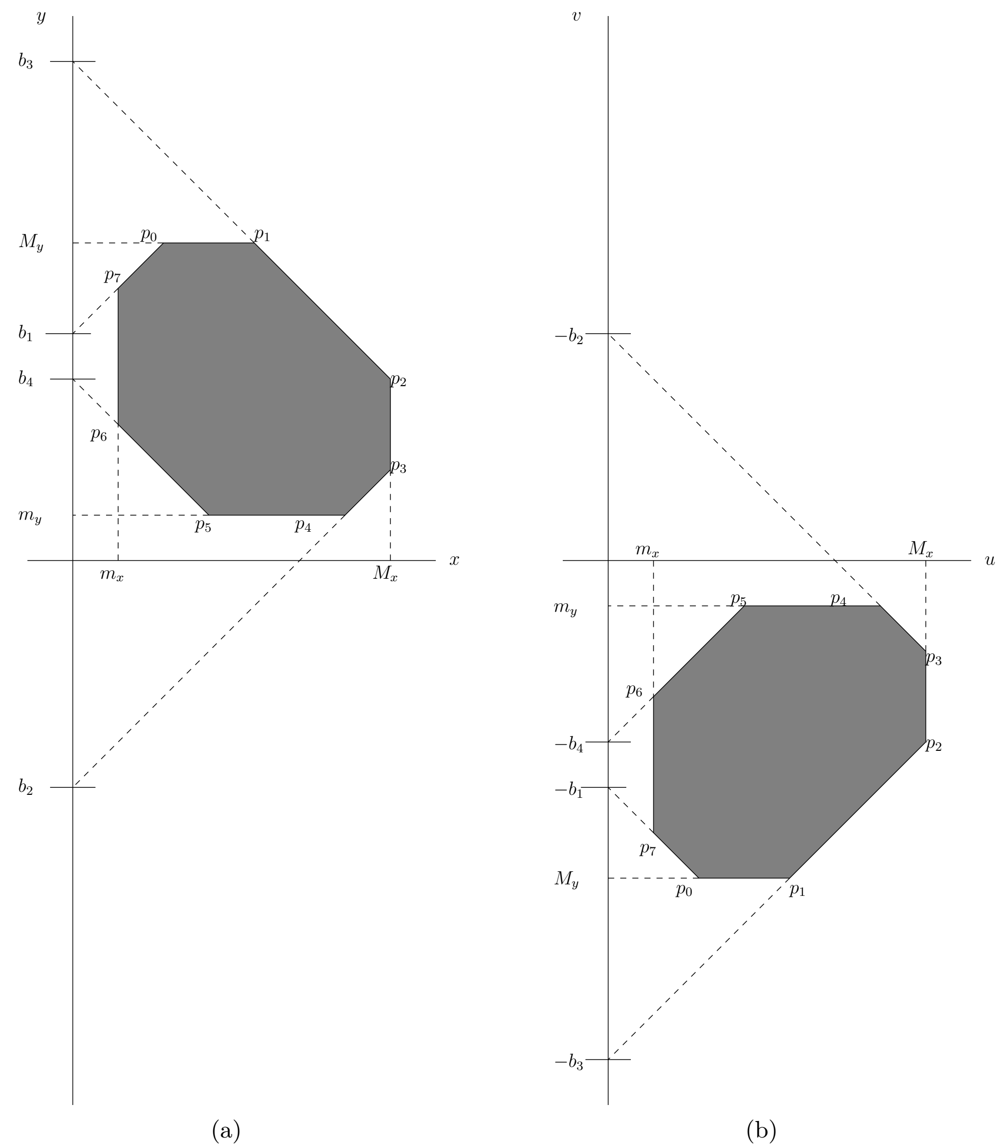


Figure C.6: The effect of the coordinate transformation $u = x$ and $v = -y$. (a) The original octagon. (b) The octagon after the coordinate change.

$$m_x \mapsto -M_x \qquad M_x \mapsto -m_x \qquad m_y \mapsto -M_y \qquad M_y \mapsto -m_y$$
$$b_1 \mapsto -b_2 \qquad b_2 \mapsto -b_1 \qquad b_3 \mapsto -b_4 \qquad b_3 \mapsto -b_4.$$

A graphical illustration of this fact is shown in Fig. C.7.

After having these mappings in place, handling the more general case is simply a matter of applying the appropriate transformation for rates ± 1 and then applying the warping equations for $|\alpha|$ and $|\beta|$.

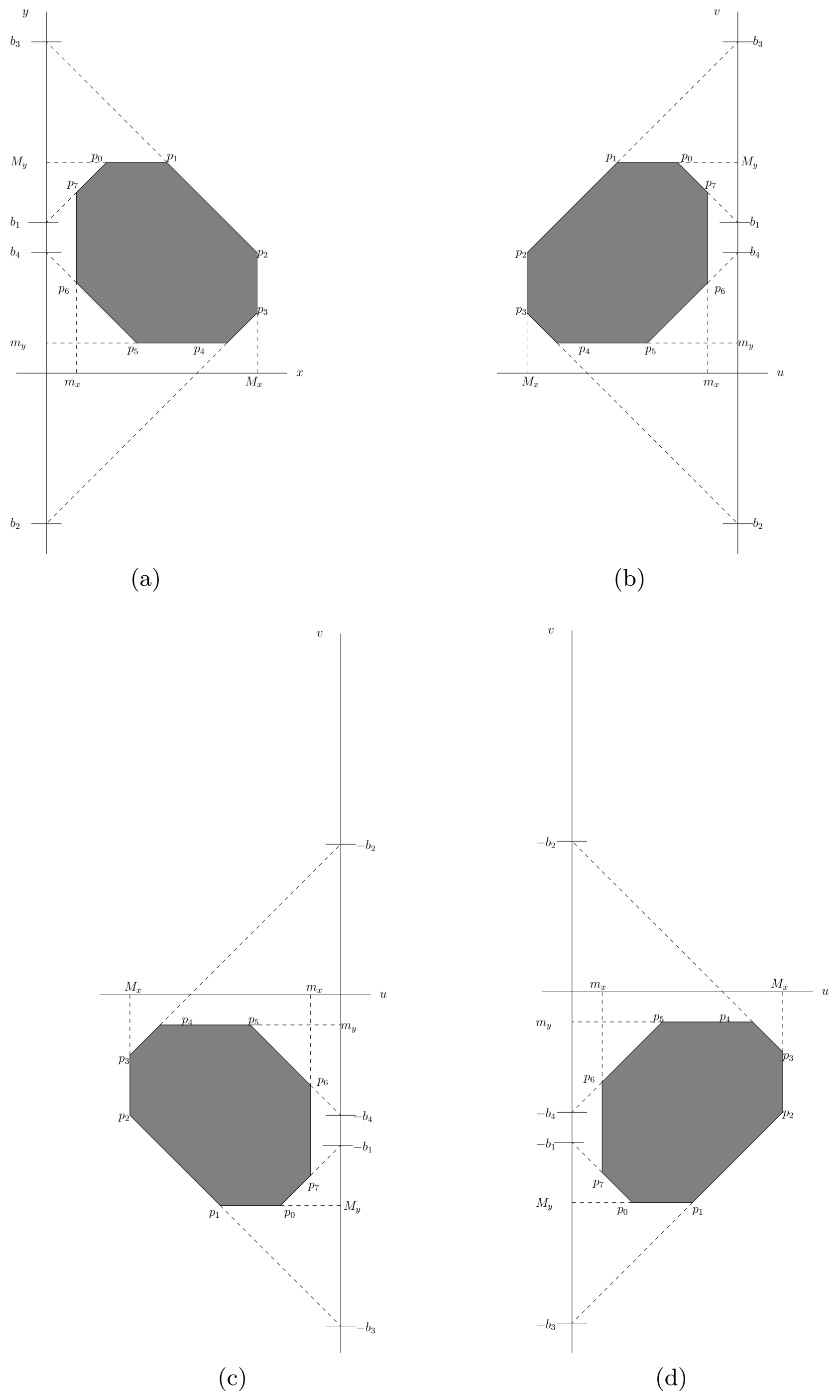


Figure C.7: The effect of the coordinate transformation $u = -x$ and $v = -y$. (a) The original octagon. (b) Reflection across the y-axis. (c) Reflection across the x-axis. (d) Reflection across the line $y = -x$.

REFERENCES

[1] "IEC 62531 ed. 1 (2007-11) (IEEE std 1850-2005): Standard for property specification language (PSL)," *IEC 62531:2007 (E)*, pp. 1–152, Dec 2007.

[2] "IEEE standard for system verilog-unified hardware design, specification, and verification language," *IEEE STD 1800-2009*, pp. C1 –1285, 2009.

[3] S. Akers, "Binary decision diagrams," *Computers, IEEE Transactions on*, vol. C-27, no. 6, pp. 509–516, June 1978.

[4] G. Al-Sammane, M. Zaki, and S. Tahar, "A symbolic methodology for the verification of analog and mixed signal designs," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, april 2007, pp. 1 –6.

[5] G. Al-Sammane, M. H. Zaki, Z. J. Dong, and S. Tahar, "Towards assertion based verification of analog and mixed signal designs using PSL," in *Forum on specification and Design Languages, FDL 2007, September 18-20, 2007, Barcelona, Spain, Proceedings.* ECSI, 2007, pp. 293–298.

[6] R. Alur, "Formal verification of hybrid systems," in *Proceedings of the Ninth ACM International Conference on Embedded Software*, ser. EMSOFT '11. New York, NY, USA: ACM, 2011, pp. 273–278. [Online]. Available: http://doi.acm.org/10.1145/2038642.2038685

[7] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, pp. 3–34, 1995. [Online]. Available: citeseer.nj.nec.com/alur95algorithmic.html

[8] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*, ser. Lecture Notes in Computer Science, vol. 736, 1993, pp. 209–229.

[9] R. Alur, T. Feder, and T. A. Henzinger, "The benefits of relaxing punctuality," *Journal of the ACM*, vol. 43, no. I, pp. 116–146, 1996.

[10] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[11] E. Asarin, T. Dang, and O. Maler, "The d/dt tool for verification of hybrid systems," in *Proceedings International Workshop on Computer Aided Verification*, ser. Lecture Notes in Computer Science, E. Brinksma and K. G. Larsen, Eds., vol. 2404. Springer-Verlag, 2002, pp. 365–370.

[12] R. Bagnara, P. Hill, and E. Zaffanella, "An improved tight closure algorithm for integer octagonal constraints," in *Verification, Model Checking, and Abstract Interpretation*, ser. Lecture Notes in Computer Science, F. Logozzo, D. Peled, and

L. Zuck, Eds. Springer, Berlin, Heidelberg, 2008, vol. 4905, pp. 8–21. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78163-9_6

[13] C. Baier and J. Katoen, *Principles of Model Checking.* The MIT Press, 2008. [Online]. Available: http://books.google.de/books?id=nDQiAQAAIAAJ

[14] F. Balduzzi, A. Giua, and G. Menga, "First-order hybrid petri nets: A model for optimization and control," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 4, pp. 382–399, 2000.

[15] A. Balivada, Y. Hoskote, and J. Abraham, "Verification of transient response of linear analog circuits," in *VLSI Test Symposium, 1995. Proceedings., 13th IEEE*, Apr 1995, pp. 42–47.

[16] B. Barbot, T. Chen, T. Han, J.-P. Katoen, and A. Mereacre, "Efficient CTMC model checking of linear real-time objectives," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, P. Abdulla and K. Leino, Eds. Springer, Berlin, Heidelberg, 2011, vol. 6605, pp. 128–142. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-19835-9_12

[17] S. Batchu, "Automatic extraction of behavioral models from simulations of analog/mixed-signal (AMS) circuits," Master's thesis, University of Utah, Salt Lake City, UT, USA, 2010.

[18] W. Belluomini and C. J. Myers, "Timed circuit verification using tel structures," *IEEE Transactions on Computer-Aided Design*, vol. 20, no. 1, pp. 129–146, Jan. 2001.

[19] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on Concurrency and Petri Nets*, ser. Lecture Notes in Computer Science, J. Desel, W. Reisig, and G. Rozenberg, Eds. Springer, Berlin, Heidelberg, 2004, vol. 3098, pp. 87–124. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-27755-2_3

[20] M. Boulé and Z. Zilic, "Automata-based assertion-checker synthesis of PSL properties," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, no. 1, pp. 4:1–4:21, Feb. 2008. [Online]. Available: http://doi.acm.org/10.1145/1297666.1297670

[21] R. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. C-35, no. 8, pp. 677–691, Aug 1986.

[22] L. Bu, Q. Wang, X. Chen, L. Wang, T. Zhang, J. Zhao, and X. Li, "Toward online hybrid systems model checking of cyberphysical systems time-bounded short-run behavior," *ICCPSâĂŹ11 Work-in-Progress Session*, 2011.

[23] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10^{20} states and beyond," in *IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 1990, pp. 428–439. [Online]. Available: citeseer.nj.nec.com/burch90symbolic.html

[24] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill, "Symbolic model checking for sequential circuit verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 4, pp. 401–424, Apr. 1994.

[25] F. Cassez and K. Larsen, "The impressive power of stopwatches," in *CONCUR 2000 – Concurrency Theory*, ser. Lecture Notes in Computer Science, C. Palamidessi, Ed. Springer, Berlin, Heidelberg, 2000, vol. 1877, pp. 138–152. [Online]. Available: http://dx.doi.org/10.1007/3-540-44618-4_12

[26] E. M. Clarke and R. P. Kurshan, "Computer-aided verification," *IEEE Spectrum*, pp. 61–67, Jun. 1996.

[27] E. Clarke, O. Grumberg, and D. Peled, *Model Checking.* Mit Press, 1999. [Online]. Available: http://books.google.com/books?id=Nmc4wEaLXFEC

[28] J. Crossley, A. Puggelli, H.-P. Le, B. Yang, R. Nancollas, K. Jung, L. Kong, N. Narevsky, Y. Lu, N. Sutardja, E. An, A. Sangiovanni-Vincentelli, and E. Alon, "BAG: A designer-oriented integrated framework for the development of ams circuit generators," in *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, Nov 2013, pp. 74–81.

[29] T. R. Dastidar and P. P. Chakrabarti, "A verification system for transient response of analog circuits," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, pp. 31:1–31:39, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1255456.1255468

[30] D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," in *Automatic Verification Methods for Finite-State Systems*, J. Sifakis, Ed., vol. 407. Springer-Verlag, 1990.

[31] Z. J. Dong, M. Zaki, G. Al-Sammane, S. Tahar, and G. Bois, "Checking properties of pll designs using run-time verification," in *Microelectronics, 2007. ICM 2007. Internatonal Conference on*, Dec 2007, pp. 125–128.

[32] A. Donzé, T. Ferrère, and O. Maler, "Efficient robust monitoring for STL," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, N. Sharygina and H. Veith, Eds. Springer, Berlin, Heidelberg, 2013, vol. 8044, pp. 264–279. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39799-8_19

[33] A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, and S. Smolka, "On temporal logic and signal processing," in *Automated Technology for Verification and Analysis*, ser. Lecture Notes in Computer Science, S. Chakraborty and M. Mukund, Eds. Springer, Berlin, Heidelberg, 2012, pp. 92–106. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33386-6_9

[34] V. Dubikhin, D. Sokolov, and A. Yakovlev, "Synthesis and verification of mixed-signal systems with asynchronous control," Newcastle Poster, 2015.

[35] C. Eisner and D. Fisman, *A Practical Introduction to PSL*, ser. Integrated Circuits and Systems. Springer, 2007. [Online]. Available: http://books.google.com/books?id=zyiTmbanAAUC

[36] G. Fainekos, A. Girard, and G. Pappas, "Temporal logic verification using simulation," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, E. Asarin and P. Bouyer, Eds. Springer, Berlin, Heidelberg, 2006, vol. 4202, pp. 171–186. [Online]. Available: http://dx.doi.org/10.1007/11867340_13

[37] A. N. Fisher, D. Kulkarni, and C. J. Myers, "A new assertion property language for analog/mixed-signal circuits," in *Languages, Design Methods, and Tools for Electronic System Design*, ser. Lecture Notes in Electrical Engineering, M.-M. Louërat and T. Maehne, Eds. Springer International Publishing, 2015, vol. 311, pp. 45–65. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-06317-1_3

[38] A. N. Fisher, C. J. Myers, and P. Li, "Ranges of rates," 2013, [Online; accessed 31-December-2014]. [Online]. Available: http://www.async.ece.utah.edu/~andrewf/ranges_of_rates.pdf

[39] G. Frehse, C. L. Guernic, R. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, and T. Dang, "Spaceex: Scalable verification of hybrid systems," in *Proceedings of the International Conference on Computer Aided Verification*, 2011.

[40] G. Frehse, B. Krogh, and R. Rutenbar, "Verifying analog oscillator circuits using forward/backward abstraction refinement," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, march 2006, p. 6 pp.

[41] G. Frehse, "Spaceex," 2012, [Online; accessed 31-December-2012]. [Online]. Available: http://spaceex.imag.fr/documentation/publications

[42] G. Frehse, B. H. Krogh, R. A. Rutenbar, and O. Maler, "Time domain verification of oscillator circuit properties," *Electronic Notes in Theoretical Computer Science*, vol. 153, no. 3, pp. 9 – 22, 2006, Proceedings of the First Workshop on Formal Verification of Analog Circuits (FAC 2005). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571066106003422

[43] G. Gardey, O. H. Roux, and O. F. Roux, "State space computation and analysis of time petri nets," *CoRR*, vol. abs/cs/0505023, 2005. [Online]. Available: http://arxiv.org/abs/cs/0505023

[44] M. Geilen, "An improved on-the-fly tableau construction for a real-time temporal logic," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, J. Hunt, WarrenA. and F. Somenzi, Eds. Springer, Berlin, Heidelberg, 2003, vol. 2725, pp. 394–406. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-45069-6_37

[45] D. Geist, "The PSL/sugar specification language a language for all seasons," in *Correct Hardware Design and Verification Methods*, ser. Lecture Notes in Computer Science, D. Geist and E. Tronci, Eds. Springer, Berlin, Heidelberg, 2003, vol. 2860, pp. 3–3. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-39724-3_3

[46] A. Ghosh and R. Vemuri, "Formal verification of synthesized analog designs," in *International Conference on Computer Design.* IEEE Computer Society, 1999, pp. 40–45.

[47] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms.* Kluwer, 1996.

[48] K. Hanna, "Reasoning about analog-level implementations of digital systems," *Formal Methods in System Design*, vol. 16, pp. 127–158, 2000. [Online]. Available: http://dx.doi.org/10.1023/A%3A1008791128550

[49] ——, "Reasoning about real circuits," in *Higher Order Logic Theorem Proving and Its Applications*, ser. Lecture Notes in Computer Science, T. Melham and J. Camilleri, Eds. Springer, Berlin, Heidelberg, 1994, vol. 859, pp. 235–253. [Online]. Available: http://dx.doi.org/10.1007/3-540-58450-1_46

[50] ——, "Automatic verification of mixed-level logic circuits," in *Formal Methods in Computer-Aided Design*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and P. Windley, Eds. Springer, Berlin, Heidelberg, 1998, vol. 1522, pp. 133–148. [Online]. Available: http://dx.doi.org/10.1007/3-540-49519-3_10

[51] W. Hartong, L. H., and E. Barke, "On discrete modeling and model checking for nonlinear analog systems," in *Proceedings International Workshop on Computer Aided Verification*, ser. Lecture Notes in Computer Science, E. Brinksma and K. G. Larsen, Eds., vol. 2404. Springer-Verlag, 2002, pp. 401–413.

[52] W. Hartong, L. Hedrich, and E. Barke, "Model checking algorithms for analog verification," in *Design Automation Conference.* Association for Computing Machinery, 2002, pp. 542–547.

[53] W. Hartong, R. Klausen, and L. Hedrich, "Formal verification for nonlinear analog systems: Approaches to model and equivalence checking," in *Advanced Formal Verification*, R. Drechsler, Ed. Springer, US, 2004, pp. 205–245. [Online]. Available: http://dx.doi.org/10.1007/1-4020-2530-0_6

[54] J. Havlicek and S. Little, "Realtime regular expressions for analog and mixed-signal assertions," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, ser. FMCAD '11. Austin, TX: FMCAD Inc, 2011, pp. 155–162. [Online]. Available: http://dl.acm.org/citation.cfm?id=2157654.2157679

[55] L. Hedrich and E. Barke, "A formal approach to nonlinear analog circuit verification," in *International Conference on Computer-Aided Design*, 1995, pp. 123–127.

[56] ——, "A formal approach to verification of linear analog circuits with parameter tolerances," in *Design, Automation and Test in Europe*, ser. DATE '98. IEEE Computer Society, 1998, pp. 649–655.

[57] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *Symposium on Theory of Computing.* Association for Computing Machinery, 1995, pp. 373–382. [Online]. Available: citeseer.nj.nec.com/henzinger95whats.html

[58] ——, "What's decidable about hybrid automata?" *Computer and System Sciences*, vol. 57, pp. 94–124, 1998.

[59] M. Horowitz, M. Jeeradit, F. Lau, S. Liao, B. Lim, and J. Mao, "Fortifying analog models with equivalence checking and coverage analysis," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, June 2010, pp. 425 –430.

[60] Intel. (2014, Aug.) "Transistors to transformations". [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/corporate-information/museum-transistors-to-transformations-brochure.pdf

[61] J.-B. Jeannin and A. Platzer, "dTL^2: Differential temporal dynamic logic with nested temporalities for hybrid systems," in *IJCAR*, ser. LNCS, S. Demri, D. Kapur, and C. Weidenbach, Eds., vol. 8562. Springer, 2014, pp. 292–306.

[62] K. D. Jones, V. Konrad, and D. NiÄDkoviÄĞ, “Analog property checkers: A DDR2 case study,” *Formal Methods in System Design*, vol. 36, pp. 114–130, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10703-009-0085-x

[63] C. Kern and M. R. Greenstreet, “Formal verification in hardware design: A survey,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 4, no. 2, pp. 123–193, Apr. 1999. [Online]. Available: http://doi.acm.org/10.1145/307988.307989

[64] R. Koymans, “Specifying real-time properties with metric temporal logic,” *Real-Time Syst.*, vol. 2, no. 4, pp. 255–299, Oct. 1990. [Online]. Available: http://dx.doi.org/10.1007/BF01995674

[65] T. Kropf, *Introduction to Formal Hardware Verification.* Springer, 1999. [Online]. Available: http://books.google.com/books?id=p3xSw3AIlToC

[66] D. Kulkarni, “Formal verification of digitally-intensive analog/mixed signal circuits,” Master’s thesis, University of Utah, Salt Lake City, UT, USA, 2013.

[67] D. Kulkarni, S. Batchu, and C. Myers, “Improved model generation of ams circuits for formal verification,” in *TECHCON 2011*, 2011.

[68] D. Kulkarni, S. Batchu, and C. J. Myers, “Improved model generation of AMS circuits for formal verification,” *2011 Virtual Worldwide Forum for PhD Researchers in Electronic Design Automation*, Nov 2011.

[69] D. Kulkarni, A. N. Fisher, and C. J. Myers, “A new assertion property language for analog/mixed-signal circuits,” in *Specification Design Languages (FDL), 2013 Forum on*, Sept 2013, pp. 1–8.

[70] K. Lata, S. Roy, and H. Jamadagni, “Towards formal verification of analog mixed signal designs using spice circuit simulation traces,” in *Quality Electronic Design, 2009. ASQED 2009. 1st Asia Symposium on*, July 2009, pp. 162 –172.

[71] C. Le Guernic and A. Girard, “Reachability analysis of hybrid systems using support functions,” in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, A. Bouajjani and O. Maler, Eds. Springer, Berlin, Heidelberg, 2009, vol. 5643, pp. 540–554. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02658-4_40

[72] S. Little, N. Seegmiller, D. Walter, C. Myers, and T. Yoneda, “Verification of analog/mixed-signal circuits using labeled hybrid petri nets,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, pp. 617–630, 2011.

[73] S. Little, D. Walter, K. Jones, C. Myers, and A. Sen, “Analog/mixed-signal circuit verification using models generated from simulation traces,” *The Inernational Journal of Foundations of Computer Science*, vol. 21, no. 2, pp. 191–210, 2010.

[74] S. R. Little, “Efficient modeling and verification of analog/mixed-signal circuits using labeled hybrid petri nets,” Ph.D. dissertation, Salt Lake City, UT, USA, 2008, aAI3333598.

[75] O. Maler and D. Nickovic, “Monitoring temporal properties of continuous signals,” in *Proceedings of FORMATS-FTRTFT. Volume 3253 of LNCS.* Springer, 2004, pp. 152–166.

[76] O. Maler, D. Nickovic, and A. Pnueli, "Real time temporal logic: Past, present, future," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, P. Pettersson and W. Yi, Eds. Springer, Berlin, Heidelberg, 2005, vol. 3829, pp. 2–16. [Online]. Available: http://dx.doi.org/10.1007/11603009_2

[77] ——, "From MITL to timed automata," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, E. Asarin and P. Bouyer, Eds. Springer, Berlin, Heidelberg, 2006, vol. 4202, pp. 274–289. [Online]. Available: http://dx.doi.org/10.1007/11867340_20

[78] ——, "Checking temporal properties of discrete, timed and continuous behaviors," in *Pillars of computer science*, A. Avron, N. Dershowitz, and A. Rabinovich, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 475–505. [Online]. Available: http://dl.acm.org/citation.cfm?id=1805839.1805865

[79] O. Maler and D. Ničković, "Monitoring properties of analog and mixed-signal circuits," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 3, pp. 247–268, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10009-012-0247-9

[80] P. M. Merlin and D. J. Farber, "Recoverability of communication protocols," *IEEE Transactions on Communications*, vol. 24, no. 9, pp. 1036–1043, Sep. 1976.

[81] A. Mine, "The octagon abstract domain," in *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, 2001, pp. 310 –319.

[82] S. Mitsch, J.-D. Quesel, and A. Platzer, "Refactoring, refinement, and reasoning: A logical characterization for hybrid systems," in *FM*, C. B. Jones, P. Pihlajasaari, and J. Sun, Eds., vol. 8442. Springer, 2014, pp. 481–496.

[83] B. Murmann, "Digitally assisted analog circuits," *Micro, IEEE*, vol. 26, no. 2, pp. 38 –47, march-april 2006.

[84] C. Myers, *Asynchronous Circuit Design.* John Wiley & Sons, Jul. 2001.

[85] L. W. Nagel and D. Pederson, "Spice (simulation program with integrated circuit emphasis)," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M382, Apr 1973. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html

[86] S. Owre, J. Rushby, and N. Shankar, "PVS: A prototype verification system," in *Automated DeductionâĂŤCADE-11*, ser. Lecture Notes in Computer Science, D. Kapur, Ed. Springer, Berlin, Heidelberg, 1992, vol. 607, pp. 748–752. [Online]. Available: http://dx.doi.org/10.1007/3-540-55602-8_217

[87] C. A. Petri, "Communication with automata," Applied Data Research, Princeton, NJ, Tech. Rep. RADC-TR-65-377, Vol. 1, Suppl 1, 1966.

[88] A. Platzer, *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics.* Springer, 2010. [Online]. Available: http://books.google.com/books?id=xCvB4ACK_qMC

[89] ——, "A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems," *Logical Methods in Computer Science*, vol. 8, no. 4, pp. 1–44, 2012, special issue for selected papers from CSL'10.

[90] ——, “The complete proof theory of hybrid systems,” in *LICS*. IEEE, 2012, pp. 541–550.

[91] ——, “A differential operator approach to equational differential invariants,” in *ITP*, ser. LNCS, L. Beringer and A. Felty, Eds., vol. 7406. Springer, 2012, pp. 28–48.

[92] ——, “Keymaera:a hybrid theorem prover for hybrid systems,” 2012, [Online; accessed 31-December-2012]. [Online]. Available: http://symbolaris.com/info/KeYmaera.html

[93] ——, “Logical analysis of hybrid systems: A complete answer to a complexity challenge,” in *DCFS*, ser. LNCS, M. Kutrib, N. Moreira, and R. Reis, Eds., vol. 7386. Springer, 2012, pp. 43–49.

[94] ——, “Logics of dynamical systems,” in *LICS*. IEEE, 2012, pp. 13–24.

[95] ——, “The structure of differential invariants and differential cut elimination,” *Logical Methods in Computer Science*, vol. 8, no. 4, pp. 1–38, 2012.

[96] A. Platzer and E. M. Clarke, “The image computation problem in hybrid systems model checking,” in *HSCC*, ser. LNCS, A. Bemporad, A. Bicchi, and G. Buttazzo, Eds., vol. 4416. Springer, 2007, pp. 473–486.

[97] A. Pnueli, “The temporal logic of programs,” in *Foundations of Computer Science, 1977, 18th Annual Symposium on*, Oct 1977, pp. 46–57.

[98] A. Puri, V. Borkar, and P. Varaiya, “ϵ-approximation of differential inclusions,” in *Hybrid Systems III*, ser. Lecture Notes in Computer Science, R. Alur, T. Henzinger, and E. Sontag, Eds. Springer, Berlin, Heidelberg, 1996, vol. 1066, pp. 362–376. [Online]. Available: http://dx.doi.org/10.1007/BFb0020960

[99] G. Rosu. (2014, Sep.) “Runtime verification”. [Online]. Available: http://www.runtime-verification.org

[100] A. Salem, “Semi-formal verification of VHDL-AMS descriptions,” in *International Symposium on Circuits and Systems*, vol. 5, 2002, pp. V–333–V–336.

[101] S. Seshadri and J. Abraham, “Frequency response verification of analog circuits using global optimization techniques,” *Journal of Electronic Testing*, vol. 17, no. 5, pp. 395–408, 2001. [Online]. Available: http://dx.doi.org/10.1023/A%3A1012751118746

[102] B. I. Silva and B. H. Krogh, “Formal verification of hybrid systems using checkmate: A case study,” in *American Control Conference*, vol. 3, Jun. 2000, pp. 1679–1683.

[103] B. I. Silva, O. Stursberg, B. H. Krogh, and S. Engell, “An assessment of the current status of algorithmic approaches to the verification of hybrid systems,” in *IEEE Conference on Decision and Control*, vol. 3. IEEE Press, Dec. 2001, pp. 2867–2874.

[104] D. Smith, “Asynchronous behaviors meet their match with SystemVer- ilog assertions,” in *Design and Verification Conference (DVCON)*, 2010.

[105] S. Steinhorst and L. Hedrich, “Model checking of analog systems using an analog specification language,” in *DATE ’08: Proceedings of the Conference on Design, Automation and Test in Europe.* New York, NY, USA: ACM, 2008, pp. 324–329.

[106] L. Tan, J. Kim, and I. Lee, "Testing and monitoring model-based generated program," *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 2, pp. 128 – 148, 2003, {RV} '2003, Run-time Verification (Satellite Workshop of {CAV} '03). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571066104810466

[107] P. Thati and G. Rosu, "Monitoring algorithms for metric temporal logic specifications," *Electronic Notes in Theoretical Computer Science*, vol. 113, no. 0, pp. 145 – 162, 2005, Proceedings of the Fourth Workshop on Runtime Verification (RV 2004). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571066104052570

[108] A. Tiwari, N. Shankar, and J. Rushby, "Invisible formal methods for embedded control systems," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 29–39, Jan. 2003.

[109] A. Tiwari, "Approximate reachability for linear systems," in *International Workshop on Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, O. Maler and A. Pnueli, Eds., vol. 2623. Springer-Verlag, Apr. 2003, pp. 514–525.

[110] A. Tiwari and G. Khanna, "Nonlinear systems: Approximating reach sets," in *International Workshop on Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, R. Alur and G. J. Pappas, Eds., vol. 2993. Springer-Verlag, Mar. 2004, pp. 600–614.

[111] S. Tiwary, A. Gupta, J. Phillips, C. Pinello, and R. Zlatanovici, "First steps towards sat-based formal analog verification," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, Nov. 2009, pp. 1 –8.

[112] D. Walter, S. Little, C. Myers, N. Seegmiller, and T. Yoneda, "Verification of analog/mixed-signal circuits using symbolic methods," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 12, pp. 2223 –2235, dec. 2008.

[113] Z. Wang, N. Abbasi, R. Narayanan, M. H. Zaki, G. Al Sammane, and S. Tahar, "Verification of analog and mixed signal designs using online monitoring," in *Proceedings of the 2009 IEEE 15th International Mixed-Signals, Sensors, and Systems Test Workshop*, ser. IMS3TW '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/IMS3TW.2009.5158695

[114] Z. Wang, M. Zaki, and S. Tahar, "Statistical runtime verification of analog and mixed signal designs," in *Signals, Circuits and Systems (SCS), 2009 3rd International Conference on*, Nov 2009, pp. 1–6.

[115] C. Yan and M. Greenstreet, "Faster projection based methods for circuit level verification," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, 2008, pp. 410–415.

[116] L. Yin, Y. Deng, and P. Li, "Verifying dynamic properties of nonlinear mixed-signal circuits via efficient smt-based techniques," in *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, Nov. 2012, pp. 436 –442.

[117] M. H. Zaki, S. Tahar, and G. Bois, “Formal verification of analog and mixed signal designs: A survey,” *Microelectronics Journal*, vol. 39, no. 12, pp. 1395 – 1404, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026269208002085

[118] M. Zaki, G. Al-Sammane, S. Tahar, and G. Bois, “Combining symbolic simulation and interval arithmetic for the verification of ams designs,” in *Formal Methods in Computer Aided Design, 2007. FMCAD ’07*, Nov 2007, pp. 207–215.

[119] M. H. Zaki, S. Tahar, and G. Bois, “A practical approach for monitoring analog circuits,” in *Proceedings of the 16th ACM Great Lakes Symposium on VLSI 2006, Philadelphia, PA, USA, April 30 - May 1, 2006*, 2006, pp. 330–335. [Online]. Available: http://doi.acm.org/10.1145/1127908.1127984